NISTIR 7078

TIN Techniques for Data Analysis and Surface Construction

Building and Fire Research Laboratory
Gaithersburg, MD 20899

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NISTIR 7078

TIN Techniques for Data Analysis and Surface Construction

Christoph Witzgall
Javier Bernal
Geraldine S. Cheok

January 2004
Building and Fire Research Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

U. S. Department of Commerce
Donald L. Evans, *Secretary*
Technology Administration

Phillip J. Bond, *Under Secretary*
Technology Administration

National Institute of Standards and Technology
Arden L. Bement, *Director*

ABSTRACT

This report addresses the task of meshing point clouds by triangulated elevated surfaces referred to as TIN surfaces. It describes the general features of this approach, and refers to prototype TIN software employed at NIST for research into the analysis of 3D point data acquired by LADAR instrumentation. Inherent in the TIN approach is the establishment of neighbor relations between data points, which provide the basis for various smoothing, filtering and screening procedures. In particular, the TIN structure is utilized to define the boundary of a planar point set. The report provides theoretical background for subsequent documentation of experimental results about the accuracy of LADAR data and their processing by TIN methods.

Key words: Approximation, data structure, Delaunay triangulation, elevated surfaces, interpolation, mesh, planar boundary, point cloud, TIN

CONTENTS

1. INTRODUCTION

NIST is investigating the utility of LADAR (LAser Detection And Ranging) devices for surface recovery with particular emphasis on applications for the construction industry. Several experiments were conducted in order to assess the accuracy of LADAR scans and the associated data processing [Cheok et al. 2000, Witzgall and Cheok 2001, Cheok, Leigh, and Ruhkin 2002]. NIST has developed prototype triangular meshing software in support of that effort. This report aims at providing a general description of the concepts underlying that software.

Contrary to photography, which produces 2D images, LADAR scanning yields collections of 3D points or "point clouds". Several commonly used software packages permit visualization of the 3D point data. These point-by-point displays provide useful images, which can be viewed at different angles and from different vantage points with various options for color coding. While valuable for terrain or object visualization, such point-based images are frequently not suitable for tasks such as determining volumes or exact dimensions of objects encountered, for instance, in object recognition efforts. Such tasks may then call for representations of point clouds by surfaces. In particular, surfaces are indispensable for volume definition and calculation. Volume definition for, say, regularly spaced point clouds may not be based overtly on a particular surface but rather on a specific formula, but any such formula can be traced to an underlying surface representation.

An important approach to providing such surfaces is known as "meshing" a point cloud. Meshing spatial data sets is, therefore, an essential step of many methods for visualization, visibility analysis, ground surface modeling, object segmentation and object recognition. Meshing also plays a role in several techniques of "registration", that is, the task of transforming the coordinates of data points that were collected from separate vantage points into those of a common frame, so as to enable the proper combination of those data sets.

The particular kind of meshing considered in this report is the "Triangulated Irregular Network" or "TIN" [Peucker et al. 1976, 1978]. Since its inception, this technique has become widely accepted as a tool for object modeling and data analysis [e.g., Polis et al. 1995]. This development accelerated with the advent of LADAR technology, which permits the rapid acquisition of large 3D "point clouds", representing complex scenes such as terrain, structures, and various artifacts. At NIST, the development of TIN methods, and ongoing research into their performance, originated with collaborative efforts sponsored by the US Army Corps of Engineers [Mandel, Witzgall, and Bernal 1987, Witzgall and Karalus 1996]. This report summarizes that development.

TIN methods, as considered in this report and along with most other meshing procedures, are based on the "location/elevation" paradigm, where data points are viewed as elevations assigned at locations or "footprints". The meshing surface is correspondingly envisioned as a "rubber sheet" draped over the data points and pulled down over the elevations by gravity, suggestive of a tent held up by tent poles. It is important to realize, that the "elevations" in this context need not represent verticality in the real world. They may, for instance, represent distances in a

specific scanning direction (Fig.1.1).

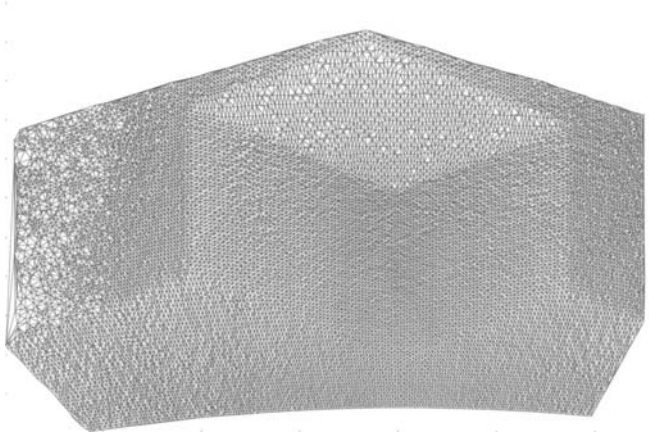


Figure 1.1. A triangulation in a plane perpendicular to the scan direction.

In a natural way, planar meshing of the footprints of a point cloud defines "neighbors" of each individual point, namely, those other points in the point cloud which connect to the point via edges of the triangular meshing. Key tasks of editing and statistically analyzing point clouds require such a notion of "proximity" in order to capture local trends and variabilities.

An alternate way of defining neighbors would be to associate neighbors according to a limit on the distance between their locations. This would amount to defining neighbors by "closeness" instead of proximity. Problems with this approach would become apparent if there are large differences in location density and, therefore, distances to immediate neighbors. Such differences are particularly pronounced if ground-based LADAR is used to scan terrain, and if the data points are given in *x,y,z*-coordinates with *z*-coordinates representing actual verticality.

One popular distance-based approach is to specify a "regular rectangular" grid in the footprint plane, and to collect data points into the "bins" of the grid according to their respective footprints, an approach referred to as "binning". A related application of grids is to reduce irregular point cloud data to regular "gridded" data sets, where elevations are assigned to regular grid or "post" points. Grid–based methods are attractive because of their conceptual simplicity. They are particularly intuitive in image processing, where pixels are naturally arranged in a grid. Binning is also employed as an optional step in NIST TIN routines.

However, grids are built with respect to two special directions, left/right and down/up, usually associated with *x*, *y*-coordinates. Grid-based methods are, therefore, not invariant to rotation of the coordinate system. In some applications, this can cause difficulties with representing lines that are at angles with the coordinate axes. A more significant potential drawback of grid-based methods is their aspect of "one-size-fits-all": they impose uniform resolution by fixing the size of the grid bins. In other words, their notion of proximity is that of closeness, and grid-based methods might, therefore, not be suitable in situations where there are large differences in

location density. In such situations, "adaptive" schemes, which adapt resolution to variations in data density, are then called for. The "quadtree" approach [Samet 1990] extends the grid concept hierarchically in order to respond to varying demands for resolution. It still, however, is tied to two particular grid directions with the concomitant difficulties in line representation.

TIN methods, on the other hand, are adaptive as well as rotation invariant. As pointed out above, they provide a very natural and immediate notion of proximity based on edge adjacency in a triangular mesh. In areas of high location density, edges in the mesh tend to be short, and neighbors close, whereas, in areas of low location density, edges in the mesh tend to be longer, and neighbors further away. In addition, as TIN methods work directly with irregular data sets, they avoid the need to "regularize" such data sets. Such regularization may result in loss of information concerning density patterns, roughness patterns, object edges as well as loss of resolution. Should regularization be indicated, then TIN meshing with subsequent evaluation of the meshed surface at grid points provides a robust and efficient alternative to the familiar "k^{th} - closest points" method for creating gridded data sets.

Real objects are, in general, not "smooth". There may be lines, along which they are not differentiable, that is, their tangential planes are not uniquely defined, such as along terrain "break lines" or "edge lines" of solids. It is important, that a surface representation of objects reflects what will be called "feature lines" in this report. The TIN technique enables the pre-specification of known feature lines, which are to be represented as strings of adjacent edges in the mesh [Bernal 1988]. Also, post-processing adjustments of a triangulation can improve the approximation of the point cloud automatically based on local data patterns.

The utility of TIN methods is restricted by their adoption of the location/elevation or "rubber sheet" paradigm, which requires no two different points of the meshed surface have the same "footprint." In other words, at each location there is a unique elevation. The term "2.5D" has been coined for such surfaces. Meshing of true 3D surfaces with nooks and undercuts thus requires different methods. The most promising approach to fully general 3D triangulated meshing is to "tetrahedralize" the points in space. The actual 3D surface meshing procedure then amounts to a procedure for determining from the tetrahedralization the boundary surface of the 3D point set [e.g., Edelsbrunner and Mucke, 1994, Amenta and Bern 1999, Amenta et al. 2000, Dey and Giesen 2001]. The strength of the tetrahedralization approach is again the fact that it provides a direct and natural neighbor relationship between 3D points. Tetrahedralization will not be discussed in this report. For NIST research into computational aspects of tetrahedralization, the reader may want to consult [Bernal 2001].

The TIN procedures discussed here are based mainly on the Delaunay triangulation principle, to be described in this report. Such procedures are computationally efficient, being in general of complexity $O(n \log n)$.

In the following Chapter 2, the basic concepts of triangulations and TIN surfaces used in this report will be introduced. The method for adaptively triangulating a set of data footprints that underlies the NIST TIN software is presented in Chapter 3. Chapter 4 describes several post processing options. They include defining the boundary of the footprint set of a point cloud

along with various options for boundary editing, as well as surface adjustments and file editing via filters and screens. Flow tables for three prototype TIN algorithms, TINvolume, TINscreen, TINfilter, developed at NIST are provided in Chapter 5, along with an in-depth discussion of pertinent data structures. Also in that Chapter, two alternate philosophies for TIN procedures, the "top down" and the "bottom-up" approach are compared. Chapter 6 examines alternate triangulation paradigms such as the "greedy" triangulation and regular rectangular grids. In Chapter 7, finally, relationships between meshing and spheres are explored.

The development of TIN procedures is an on-going effort at NIST as well as in the field of computational geometry, in general. One of the purposes of this report is to identify additional development and research that is necessary to support current NIST projects. Pertinent research and development subjects are stated at the end of related sections.

1.1 ACKNOWLEDGEMENTS

The work reported here spans almost two decades of development in terrain representation and analysis, initiated by the U.S. Army Corps of Engineers' Topographic Engineering Center (TEC). It profited critically from consultations with Douglas R. Shier of Clemson University. It would not have succeeded without the encouragement, involvement and support by Paul Logan, Stephen DeLoach, Douglas Caldwell, George Lukes, and Anthony Niles. Special thanks are owed to TEC collaborators Betty Mandel [Mandel et al. 1987] and Randall Karalus [Witzgall and Karalus 1991]. The encouragement and support of Drs. S. Shyam Sunder and William C. Stone of NIST's Building and Fire Laboratory has made it possible to build further on that foundation.

2. POINT CLOUDS AND ELEVATED SURFACES

A LADAR device gathers 3D data in the form of a

"point cloud",

that is, a collection of 3D locations represented in an instrument-centered coordinate system. The immediate output will typically be in "polar coordinates", or

"angle, angle, distance" (φ, θ, r),

where the angles φ, θ represent instrument settings, and the distance or "range" r is measured. The polar coordinates are typically converted to "Cartesian coordinates" (x, y, z) centered at the instrument. The coordinate z of a point is interpreted as its elevation, and the projection (x, y) as its location or

"footprint".

In this report, the data points P_i are assumed to be given in those coordinates:

$$P_i = (x_i , y_i , z_i),$$

with footprints $p_i = (x_i , y_i)$.

For the purpose of this report, a particular class of surfaces will be considered. These surfaces are characterized by the following

Single Elevation Property: *No two surface points share the same footprint.*

Clearly, not all surfaces have this property. Surfaces with undercuts, for instance, are excluded. A surface with the single elevation property is referred to as an

"elevated surface", "parametric surface", or "2.5 D surface".

Note also that, if the point cloud contains points with identical footprints, points to be called

"duplicate points"

in this report, and if two such duplicate points have different elevations, then it will not be possible to pass an elevated surface through those points. Dealing with such points will be discussed in Sections 3.1.1 and 4.4.2.

2.1 TRIANGULATED IRREGULAR NETWORKS (TINs)

The methods to be discussed in this report are based on a particular kind of triangular mesh, commonly referred to [Peuker et al. 1976, 1978] as a

"Triangulated Irregular Network" or "TIN".

Here elevated surfaces are constructed with reference to a

"triangulation"

in the footprint plane, usually – but not necessarily – assumed to be horizontal. Such a triangulation covers the footprints by triangles which do not overlap, that is, have no common points other than vertices and along edges, where two triangles join. If elevations z_v are specified at the vertices $q_v = (x_v, y_v)$ of triangles of the TIN, then a

"TIN surface"

is created (Fig. 2.1) that passes through the 3D points $Q_v = (x_v, y_v, z_v)$ by elevating each planar triangle t_k with vertices $(x_{v_1}, y_{v_1}), (x_{v_2}, y_{v_2}), (x_{v_3}, y_{v_3})$ to the 3D triangle T_k with the corresponding vertices

$$(x_{v_1}, y_{v_1}, z_{v_1}), (x_{v_2}, y_{v_2}, z_{v_2}), (x_{v_3}, y_{v_3}, z_{v_3}).$$

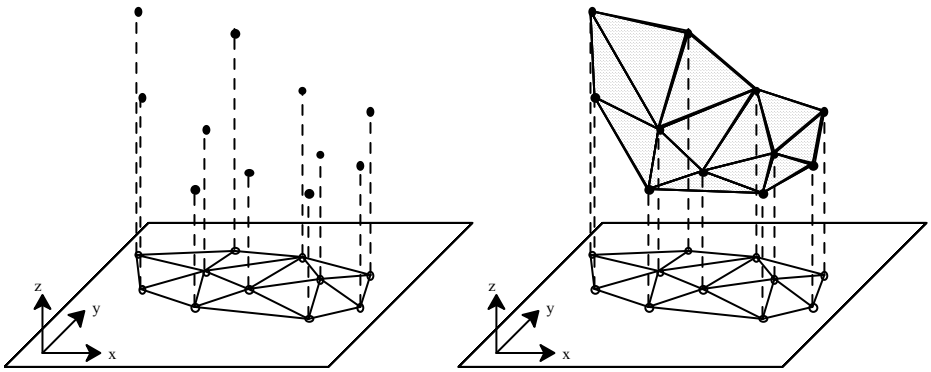


Figure 2.1. A triangulation of data footprints (TIN) and their TIN surface.

The image of "tent poles" of height z_v, supporting the surface at the locations q_v, is appropriate. It also suggests the idea of raising or lowering those tent poles so as to "best" represent the points in the point cloud – a procedure to be discussed in Section 4.2.

So far no assumption has been made about how the footprint points $q_v = (x_v , y_v)$ and their corresponding surface vertices $Q_v = (x_v , y_v , z_v)$, which describe the TIN surface, are to be chosen. The typical TIN approach, which is also adopted here, is to assume that – with minor exceptions –

> *triangulations are based on the footprints* $p_i = (x_i , y_i)$ *of existing data points* $P_i = (x_i , y_i , z_i)$.

However, it should be realized that other choices of footprints, e.g. post points on a regular grid, are possible, provided a method for assigning elevations at these footprint points has been specified.

If all data footprints are included in the triangulation, then the term

"full triangulation"

is used as opposed to

"partial triangulation",

where a selected subset of the data footprints is triangulated. In surface construction, partial triangulations may play a greater role than full triangulations since they permit approximation procedures such as those described in Sections 4.2 and 4.3. Also, computer memory limitations may preclude full triangulations.

2.1.1 Delaunay Triangulation

There are many ways of triangulating a set of planar points and, consequently, there are many TIN surfaces through the corresponding sets of 3D vertices. It would therefore be highly desirable to identify a triangulation concept that, given a set of vertices, defines triangulations that are essentially unique, and that would come close to reproducing the intuitive triangulation an analyst would choose. For instance, the occurrence of very long edges and of associated "skinny" triangles will, in general, be reduced. Obviously, such triangles would distort the appearance of any corresponding TIN surface. Such a triangulation concept exists, and it is based on the following:

> **Empty Circle Criterion:** *No triangulation vertex* $p_v = (x_v , y_v)$ *lies in the interior of the circumcircle of any triangle of the triangulation* [Delaunay 1934].

Such triangulations (Fig. 2.2) are widely considered a standard, employing the terms

"Delaunay triangulation" or "Delaunay TIN,

and for the corresponding TIN surface, the term

"Delaunay surface".

In fact, the term "TIN" is often used synonymously with "Delaunay TIN". A Delaunay triangulation is, in general, uniquely determined by the empty circle criterion, the exception being the case in which the circumcircle of one of the triangles contains more than three vertices on its periphery. In that exceptional case, the vertices on the periphery of such a triangle determine a convex polygon which can be subdivided by diagonals in different ways into triangles, and each of these subdivisions is acceptable within a Delaunay triangulation. An alternate Delaunay criterion based on angles in the triangulation avoids some of those ambiguities. The reader may want to consult the standard textbook by Edelsbrunner [1987].

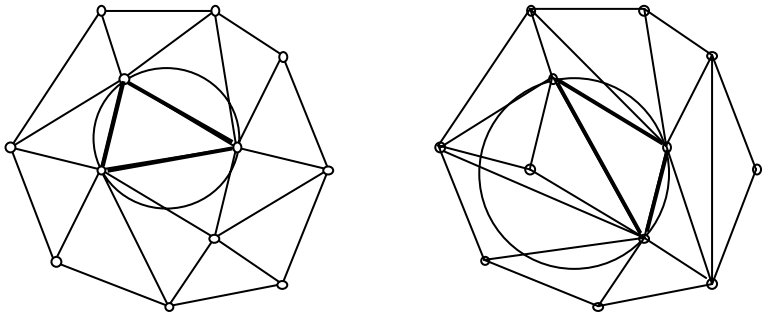


Figure 2.2. Two triangulations of the same set of points. The one on the left is Delaunay.

The existence of a Delaunay triangulation follows from its construction as the "dual" of the "Voronoi diagram" of a given vertex set (Fig. 2.3).

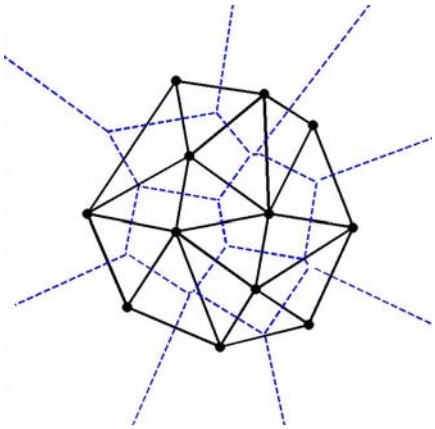


Figure 2.3. Voronoi diagram and its associated Delaunay triangulation.

Each vertex of the triangulation is surrounded by a "Voronoi cell", which consists of all planar points that are closer to this vertex than to any other one. Two vertices are then connected by an edge in the associated Delaunay triangulation if, roughly speaking, the cells in which the vertices are located border on each other. Fortune [1987] computes a Voronoi diagram first in order to provide an *O (nlog n)* method for Delaunay triangulation.

Statistics for geometrical parameters of Delaunay triangulations have been derived, and constitute a generalization of order statistics to two dimensions [Stoyan, Kendall, and Mecke 1987]. For a survey of Voronoi-Delaunay techniques see, for instance, [Aurenhammer 1991, Beichl et al. 1996, Bernal 1993].

2.1.2 Constrained Delaunay Triangulation

The concept of a Delaunay triangulation permits an important generalization. Given a set of

"constraint edges",

namely, a set of straight line segments in the footprint plane, a

"constrained Delaunay triangulation"

is required to have those pre-specified constraint edges as edges of the triangulation. The endpoints of the pre-specified constraint edges are necessarily vertices in the resulting constrained triangulation. In addition, there may be other `

"pre-specified vertices",

which also are required to be a part of the desired triangulation.

Constraint edges may not "cross" each other, and may not contain pre-specified vertices. More precisely, the specification of constraint edges and additional pre-specified vertices must satisfy the following:

> **Constraint Condition:** *If two different constraint edges meet, they must meet at a single point, and this point must be an endpoint of both edges. Additional pre-specified vertices may not meet any constraint edge, nor duplicate the footprint of another pre-specified vertex.*

Given a set of constraint edges, a vertex is considered

"visible"

from some specified point, if its view is not obstructed by one of the constraint edges. Constrained Delaunay triangulations are then characterized by the following

> ***Constrained Empty Circle Criterion:*** *No triangulation vertex* $p_v = (x_v , y_v)$ *lies in the interior of the circumcircle of any triangle of the triangulation and is also visible from the interior of that triangle* [Bernal 1988].

Constrained Delaunay triangulations are again essentially unique. The only exceptions occur if the periphery of the circumcircle of a triangle contains additional vertices that are visible from the interior of the triangle.

If there are pronounced valleys or ridge lines in the scene, and if their locations are known, then constructing a triangulation in which triangle edges follow these lines, assures that they are properly represented by the resulting TIN surface. Specifying line segments that follow such crease lines and constraining triangulations to contain these line segments will serve this purpose.

2.2 INTERPOLATING AND APPROXIMATING DATA

There are two paradigms for representing a point cloud by a surface: interpolation and approximation. Interpolation requires that the surface representing the point cloud actually pass through every point of the cloud exactly whenever possible. Approximation, on the other hand, does not attempt to reach every point. Interpolation might be considered the most conservative approach to constructing TIN surfaces from given elevation data points. If, however, the purpose of interpolation is to construct a representative surface for a point cloud, caution is warranted. Indeed, in this approach, data points arising from artifacts, process-malfunctions, or other outliers are accommodated in the same way as all the other data points. In addition, measurements are usually subject to noise, that is, random statistical deviations from the "true" value. If a point cloud is dense as well as noisy, this may also present a situation in which interpolation may not be desirable, unless it has been preceded by screening or editing procedures, which amounts to de facto approximation. Finally, if there are many data points, the interpolating TIN surface structure may be too cumbersome to use. Interpolation is useful when the objective is to identify outliers, gauge noise, or implement local filtering procedures as discussed in Chapter 3.

Approximation may simply amount to interpolation of a specified subset of a point cloud. That, of course, begs the question of "how good" is such an approximation, or any other approximation for that matter. To gauge the proximity of the surface to the given point cloud a

"measure-of-fit"

is commonly stipulated. Then it becomes natural to search for "best" approximations, that is, approximations which in some way minimize that measure-of-fit. Having settled on a measure-of-fit to be minimized, the set of candidate surfaces from which the "best" approximation is to be

selected has to be agreed upon, for instance, the set of all TIN surfaces belonging to the same footprint triangulation. This is the approach followed here. More specifically, the triangulation will be a partial triangulation of the footprints of the data cloud. The footprints entering the triangulation will be selected sequentially, as will be described in Chapter 2. Data points corresponding to those footprints are then either (*i*) interpolated, that is, these data points are chosen as the vertices of the TIN surface to be constructed, or (*ii*) the vertices of the TIN surface are chosen at the selected data footprints, but their respective elevations are determined so as minimize the specified measure-of-fit, by which the TIN surface approximates the entire point cloud. Those options will be discussed in Section 4.2.

The results of an approximation procedure depend critically on the choice of the measure-of-fit. Which of those measures-of-fit will give rise to a desirable approximation by a surface? How to recognize desirable approximations? Such issues require what might well be the most urgent research agenda.

2.2.1 Distance Measures

In this section, several measures-of-fit will be identified. They belong to a special class of measures-of-fit that may be called "distance-based", and are most commonly applied. Here, a concept of distance measure is chosen which indicates the closeness to the representing surface of each point $P_i = (x_i , y_i , z_i)$ in the point cloud. The following are natural choices for that kind of distance measure, which need to be explored and evaluated.

- Vertical residual (implemented)
- Euclidean distance to closest point on surface
- Deviation in scan direction

The first option is naturally associated with the 2.5D surfaces at issue. It is based on the notion of the

$$\text{“ residual”} \quad r_i = z_i - \hat{z}_i$$

of the data point $P_i = (x_i , y_i , z_i)$, where $\hat{z}_i$ denotes the elevation of the TIN surface at the footprint $p_i = (x_i , y_i)$. In other words, the point $\hat{P}_i = (x_i , y_i , \hat{z}_i)$ is the surface point that shares a vertical line with the data point p_i. The vertical distance determined in this fashion is then given by the absolute value or "size" of the residual

$$d_i = | r_i | = |z_i - \hat{z}_i| .$$

At steep areas of the surface, the vertical distance expressed by the residual may send a misleading message about the actual proximity of a data point to the approximating surface. Using the actual shortest distance to the surface may be expected to result in a more realistic approximation. Similarly, if the scan direction is known or can be estimated, then measuring the

distance of a data point from the surface along a ray in that direction appears to be particularly attractive because it relates to the physical process of scanning.

2.2.2 Norms for Measures-of-Fit

A measure-of-fit should be a single number "grading" the quality of an approximation. To arrive at such a number, the n individual distances d_i of data points $P_i = (x_i , y_i , z_i)$, as determined according to a suitably chosen common distance measure, have to be combined to arrive at a single number. This is usually accomplished by choosing a

"norm",

namely, a way of assigning a "length" to a vector satisfying proportionality and "subadditivity", that is, the length of the sum of vectors does not exceed the sum of their respective lengths. The following norms are usually considered.

- "Root-Mean-Square (RMS)": $\sqrt{\frac{1}{n}\sum d_i^2}$
- "Average-Size-Deviation (ASD)": $\frac{1}{n}\sum d_i$
- "Maximum Error (MAX)": $\max d_i$

The RMS norm differs only by the factor $\sqrt{n}$ from the so-called "L_2 norm", which is the ordinary Euclidean length of a vector. Similarly, the sum of the absolute values of the components of a vector is its "L_1 norm" and relates to the ASD measure, whereas the size of the largest component is known as the "L_∞ norm ".

In principle, any norm could be applied to any of the distance measures for individual points described earlier. In many cases, minimizing the RMS norm amounts to solving the familiar least-squares problem and is found in most reported applications. The RMS norm may be adversely affected by outliers. The ASD norm is more robust, but harder to compute (see Section 4.2.2). The MAX norm was found not to give useful results since it is dominated by outliers.

Not all data points may call for the same accuracy of proximity: it may be necessary to have a more detailed representation in some areas than in others. The classic way to accommodate such differentiation by importance is to stipulate weights $w_i > 0$ at data points $P_i = (x_i, y_i, z_i)$ – large weights indicating importance – and minimizing, say, one of the following norms

- "weighted RMS norm": $\sqrt{\frac{\sum w_i d_i^2}{\sum w_i}}$

- "weighted ASD norm": $\frac{\sum w_i d_i}{\sum w_i}$

of the distances d_i from the respective TIN surfaces. If all weights are equal, say, $w_i = 1$ for all $i = 1, \ldots, n$, then $\sum w_i = n$, and the previous definitions of the two norms are obtained. Recall that, if the distances are measured vertically, then $d_i = |r_i|$, where r_i denotes the residual of the data point $P_i = (x_i, y_i, z_i)$ with respect to the surface.

While measures-of-fit are of particular importance for registration purposes, they also play roles in the meshing process:

- As post-process statistics, they may be used to indicate how well a TIN surface approximates a point cloud.

- They may suggest adjustments to be applied to a TIN surface so as to improve its representing a point cloud.

- They may be engaged to guide and/or terminate the adaptive selection process for generating a TIN surface.

Most of these possibilities still remain to be examined. In this work, RMS, ASD, and MAX norms of vertical residuals have been implemented for the purpose of statistics to indicate how well the mesh represents the underlying point cloud. For vertical residuals, the RMS adjustment has been implemented and examined (Section 4.2.1).

RESEARCH ITEM: Include various measures-of-fit into the meshing process itself, that is, into key point selection and ranking.

3. COMPUTING TIN SURFACES

The NIST approach to constructing a Delaunay TIN surface, to be described and examined here, is to sequentially select data points as vertices of this TIN surface, constructing the corresponding intermediate surface each time a new vertex has been added. Thus each additional vertex triggers an update of a previously constructed intermediate TIN surface.

As pointed out in Section 2.1, a partial triangulation, corresponding to a surface which contains only a fraction of the vertex pool, is often desired, particularly, if the TIN surface is to approximate rather than interpolate a given point cloud. The above

"bottom-up"

approach to constructing partial Delaunay triangulations has been chosen for this work. Here the selection of each new vertex is based on properties of the previously constructed intermediate TIN surface, a process referred to as "adaptive" selection. The alternate

"top-down",

option is to determine a full Delaunay triangulation first, and then adaptively deleting vertices until a desired partial triangulation has been reached. Those two alternatives are compared in Section 5.4.

3.1 THE INSERTION METHOD FOR DELAUNAY TINs

The methods implemented in this work use the so-called

"insertion method"

(Lawson [1977]) for Delaunay triangulation. For another method, see [Fortune 1987]. The insertion method can add vertices in any specified sequence. If a full triangulation is required, then the sequence of insertion may, for instance, be as the data points appear on an original input file. Other insertion sequences such as constraint information (Section 2.1.2) may be specified as will be described below. Most importantly, if a partial triangulation is required, insertion points can be selected adaptively based on the current intermediate TIN surface in order to attempt to optimize the selection. There are various stopping criteria for the insertion process, in case the goal is not the full triangulation. The insertion process may be terminated if a specified number of vertices has been reached.

In the procedures discussed in this report, all data footprints are enclosed in a rectangular

$$\text{"map"} \ \left\{(x,y):\quad x_{\min} \le x \le x_{\max},\quad y_{\min} \le y \le y_{\max}\right\},$$

and after including its four corners, the entire map will be covered by the triangulation. If there are no data points at those corners, a specified common arbitrary value will be assigned.

It should be realized, that the choice of the map area affects what triangulation remains after the deletion of the map corners. If the map area is sufficiently large relative to the extent of the point cloud, then the remaining triangles will form the convex hull of the data footprints. Otherwise, some of those triangles will disappear along the boundary.

A first – trivial – Delaunay triangulation can be achieved by splitting the map into two triangles along one of its diagonals. Starting with this triangulation, the insertion method will be applied in two phases. During the first phase, the insertion method is used to construct an

"initial Delaunay triangulation."

In this phase, the points to be inserted are determined by criteria that are independent of the current intermediate TIN surface as will be explained in Section 3.3. The second phase starts with the initial Delaunay triangulation and terminates with what will be called the

"selected Delaunay triangulation."

During that second phase, described in Section 3.2, each vertex to be inserted next is selected according to an optimization criterion taking into account the current state of the triangulation. The two phases thus differ only as to whether insertion points are selected adaptively. The user may also decide to forego adaptive selection altogether, say, if a full triangulation is desired so that all possible points are eventually selected anyway. In this case, there will be no second phase, and the initial triangulation is also the complete one.

On the other hand, the user may decide to start adaptive selection immediately with the split map as the initial triangulation. In this case there will be no first phase. In practice, however, a richer initial triangulation is recommended for reasons to be discussed in Sections 3.2 and 3.3.

3.1.1 Inserting a New Vertex

The basic operation of the insertion method is the insertion of a data point $P_i = (x_i , y_i , z_i)$ into the current intermediate TIN surface. This is accomplished by inserting its footprint point $p_i = (x_i , y_i)$ into the current Delaunay triangulation. The insertion proceeds as follows (Fig. 3.1). A triangle t_k containing p_i is determined. If p_i lies in the interior of this triangle t_k , a new triangulation is created by connecting p_i to the corners of t_k , so that there are now three triangles where previously was just triangle t_k (Fig. 3.1a). In the unlikely case that p_i falls on an edge of triangle t_k and if this edge is not part of the boundary of the map, triangle t_k shares that edge with another triangle of the old triangulation. In this case, a new triangulation is created by connecting p_i to the two opposite corners of the above two triangles, so that now four new triangles replace two triangles in the old triangulation (Fig. 3.1b). If the edge containing p_i

is part of the map boundary, then p_i is connected to the opposite corner of triangle t_k, splitting that triangle into only two new ones. Finally, if p_i coincides with a vertex of triangle t_k, then a duplicate point (Section 2.1) has been encountered. Only one of those two points may be included into the TIN. Which of those points is retained as a vertex depends on the particular implementation. In most current NIST routines, the point p_i will be simply passed over.

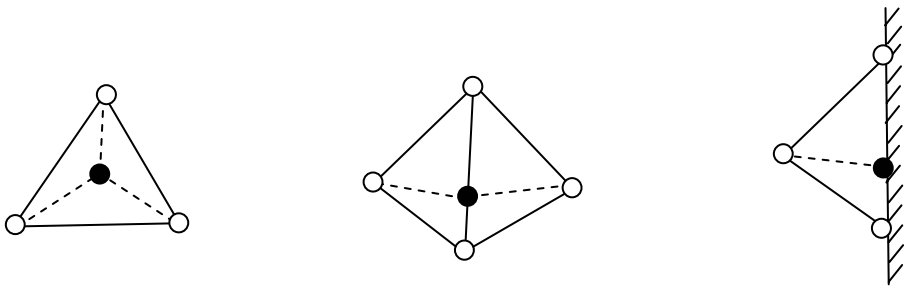


Figure 3.1. The three cases for inserting a new footprint point.

3.1.2 Diagonal Interchanges

The new triangulations generated by the above procedures are, in general, no longer Delaunay triangulations. They will have to be adjusted to meet the empty circle criterion. That adjustment takes the form of repeated

"diagonal interchanges" or "diagonal flips".

Such a diagonal interchange (Fig. 3.2) may be carried

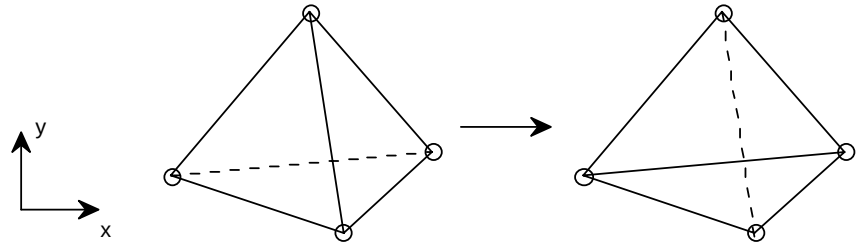


Figure 3.2. Diagonal interchange.

out whenever two triangles of a triangulation are adjacent along an edge and together form a quadrangle whose two diagonals intersect in its interior. One of those diagonals is the common edge of the above triangles. By using the other diagonal to divide the quadrangle, two new triangles are created in the space of the two old ones and become part of an altered triangulation.

Unless an edge *e* of any triangle *t* in the triangulation lies on the map boundary, it will be shared by a unique adjacent triangle. Each such neighboring triangle, furthermore, has a unique vertex *v* which is not an end of edge *e*. Here such vertices will be called

"opposite vertices"

of the triangle. Lawson (1977) has shown that

> *if the empty circle criterion holds for the opposite vertices of each triangle, then it holds universally*.

It is also easy to verify that

> *if an opposite vertex of a triangle t violates the empty circle criterion, i.e., if that vertex lies inside the circumcircle of t, then that vertex together with t defines a quadrangle which permits a diagonal interchange.*

That diagonal interchange will rectify that particular violation of the Delaunay condition. The readjustment algorithm (Fig. 3.3) then is as follows: if the

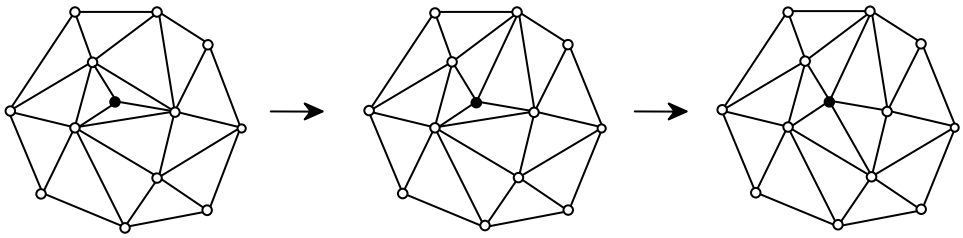


Figure 3.3. Adjustment to achieve a Delaunay triangulation.

opposite vertices of the new triangles created by the insertion do not violate the empty circle criterion, then the new triangulation is already Delaunay by definition. Otherwise, select an opposite vertex which violates the empty circle criterion, and execute the associated diagonal interchange. This creates new triangles whose opposite vertices may or may not violate the empty circle criterion. In the former case, a beneficial diagonal interchange is again triggered. In the latter case, attention shifts to another one of the new triangles.

Lawson (1977) has also proved that the adjustment procedure terminates with a Delaunay triangulation containing the newly inserted point as a vertex.

3.1.3 Determinant Criteria

The mathematical mechanism for deciding whether a given point $p = (x , y)$ lies within the circumcircle of a planar triangle is as follows.

Circle Criterion: *Suppose the vertices*

$$p_1 = (x_1 , y_1) , \; p_2 = (x_2 , y_2) , \; p_3 = (x_3 , y_3)$$

of a triangle are in positive, that is, counterclockwise orientation, that is,

$$\text{determinant} \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix} > 0 .$$

Then the point $p = (x , y)$ *lies within the circle through the vertices* $p_1 , p_2 , p_3 ,$ *if*

$$\text{determinant} \begin{vmatrix} x_1 & x_2 & x_3 & x \\ y_1 & y_2 & y_3 & y \\ x_1^2 + y_1^2 & x_2^2 + y_2^2 & x_3^2 + y_3^2 & x^2 + y^2 \\ 1 & 1 & 1 & 1 \end{vmatrix} > 0 .$$

If that determinant vanishes or is negative, then the point p *lies on the periphery or outside, respectively.*

3.1.4 The Insertion Approach in the Presence of Constraints

If there are pre-specified constraint edges, the insertion method for constructing a constrained Delaunay triangulation (Section 2.1.2) involves now two basic operations: (*i*) inserting a vertex, and (*ii*) inserting one of the constraint edges once its endpoints are already present in the current triangulation. After either insertion step, the resulting triangulation may need to be adjusted in order to reestablish the – constrained – Delaunay property. The two insertion steps are described below.

Inserting a point into the current constrained Delaunay triangulation is identical to the procedure described in Section 3.1.1, except that if the point happens to lie on a previously inserted constraint edge, the insertion of this point is aborted. The readjustment process also requires a slight modification. If a diagonal interchange is indicated, where the current diagonal is a constraint edge, then that diagonal interchange is blocked.

The insertion of a constraint edge e is more involved. One approach, developed by Bernal [1988], is to remove all those previously established edges which are crossed by the new constraint edge e. This creates a polygon divided into two polygonal parts by the insertion of e. These two portions are then triangulated by inserting edges which generate triangles. New triangles are examined for possible Delaunay violations and the corresponding diagonal interchanges are carried out until a constrained Delaunay triangulation has been achieved. NIST TIN routines use an alternate approach described in [Bernal 1995], in which edge crossings are removed by diagonal interchanges. A diagonal interchange that reduces the number of edge crossings may, however, not exist, and must first be established by a sequence of precursor interchanges that keep the number of edge crossings constant. Possible Delaunay violations are removed after the edge e has been inserted.

That edge insertion process is illustrated in Figs. 3.4 and 3.5. The edge e to be inserted runs from point p_1 to point p_6 as indicated by the dotted line. Edge e may not be inserted directly, because it would cross three established edges. The goal is to adjust the triangulation by diagonal interchanges in order to create a situation in which the edge e can be inserted without crossing other edges. In most such situations, there are obvious interchanges which would reduce the number of edge crossings. However, as the example of Fig. 3.4 shows, there are exceptional cases for which such interchanges do not exist at the outset. In those cases, certain precursor interchanges are needed, which keep the number of edge crossings constant but enable other interchanges to reduce the number of crossings. In the example, the interchange of diagonals p_3p_4 and p_2p_5 is such a precursor, creating a triangulation in which there are now two diagonal interchanges, each of which would reduce the number of edge crossings. A resulting sequence of interchanges is depicted in Fig. 3.5. The final exchange involves the edge p_2p_5 and the new constraint edge $e = p_1p_6$.

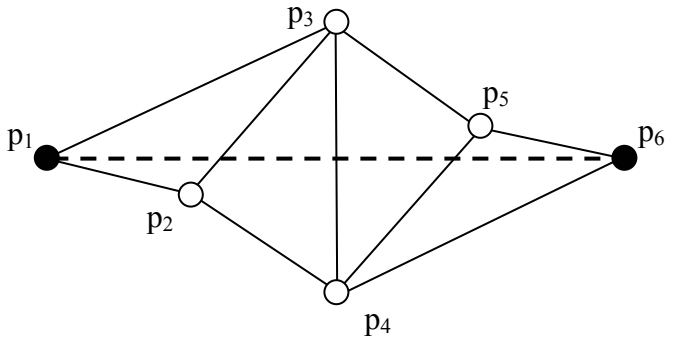


Figure 3.4. Desired insertion of a constraint edge (dotted line).

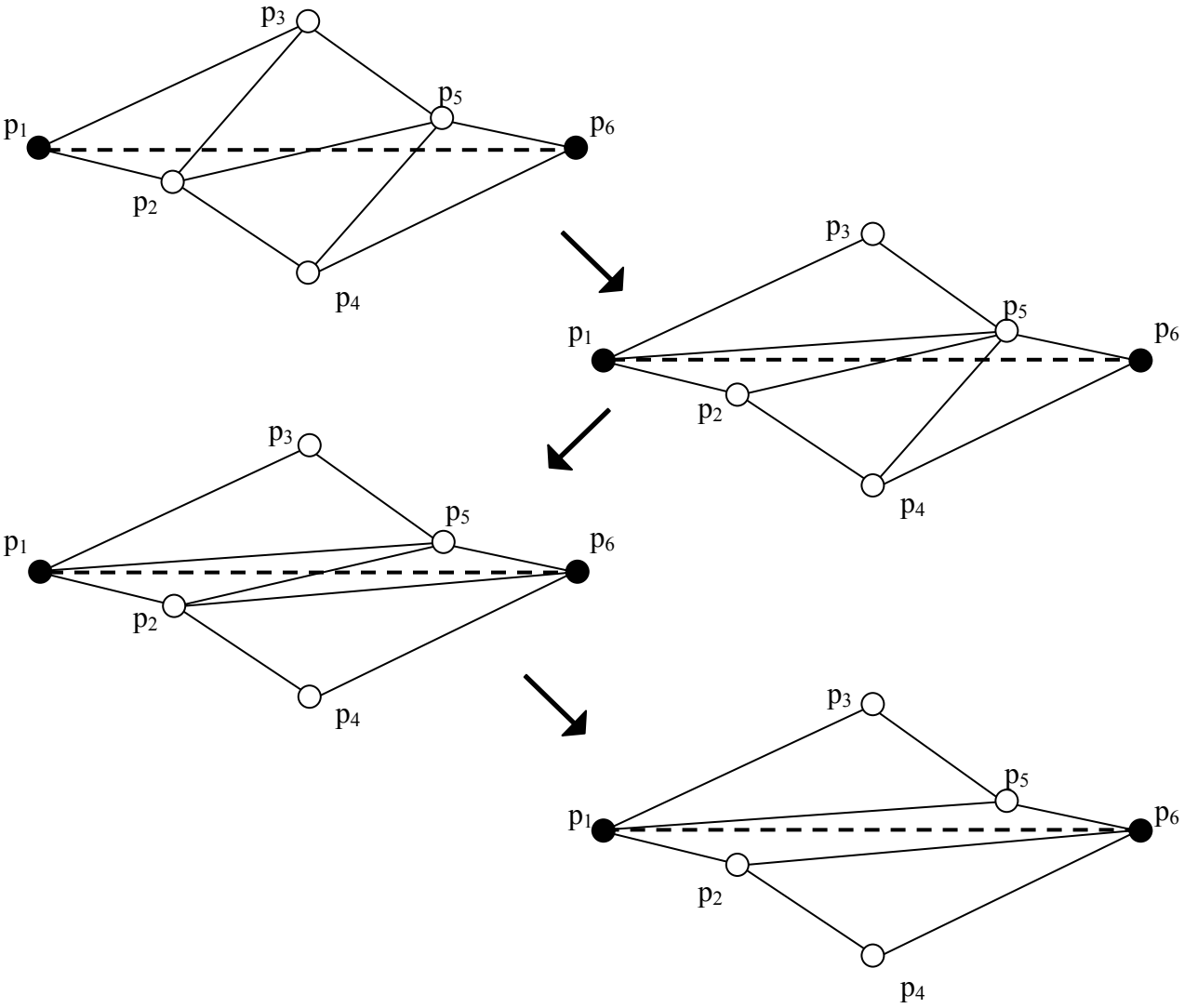


Figure 3.5. Sample sequence of diagonal interchanges for edge insertion. The configuration on top arises from the one in Fig. 3.4 by interchanging the center edge.

The edge insertion approach based on diagonal exchanges has the advantage whenever the edge to be inserted crosses only a few edges, say, one or two. If many edges are crossed, a large number of diagonal interchanges may be required, and it is expected that the other approach, which deletes all crossed edges and re-triangulates the resulting polygons, is to be preferred.

3.1.5 Data Point Locating

Insertion of a data footprint $p_i = (x_i , y_i)$ into an existing triangulation requires determining a triangle that contains that point. This is a major computational problem as the number of triangles in the triangulation may be large so that checking triangles for containment in arbitrary sequence would be prohibitive. In many applications, following method for locating the foot print $p_i = (x_i , y_i)$ of a data point $P_i = (x_i , y_i , z_i)$ in a triangle of the initial triangulation is practical.

> **Triangle Stabbing:** *Suppose some planar point $p^* = (x^*, y^*)$ is known to lie in triangle t^*. Connect the planar points p_i and p^* by a straight line segment* (Fig. 3.6). *As this line segment leaves the triangle t^* it enters a neighboring triangle, which is examined as to whether it contains the footprint of interest p_i. If it does, the search was successful. Otherwise, the line segment is followed to the next triangle it penetrates, and so on, until the triangle which contains the footprint p_i has been reached.*

If the sequence of data points exhibits a pattern by which footprints tend to follow their predecessors closely, then successive data points may frequently occupy the same triangle in the first place or, otherwise lie in a triangle close by so that only few triangles need to be stabbed.

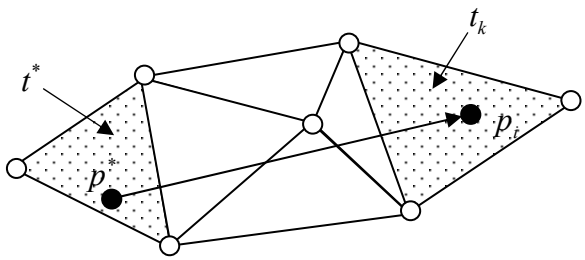


Figure 3.6. Locating footprint p_i by "stabbing" from located point p^*.

The need to determine the triangles that contain given locations arises in many other applications after a triangulation has been determined. The problem of locating a point within a triangulation or, more generally, a polygonal subdivision has been examined early on [Lee and Preparata 1977, Preparata 1981, Kirkpatrick 1983] and is treated in the standard textbook by [Preparata and Shamos 1985].

An alternative to triangle stabbing is to retain the hierarchy of previous triangles [Guibas et al. 1990] and to utilize that information for locating a given point p in its triangle. Hash coding techniques are also frequently employed [see Heckbert and Garland, 1995]. Imposing a grid and linking triangles that intersect each bin is attractive since it is easy to locate a point in a grid cell, and the triangle search can then be restricted to the triangles intersecting this cell. It should be

kept in mind, however, that key point selection criteria typically require access to all data points located in a particular triangle.

RESEARCH ITEM: Systematically compare point location algorithms, particularly, for the case that the data points do not exhibit a sequential pattern. In that latter case, rearranging the data set by binning as proposed in Section 3.3.2 may establish a suitable pattern.

3.2 SELECTIVE INSERTION CRITERIA FOR APPROXIMATION

The bottom-up approach for adaptively selecting a subset of critical data points with the purpose of approximation is at the core of this report. For this reason, phase two of the NIST insertion procedure as outlined above will now be described in detail, with additional information regarding various options and strategy choices. The processes involved in phase one, the generation of the initial Delaunay triangulation, will be described in Section 3.3.

The starting point for phase two is an initial Delaunay triangulation, with a subset of the original data points already selected as vertices. The remaining data points are then examined with respect to the current (constrained) Delaunay triangulation. In each triangle t_k of the triangulation, a

"key point"

is selected among the data footprints p_i which are located in the triangle. For the purposes of this report, only one selection rule will be considered.

> **Key Point Selection**: *Select the point in the triangle for which the size of the residuals* $|r_i| = |z_i - \hat{z}_i|$ *is largest.*

The triangles and their key points are then ranked by a suitable criterion and maintained in a sorted heap accordingly. The highest ranking triangle is located at the top of that list. The key point of this triangle is first in line to be selected as the next critical point to be inserted, unless the process is terminated. The ranking thus determines the sequence in which key points are selected as critical points. Here two alternate criteria for ranking triangles have been implemented.

> **Ranking by Maximum Deviation**: *Rank triangles* t_k *according to the size of the residuals* $|r_i| = |z_i - \hat{z}_i|$ *of the respective key points of the triangles.*

This ranking criterion is geared towards an aggressive reduction of the maximum residual error. The following criterion, however, is more often used in applications of NIST software.

> **Ranking by Volume Deviation**: *Rank triangles* t_k *according to the product* $|z_i - \hat{z}_i| \times area(t_k)$ *of the size of the key residuals and the area of the triangle.*

The product between size of the residual at the key point of the triangle and its area describes up to a factor of 1/3 the volume change that an insertion of the key point would cause, if the current Delaunay surface were considered as bounding a volume. For this reason, the term "ranking by volume deviation" was chosen. When using that criterion, the resulting triangulation tends to be more locally homogeneous as to triangle size, because a large triangle may reach top rank – and be broken up – even if its associated residual is relatively small. Some experience at NIST appears to indicate that the ranking by volume deviation yields better results than ranking by maximum deviation, if the given data are to support volume computation. There are other applications, where ranking by maximum deviation is preferred. One such application is to identify instances of vegetation such as bushes and trees where ranking by maximum deviation causes new vertices to be selected mostly in such areas of high surface variability, namely the areas of bushes and trees.

RESEARCH ITEM: Investigate alternatives to the above key point selection criterion such as choosing data points at which the median of the residuals in their respective triangles is assumed.

3.2.1 Data Linking and Triangle Sorting

As pointed out in Section 3.1.5, insertion techniques face a major computational challenge in trying to find a triangle t_k containing a given footprint point $p_i = (x_i , y_i)$. That computational challenge is even more severe for adaptive insertion techniques. For every triangle, all associated data points have to be examined in order to determine the key point of the triangle.

Various measures taken in this work aim to reduce computational effort. One is the use of linked lists, that is, to

> **link** *all data points* $p_i = (x_i , y_i)$ *to their respective triangles* t_k *in the current triangulation.*

Linking data points to their triangles serves an additional purpose. It provides ready access to the data points associated with a triangle for gathering statistics about data representation.

The following procedure is therefore used. Given the initial Delaunay triangulation, data points are linked in a single process to triangles containing their respective footprints. Concurrent with that process, the key points are established for those initial triangles. Subsequently, as new triangles are generated and others changed during the insertion process, new and updated linkages are continuously required. This process is computationally expensive if the triangles of the emerging triangulation are still large and contain many data points to be linked or re-linked.

The reason for the single process of initial linking of all data points is that it makes it possible to take advantage of the fact that the data points are usually not randomly distributed but are part of a pattern in which the footprints of subsequent data points tend to follow close to each other, so that the point location process is relatively efficient.

The next measure to be taken is to

> **heap-sort** *the triangles, with their key points attached, according to the ranking criterion of choice.*

Again, the heap is first constructed after the process of linking the data points to their associated initial triangles has been completed and, concurrently, key points have been determined for these triangles. Then all initial triangles with associated data points are being sorted into the heap. As new triangles with key points are created during the adaptive phase of the insertion procedure, the heap will have to be continually updated.

The selection procedure then reduces to picking the triangle from the top of the heap, inserting its key point using the insertion procedure described in Sections 3.1.1 and 3.1.2, and updating the linkage structure as well as the heap.

3.2.2 Approximation Statistics

For any TIN surface associated with a partial triangulation created during phase two of the TIN calculations – typically the selected Delaunay triangulation but also the adjusted triangulations described in Sections 4.1 and 4.2 – the linkage structure, which relates data points to triangles, can be used for calculating RMS, ASD, and MAX measures-of-fit as defined in Section 2.2.2. Calculating these measures-of-fit requires determining for each data point $P_i = (x_i , y_i , z_i)$ its residual $r_i = z_i - \hat{z}_i$, where $\hat{z}_i$ denotes the elevation of the TIN surface at the footprint point $p_i = (x_i , y_i)$. Determining this residual requires again locating the footprint $p_i = (x_i , y_i)$ in a triangle t_k of the triangulation. This task is now obviated by the linkage structure, which lists for each triangle its associated data points, enabling the determination of all corresponding residuals with respect to the plane through the elevated triangle T_k .

3.3 CONSTRUCTING AN INITIAL TRIANGULATION

The first step is to construct the trivial Delaunay triangulation of two triangles by splitting the rectangular map (Section 3.1).

The next step is to accommodate constraint information (Sections 2.1.2 and 3.1.4) if such is present. The pre-specified points are inserted first, to be followed by the constraint edges. For each constraint edge, the endpoints need to be part of the triangulation prior to the insertion of the edge itself and have to be inserted if necessary.

The computational effort of the linking/selection process is greatest in the early stages of the procedure when triangles that need to be re-linked contain many data points. For this reason it is not recommended to start with a sparse initial triangulation compared to the anticipated size of the selected triangulation. An additional reason is that a triangulation represented by only a few data points does not provide good guidance for the selection of key points and, subsequently, critical points.

This raises the question how to determine those data points which should be included in the initial triangulation. Because of the expense of the point location process, it is important to insert those points in a suitable sequence as explained in Section 3.1.5.

3.3.1 Point Selection by Binning

Of the many ways an initial selection of data points can be achieved, the following has been implemented.

> **Selection by Binning:** *Divide the map area into an almost square grid of bins. In each bin of the grid select the data point P_i whose footprint $p_i = (x_i , y_i)$ lies closest to the center of the bin.*

The initial triangulation is then constructed by inserting the selected points in the order of their bins.

A general strategy for selecting the bin size has not been developed. That selection is left to the user. The following considerations may guide the selection. A large bin size will result in a sparse initial triangulation, which will not appreciably reduce the computational effort. On the other hand, choosing the bin size too small will result in a dense initial triangulation with increased costs of linking and sorting (see Section 3.2.1). If a partial triangulation is desired, then the loss of accuracy due to starting with a triangulation that is too sparse to provide good guidance for the adaptive selection of vertices is to be balanced against starting with a triangulation that is at too advanced a state to realize the advantages of adaptive vertex selection. In a subsequent report [Cheok and Witzgall 2004], an experiment with varying bin sizes will be described.

In this work, bin size is set by specifying the number of bins per row. The number of bins per column is then automatically chosen so as to make the individual bins as square as possible. As each data point $P_i = (x_i , y_i , z_i)$ is read, the bin in which its footprint location resides is found by dividing the coordinates by their respective bin size and then rounding the results to the smallest integer that is not exceeded. This procedure is combined with the search for the data point closest to the center of the bin. A pointer to this data point – if the bin is not empty – is stored in a 2-dimensional array.

3.3.2 Data Rearrangement by Bins

The binning procedure described in the previous section may become memory intensive if many bins are specified. In fact, in some binning applications, there may be many more bins than data points. In the context of generating initial triangulations and for other binning applications, the setting up of a 2-dimensional array of bins can be avoided by preprocessing the data set.

To this end, the data points are read in the original sequence, the two bin numbers are determined as outlined previously, and a new file is created which adjoins those two bin numbers to each data line. This expanded data file is then sorted by those bin numbers as first and second keys. In a final step, the bin numbers are deleted. The resulting file has thus been re-arranged in such a way, that the points of each bin follow each other in sequence before points in the subsequent bin are listed. As the re-arranged file is read, it is easy to sense when the points in a bin have been exhausted, and the next bin is entered. The points in each occupied bin may thus be processed according to any desired criterion as, for instance, closeness to the bin center. The procedure readily extends to binning in three dimensions, where memory problems tend to be more severe than in two dimensions.

Re-arranging a data file by bins may also impose a sequential pattern of the kind that is advantageous in the context of data linking as described in Sections 3.3.1 and 3.3.2.

4. POST PROCESSING ISSSUES

After a selected Delaunay triangulation has been established, several post processing tasks may then be carried out.

First, artificial elevations may have been assigned at the map corners. In that case, the map corners and their adjacent triangles may need to be deleted. More generally, triangles may cover no-data areas such as the inevitable occlusions or "shadows", associated any line-of-sight instrument. It may then be desirable to identify and delete such triangles. Those processes are part of the general task of defining the boundary of a planar data set. That task is often considered a trivial aspect of data preparation, expected to have been completed before any technical data analysis were to be undertaken. Increasingly, however, defining data boundaries has been recognized as a fundamental problem of computational geometry [Edelsbrunner, Kirkpatrick, and Seidel, 1983]. For instance, – as pointed out in the Introduction – the general problem of meshing to create a general 3D surface may be considered as the task of bordering a 3D point cloud. Bordering 2D data points will be discussed in Section 4.1 and its subsections.

Second, once a specified partial triangulation has been reached, it may be desired to improve the quality of the approximation of the given point cloud by the corresponding TIN surface as measured by a specified measure-of-fit of the entire point cloud. The measures-of-fit considered here are the two norm-based measures, RMS and ASD, of vertical residuals in a 2.5D environment. Such "elevation adjustments" will be discussed in Section 4.2 and its subsections.

Third, as detailed in Section 4.3, there are situations in which the entire set of data points indicates that a given partial triangulation does not reflect some of the realities of the point cloud, and that "triangulation adjustments" are called for. Such triangulation adjustments proceed by diagonal flips whenever the corresponding surface modification results in an improved measure-of-fit.

Fourth, TIN surfaces based on full triangulations may be used for smoothing and removal of outliers of point clouds. "Filtering" adjusts the elevations of TIN vertices based on the elevations of their respective neighbors, "Screening" removes outliers as indicated by neighboring elevations. These editing procedures are discussed in Section 4.4.

4.1 BOUNDARY DELINEATION

In the context of this report, the delineation of the footprint region is a primary task. The approach taken is to distinguish

"relevant triangles"

among the footprint triangles t_k of the TIN. All other triangles are then removed, leaving the set of relevant triangles as a first instance of a footprint region. The latter may exhibit various undesirable features, which may require subsequent editing to be discussed later.

The first question is how to determine whether a triangle is indeed relevant. In general, triangles connected to map corners are not relevant. Also, triangles which cover shadows are typically characterized by very long edges or, alternatively, by circumcircles of large diameters. The latter option relates to the concept of "α -shapes" [Edelsbrunner, Kirkpatrick, and Seidel, 1983].

The following choices are therefore offered for designating relevant triangles or, equivalently, deleting irrelevant ones:

- all triangles are relevant
- only triangles meeting map corners are deleted
- triangles containing map corners or with edges longer than a specified edge length are deleted
- triangles containing map corners or with circumcircles of a diameter larger than a specified diameter are deleted.

4.1.2 Boundary Editing

The boundaries of TINs delineated by deleting irrelevant triangles using any of the above basic criteria, however, are typically very irregular (Fig. 4.1).

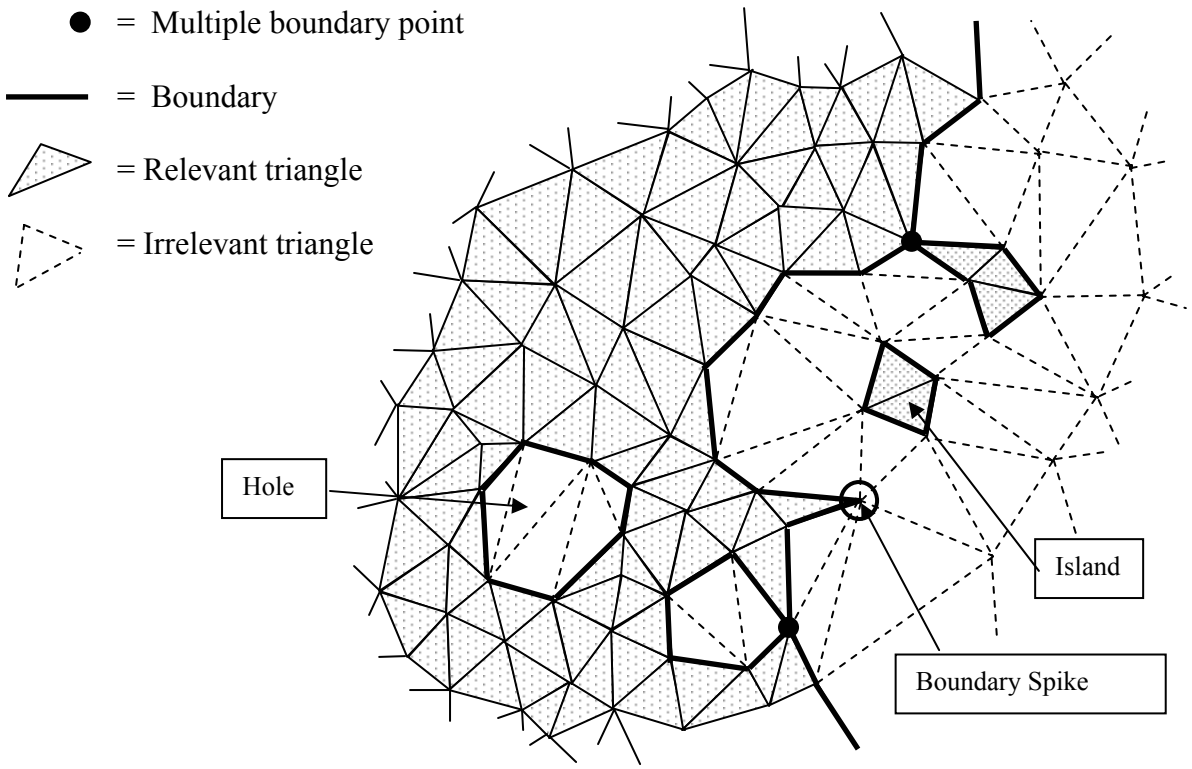


Figure 4.1. Boundary irregularities.

There may be

"boundary spikes",

that is, consecutive boundary edges which form an overly acute angle. Another problem is the occurrence of

"multiple boundary points".

Here four or more boundary edges, instead of the usual two, are incident to the same boundary vertex. Triangles may have been deleted in the interior of a contiguous set of triangles, thus forming a

"hole"

in the triangulation. Such a hole may represent a legitimate feature if, for instance, the deleted triangles describe an occlusion or some other kind of "no-data" area. It may, however, be desirable to fill in holes with only a very small number of deleted triangles. Conversely, it may not be desirable to have small clusters of triangles isolated from the bulk of TIN triangles. Consider two triangles in the TIN to be "connected" if there exists a chain of triangles, any two subsequent ones sharing an edge, which connects the two original triangles. Consider a subset of TIN triangles a "connected component" if any two of its triangles are connected by a triangle chain, and if the set cannot be enlarged without losing connectivity. In many cases, the TIN triangles form one large connected component, comprising the bulk of the triangles, with isolated small clusters, referred to as

"islands",

forming the remaining connected components. In that case, it is an option to consider only triangles in that main component as relevant. Alternatively, one may want to delete islands consisting of fewer than a specified number of triangles. Note that two connected components may still touch each other at isolated vertices. These vertices are necessarily multiple boundary points.

For those occurrences, NIST implemented the following editing options, which either add to or subtract from the set of relevant triangles.

- Delete a boundary spike with an angle smaller than a specified lower bound by deleting the corresponding relevant triangle;
- Disconnect multiple boundary points by deleting triangles from the side that forms the smallest angle;
- Fill holes, if they contain no more irrelevant triangles than a specified lower bound, by re-classifying those triangles as relevant;
- Either eliminate islands by restricting relevant triangles to the largest connected component, or delete islands if they consist of fewer than a specified number of triangles by marking those triangles as irrelevant.

The above operations are not independent in that each corrective action, except the last one, may cause the need for additional corrective measures, and it also may render some measures unnecessary. For instance, filling the hole at the lower end of Fig. 4.1 will also result in the removal of a multiple boundary vertex. Disconnecting a multiple boundary vertex such as the one in the upper portion of Fig. 4.1 may – under different circumstances – generate an acute boundary spike. Boundary editing procedures should, therefore, be applied repeatedly.

RESEARCH ITEM: Determine preferred order of executing the above editing operations.

4.2 ELEVATION ADJUSTMENTS

A first approach to improving the approximation of a particular point cloud is to adjust the elevations at the TIN vertices once a partial triangulation has been constructed having used the procedures outlined in Chapter 3, and afterwards, the procedures outlined in Section 4.1. The task considered now is to adjust the elevations at the vertices of the triangulation so as to minimize either RMS or ASD. These optimization processes are referred to respectively as either

- RMS elevation adjustment, or
- ASD elevation adjustment.

It is important to realize, that it is the approximating TIN surface and not the underlying point cloud that is adjusted by changing the elevations at the vertices of the TIN surface. Any original data point, which gave rise to an adjusted vertex, remains a member of the point cloud, with its given elevation unchanged, and still contributes, like any other data point, to the calculation of the measure-of-fit upon which the approximation is based.

As the construction procedures of the previous chapters are assumed, each TIN vertex corresponds to a given data point. For an actual elevation adjustment to occur at a vertex v, there needs to be at least one data point nearby, which is not a vertex but has a footprint that lies in a triangle adjacent to vertex v, but not on an edge opposite to v in that triangle. Unless there exists such a data point nearby, the optimal elevation of the TIN vertex will always be the elevation of the associated original data point. In particular, if the triangulation is a full triangulation with all data points used as vertices, then elevation adjustments are moot.

In more general situations, not part of this work, the vertices of a specified TIN surface may not correspond directly to data points of the point cloud which is to be approximated. The footprints of the TIN surface may, for instance, form a regular grid, whereas the data points are irregularly spaced. Techniques similar to the elevation adjustments discussed below can be used in order to optimize elevations for the vertices of that surface.

4.2.1 The RMS Elevation Adjustment

The RMS adjustment amounts to solving a standard "least squares problem" for which many highly developed algorithms are available. The current implementation relies on a simple iterative scheme. Note that a change of elevation at a vertex *v* of the TIN surface affects the surface only above the area formed by the triangles adjacent to vertex *v*. This suggests adjusting the elevation at this vertex so that the resulting surface change minimizes the RMS above that area. Carrying out such adjustments in turn for every vertex constitutes an iteration step. For infinitely many steps, the vertex elevations will converge to limits which define the unique "best" TIN surface over the given fixed triangulation and with respect to the RMS norm of vertical residuals. The convergence follows from the fact that the method can be viewed as the minimization of a positive definite quadratic form by, in turn, minimizing each variable by itself while keeping the other variables fixed. Such a process is well known to converge.

For practical purposes, the process is terminated after (*i*) a preset number of iterations have been reached, or (*ii*) all individual elevation adjustments during an iteration step have remained below a specified tolerance.

All methods for minimizing straight RMS also work for weighted RMS with only minor modifications.

4.2.2 The ASD Elevation Adjustment

The ASD elevation adjustment differs from the RMS elevation adjustment described above only in that the ASD norm replaces the RMS norm. The aim is thus to minimize

$$\frac{1}{n}\sum |\, r_i|,$$

where r_i is again the vertical residual of the point $(x_i\,,\, y_i\,,\, z_i)$ with respect to the TIN surface. Finding that optimal ASD adjustment is, however, more involved. It is an example of a "linear programming problem" [see, for instance, Gass 1985]. While there are powerful computer codes available for solving general linear programming problems, the approach pursued in the special case at issue here is to use a sequence of weighted least squares minimizations. The underlying idea is based on the observation, that if one were to have advance knowledge of the optimal ASD adjustment and its respective residuals r_i, then its value could be actually restated as a weighted sum of least squares:

$$\frac{1}{n}\sum |\, r_i\,| = \frac{1}{n}\sum_{i:\, r_i \neq 0} \frac{r_i^2}{|\, r_i\,|},$$

where, in the summation on the right , the terms with $r_i = 0$ have been removed in order to avoid division by zero.

The ASD-optimal TIN surface and its corresponding residuals are, of course, not known in advance. However, those residuals may be estimated using a straight RMS adjustment to begin with, then repeating the RMS adjustment with weights

$$w_i = \frac{1}{|\tilde{r}_i|},$$

where the $\tilde{r}_i$ are the residuals with respect to that weighted RMS optimization. Again, care must be taken to avoid divisions by zero, say, by setting a lower limit for the size of those residuals from which weights are derived, and using a constant positive weight for data points with smaller residuals.

The TIN surface resulting from the RMS adjustment with the above weights then gives rise to a new and, hopefully, better set of weights for a subsequent weighted RMS adjustment. This process is then repeated, with suitable adjustments of the minimum size limits for those residuals. The iteration is stopped, if the ASD value no longer improves within a specified tolerance. The theoretical foundation of this iteration algorithm is not yet fully established. For an experimental examination see [Bernal and Witzgall 1999].

DEVELOPMENT ITEM: Complete ASD elevation adjustment option.

4.3 TRIANGULATION ADJUSTMENTS

There are situations in which Delaunay triangulations may not be desirable. This occurs, for instance, if – in terrain parlance – an edge of a Delaunay triangle crosses a valley or tunnels through a ridge (Fig. 4.2). If the course of the valley or of the ridge is known, then it may be described by a sequence of straight line segments in the footprint plane, and the Delaunay triangulation may be constrained (see Section 2.1.2) to contain those segments as edges. Otherwise, such problems remain mostly undetected, unless there are sufficiently many data points in the region of the affected triangles. For such cases, the following procedure for adjusting the triangulation has been developed.

Whenever, a strictly convex quadrangle consisting of two adjacent triangles is encountered, it can be checked, whether interchanging the diagonals (Fig. 4.2) improves a measure-of-fit (RMS or ASD) within the quadrangle. The sequence in which the adjustments are carried out matters. In the NIST implementation, the diagonal interchanges with the biggest improvements are carried out first.

RESEARCH ITEM: Investigate the interplay between elevation adjustments and triangulation adjustments. Should triangulation adjustments precede elevation adjustments or vice versa? Should elevation adjustments be repeated after triangulation adjustments?

a. Does not retain valley line

b. Retains valley line

Figure 4.2. Adjusting triangles crossing valley.

4.4 FILTERING AND SCREENING

This section, as the previous ones, addresses post-processing procedures, that is, procedures to be executed after a triangulation has been achieved and its boundary has been defined. The adjustments described in the previous sections essentially reacted to information represented by data points not included as vertices of the TIN surface. In other words, they were geared to partial triangulations (Section 2.1). The procedures addressed in this section, in contrast, are designed for full triangulations, and they are based on the neighbor concept. Also, their purpose is to edit a data file rather than to adjust a TIN surface.

The procedures are based on the resulting neighbor relations between the vertices of full triangulations, namely,

> *data points* $P_i = (x_i , y_i , z_i)$ *and* $P_j = (x_j , y_j , z_j)$ *are neighbors if footprints* $p_i = (x_i , y_i)$ *and* $p_j = (x_j , y_j)$ *are connected by an edge in the full triangulation.*

Two classes of editing procedures will be discussed, and are being referred to as

"filtering" and "screening".

Filtering adjusts individual data points in order to align them with their respective neighbors. Screening aims at removing undesirable data points such as "outliers", which are out of line

compared to the elevations of its neighbors. Screening is provided by the NIST routine *TINscreen*, filtering, by the NIST routine *TINfilter*.

In those procedures, the data adjustment is recorded but not finalized until the adjustment process in completed. This ensures that each individual adjustment will be guided by the original coordinates of the neighboring points. An alternate approach would be to edit the data "on the fly", so that the editing of each data point is affected by the previous editing of the neighboring data points. This approach is not discussed in this report.

DEVELOPMENT ITEM: The process of screening typically needs to be repeated several times for the same point cloud. Rather than rerunning *TINscreen* and thus re-creating the triangulation from scratch, the necessary deletions of vertices should be carried out by adjusting the current triangulation, if desired, and repeat the screening process within a single run of the program.

4.4.1 Planarity Preservation

Gaussian and median filters are frequently implemented by adjusting the elevation z_i of a data point $P_i = (x_i , y_i , z_i)$ to the mean or median, respectively, of surrounding elevations while keeping the coordinates x_i and y_i unchanged. Such filters work if the set of neighbor locations is symmetrically arranged around the location of the data point to be filtered as, for instance, for gridded data points not on the boundary of the grid. In the absence of symmetry, filtering may be counterproductive in that it spoils rather than improves the smoothness with which a point cloud surrounds the surface of an object. If the point to be filtered and its surrounding points lie on a plane, Gaussian filters, for instance, do not preserve that fact if they only adjust the elevation of the data point. In this report, a filtering procedure is called

"planarity preserving"

if, in the case that a data point and all his neighbors lie on a single plane, so does the adjusted data point.

In order to construct planarity preserving elevation filters, the following approach was taken. A plane was fit to the point under consideration and its triangulation neighbors. The elevation at this point is then adjusted so it belongs to that plane. This extends the definition of Gaussian and median filters to the case of irregular surroundings. If the plane is chosen to minimize the RMS of the residuals, a Gaussian filter results. Similarly, if the plane minimizes the ASD of the residuals, a median filter is produced. Screening is also based on deviations from a locally fitted plane. Failure to do so might result in unnecessary removal of data points.

DEVELOPMENT ITEM: Include planarity preservation into current version of *TINscreen*. Also, *TINscreen* is currently based only on ASD, and does not provide an RMS option.

4.4.2 Screening for Duplicate Points

Duplicate points were defined in the opening section of Chapter 2 as data points with identical footprints. Since no two duplicate points can be vertices in a TIN surface, a full triangulation does not necessarily represent all data points in a point cloud. However, only vertices are subject to filtering or screening. Other data points are not affected. In the case of screening, it may happen that a particular TIN vertex is not an outlier and is therefore retained, but another data point, which shares its footprint, is an outlier and should be deleted. This necessary deletion, however, is not carried out because the data point is not a vertex. Similarly, the elevation of a vertex is adjusted during the filtering process, but its duplicate data points – if they exist – retain their elevations even if those elevations are out-of-line. For these reason, the editing routines *TINscreen* and *TINfilter* should not be used in the presence of large numbers of duplicate points. Possible remedies are

- extend the editing to duplicate points which are not vertices;
- examine series of respective duplicate points and select a representative vertex prior to editing as an option to be incorporated into the editing routines;
- implement a stand-alone screening routine for the purpose of selectively removing duplicate points.

The difficulty with the first option is to decide whether or to which extent those additional data points should be counted as surroundings in the filtering/screening process of neighboring vertices.

DEVELOPMENT ITEM: Implement screening procedures for duplicate points.

5. IMPLEMENTATION ISSUES

5.1 SCHEMATIC DESCRIPTIONS OF TIN ROUTINES

Three TIN routines,

TINvolume, TINscreen, TINfilter

are described in tabular form in this section. The *TINvolume* routine is dedicated to determining a TIN surface as set forth in Chapter 3 with the options of footprint delineation and elevation-triangulation adjustments described in Chapter 4. The output options are a regular gridded sampling of the TIN surface, a VRML representation of the TIN surface in 3D as well as a 2D line representation of the triangulation in ADDWAMS format. It also determines cut/fill volumes for specified elevation levels. For a description of the procedure for volume determination, the reader is referred to the subsequent report [Cheok and Witzgall 2003]. *TINscreen* and *TINfilter* implement the file editing procedures outlined in Section 4.4. The latter provides options for both Gaussian and median filters.

Table 5.1. *TINvolume* routine.

Step	Description	Comments
Input	Read input files	• *x, y, z* elevation data • map corners • pre-specified vertices (optional). These points are required by the user to occur among the TIN vertices.
	Specify limit on number of TIN vertices	
	Specify process options	• ranking strategy: max size/ max volume • triangle relevance threshold
	Specify RMS vertex adjustment	• number of passes • termination tolerance
	Specify output	• names of output files • output format: VRML, etc.
Initial triangulation	Split map into initial two triangles	• enter map corners if necessary
	Enter pre-specified points and constraint edges (both optional)	
	Insert points from pre-binning	• bin points are closest to bin centers
Linking	Link data points to triangles and identify their key points	• linked lists connect data points located in the same triangle • key points are determined concurrently.
Ranking	Heap-sort triangles by key point rank	• rank by size of residual • rank by area-weighted residual

Step	Description	Comments
Main iteration - while number of vertices ≤ specified limit	Insert highest ranking key point	
	Make triangulation Delaunay again using diagonal flips	
	Re-linking / re-ranking	• the linking structure is reset for the new triangles. • new key points are determined concurrently • the triangles are re-sorted in accordance with ranking strategy
Delineate footprint region	Determine "relevant" triangles to constitute the "footprint region" of the TIN surface	• delete triangles by edges lengths • delete by circum-diameters
	Apply boundary editing	• remove multiple vertices • remove boundary spikes • remove "holes" • remove "islands"
Adjust Elevations	Adjust elevations at TIN points so as to minimize RMS or ASD.	•
Error Statistics	MAX, RMS, ASD errors	• comparing surface against data points
Volume - for specified elevation level	Calculate cut/fill volumes by prismodial decomposition	

The *TINscreen* routine is based on a full triangulation, that is, it first interpolates all possible data points by a TIN surface. It then finds for each TIN vertex v the median of the elevations of the neighboring vertices, that is, the vertices connected to vertex v by an edge. If the elevation z at vertex v falls outside either a specified upper or lower tolerance bound around the median value, then the data point is considered an outlier and is eliminated. The flow of the routine is exhibited in Table 5.2.

Table 5.2. *TINscreen* routine.

Step	Description	Comments
Input	Read input files	• *x, y, z* elevation data • map corners
	Specify upper/lower outlier tolerances	
	Specify output	• screened point cloud
Insertion method	Enter data points in any order	
Identify outliers - for each TIN vertex	Find median of elevations of neighboring TIN vertices and determine whether TIN vertex is outlier	• A TIN vertex is an outlier if it exceeds the specified upper tolerance above the median, or the lower tolerance below the median.
Eliminate outliers	Construct screened x,y,z-file	• order of data points is preserved

The *TINfilter* routine is based on a full triangulation, that is, it first interpolates all possible data points by a TIN surface. For each vertex *v*, it fits a plane through points adjacent to *v*, and finds a filtered elevation for which the vertex is reduced to that plane. Those filtered elevations are saved until the end when all elevations are replaced by their respective filtered elevations. A choice of Gauss and median filters is offered.

Table 5.3. *TINfilter* routine

Step	Description	Comments
Input	Read input files	• *x, y, z* data points • map corners
	Select type of filter	• Gauss • Median
	Specify output	• name of filtered output file
Triangulation	Insertion method	
Get filtered elevation - for each elevation	Fit filtering plane to vertices determined by adjacency	• L2 (Gauss) • L1 (median)
	Determine elevation $\hat{z}$. The vertex $(x, y, \hat{z})$ lies on filtering plane	
	Save the determined filtered elevation	
Reset to filtered elevation $\hat{z}$ - for each vertex	Construct filtered x,y,z-file	

5.2 DATA STRUCTURES

Key questions about algorithm design center on the "data structures" on which they are based. The issue is what kind of arrays (or files) need to be available in order to execute, and also what information is to be provided, and in what form, as the result of an algorithm. Typically, memory requirements and speed of execution have to be balanced against each other.

In this section, some commonly used data structures are discussed. Memory requirements are assessed and compared using estimates of the number of triangles and edges as a function of the number of vertices. Such estimates are derived in Section 5.2.1 below.

Many applications need the data structures that enable the convenient retrieval of the adjacency relationships between vertices, edges and triangles. In fact, the insertion method for creating TIN surfaces discussed in Chapter 3 requires such adjacency information for each of the intermediate surfaces as they are formed. In many applications such as "Geographic Information Systems", attributes need to be associated with vertices, triangles and edges, and should be readily accessible. Examples are elevation, color, shading, or bounding of particular areas, just to name a few. The following discussion will focus on data structures meeting those requirements, which will be formalized in Section 5.2.3.

5.2.1 Estimating the Number of Triangles and Edges

In this work, it was found convenient to work with

"directed edges" or "half-edges",

in other words, to distinguish between the edge leading from, say, vertex v_1 to vertex v_2 , and the edge leading in the opposite direction from vertex v_2 to vertex v_1. In the sense of edges used above, they both occupy the "same" undirected edge, and are referred to as

"mirrors"

of each other. Directed edges distinguish between their origin and their destination. Those roles are reversed for their respective mirror edges. Also, each half-edge belongs to a unique triangle that lies to the left of the half-edge and contains its origin and destination as vertices. On the other hand, each triangle contains three half-edges, which succeed each other so that the destination of any one of the three half-edges coincides with the origin of the next.

In order to assess the memory requirements of triangulation data structures, it is convenient to use the following

> ***Rule of Thumb for the Number of Triangles:*** *The number of triangles in a triangulation is approximately twice the number of vertices, and the number of half-edges is approximately six times the number of vertices:*
>
> $$\#triangles \approx 2 \times \#vertices; \quad \#half\text{-}edges \approx 6 \times \#vertices .$$

For a proof, consider Euler's relation for a convex polytope:

$$\#faces + \#vertices = \#edges + 2.$$

Interpreting a contiguous triangulation as the image of such a 3D polytope with the "outside" of the triangulation as well as possible "holes" counted as faces along with the triangles, the relationship,

$$1 + \#holes + \#triangles + \#vertices = \#edges + 2,$$

results. Replacing the number of edges by the number of half-edges in the above relation, yields,

$$2 + 2 \times \#holes + 2 \times \#triangles + 2 \times \#vertices = \#half\text{-}edges + 4 .$$

In general, each half-edge belongs to a triangle. Exceptions are the "boundary edges" along the outside and also along the holes of the triangulation. On the other hand, each triangle has three edges. Thus

$$\#half\text{-}edges = 3 \times triangles + \#boundary\ edges\ .$$

The above two relations provide two linear equations for two unknowns, namely, the number of triangles and the number of edges, leading to the solution

$$\#triangles = 2 \times \#vertices + 2 \times holes - \#boundary\ edges - 2,$$
$$\#half\text{-}edges = 6 \times \#vertices + 6 \times holes - 2 \times \#boundary\ edges - 6.$$

Since the number boundary edges and holes tends to be small compared to the number of vertices, the above rule of thumb for estimating the number of triangles follows.

5.2.2 Some Conventional Data Structures for Irregular Data

At the most basic level, any irregular 2D triangulation is defined by listing the triangles as given by the coordinates of their respective vertices. In essence, one has a sequence of triangles and/or other information items, each featuring a triangle identifier followed by the *x, y*-coordinates of the triangle vertices in some suitable arrangement.

$$x_1(t),\ y_1(t),\ x_2(t),\ y_2(t),\ x_3(t),\ y_3(t),\quad t = 1, \ldots, \#triangles\ .$$

In order to avoid listing vertex coordinates repeatedly – a vertex typically belongs to several triangles – a separate list of vertex coordinates is typically used. Conveniently, this list also includes the *z*-coordinates of the vertices, so that the following data structure results:

$$x(v),\ y(v),\ z(v),\ \ v = 1, \ldots, \#vertices,\quad v_1(t),\ v_2(t),\ v_3(t),\quad t = 1, \ldots, \#triangles.$$

Here vertex numbers substitute for the coordinate triples of the previous data structure. These vertex numbers provide a table look-up for the coordinates of the corresponding vertices. The ADDWAMS format used in ESRI's ARCINFO[1] system is typical for this approach, which can clearly be appended to describe the 3D triangles of a TIN surface. The device of a separate vertex list is employed by the VRML data structure. Both the ADDWAMS and the VRML data structures are devised for more general constructs than triangles. The key information item is a "line" of consecutive contiguous straight line segments defined as sequences of points. Polygons are represented as lines whose last point duplicates the first point. Thus four points are required to describe a triangle.

The sequence in which the vertices of a triangle are listed defines the "orientation" of the triangle and thus the direction of its surface normal. By general convention,

[1] Certain trade names and company products are mentioned in the text or identified in an illustration. In no case does such an identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

a triangle in the plane is "positively oriented" if its vertices are numbered in counterclockwise rotation.

As to the surface normal of a triangle in general spatial position,

a normal vector is perpendicular to the plane of the triangle, and oriented so that its vertices appear in clockwise order when viewed in the direction of the normal.

In particular, any normal of a positively, i.e. counterclockwise oriented, triangle is parallel to the z-axis of the coordinate system, since viewing in that direction means viewing the plane from below, and the vertices of the triangle are then seen in clockwise orientation. As the triangles of the underlying triangulation are positively oriented, all surface normals of a TIN surface have a positive z-component. Of course, normals cannot be zero, and are often understood to be of length one.

Related to the data structures discussed above is the following

"triangle-based data structure":

$$x(v)\,,\ y(v)\,,\ z(v)\,,\ trefv(v)\,,\quad v = 1, \ldots, \#vertices,$$
$$v_1(t)\,,\ v_2(t)\,,\ v_3(t)\,,\ t_1(t)\,,\ t_2(t)\,,\ t_3(t)\,,\quad t = 1, \ldots, \#triangles.$$

This data structure is used in several TIN routines developed at NIST. Again, the vertices $v_1(t)\,,\ v_2(t)\,,\ v_3(t)$ are positively oriented. The triangles $t_1(t)\,,\ t_2(t)\,,\ t_3(t)$ are the triangles – at most three – which are adjacent along the edges of triangle t. If there are fewer than three, suitable default markers have to be employed. The position of the adjacent triangles should relate to the sequence of vertices of the triangle t. For instance, triangle $t_1(t)$ may be chosen to meet the edge with the ends $v_3(t)\,,\ v_1(t)$. The remaining adjacent triangles are then chosen accordingly.

The look-up function $trefv(v)$, listed above as part of the triangle-based data structure, furnishes a triangle of which v is one of its vertices. Should the vertex v lie on the boundary of the map, then the triangle $trefv(v)$ should border boundary, more precisely, the half-edge of this triangle originating at vertex v should be a boundary edge. The function $trefv(v)$ enables the following key task of characterizing the star of a vertex in the triangulation.

Star Sweep: *For vertex v, determine, in positive rotation, the triangles t_s, half-edges h_s, or vertices v_s such that t_s and v_s are adjacent to vertex v, and h_s originates at vertex v.*

One of the advantages of the above triangle based data structure is its moderate memory requirement:

$$6 \times \#triangles + 4 \times \#vertices = 16 \times \#vertices.$$

In Section 5.2.3, it will be formally established that the triangle based data structure is rich enough to support triangulation algorithms.

The following

"edge-based data structure"

is essentially the one used for the implementation of the TIN routines described in this report. This data structure introduces a separate list structure for half-edges *h,* and directly establishes the arrays

$$\begin{aligned} vdesh(h) &= \textit{destination of half-edge } h \\ tlefh(h) &= \textit{triangle to the left of half-edge } h. \end{aligned}$$

The core of the data structure then consists of the following lists:

vertex list:	$x(v), y(v), z(v),$	$v = 1, \ldots,$ *#vertices*
edge list:	$vdesh(h), tlefh(h),$	$h = 1, \ldots,$ *#half-edges*
triangle list:	$hedgit(i, t), i = 1, 2, 3,$	$t = 1, \ldots,$ *#triangles,*

The edge list functions describe the destination vertex and the triangle to the left. The triangle lists provide the three half-edges of the triangle, that is, the three half-edges for which the triangle is adjacent to their left. Two more functions,

$hmirh(h)$ = "mirror" of half-edge h,
$hrefv(v)$ = "reference" half-edge originating at vertex v,

are required for the manipulations necessary for adaptively generating TINs, but need not be represented by individual arrays. As to the function $hmirh(h)$, the half-edges can be numbered in such a way that each half-edge is next to its mirror: each half-edge with an odd number precedes its mirror, and each half-edge with an even number follows its mirror. The mirror of any half-edge is thus readily determined. The function $hrefv(v)$ assigns to each vertex v a half-edge, which originates at v, and which will be referred to as its "reference edge". This assignment is not unique, and in general there is no particular preference as to which of the originating half-edges to select. The exceptions are the vertices on the boundary of the map. Then it is convenient to select as the reference edge the one whose mirror represents a boundary edge. The purpose of the reference edge is to enable the sequential identification of the triangles which contain vertex v. For instance, a first such triangle is the triangle $tlefh(hrefv(v))$. Again, no separate array is needed, if the half-edges are numbered in such a fashion that the pairs of mirrored half-edges are in the same sequence as the vertices for which they are the reference edges.

Using the rule of thumb given in Section 5.2.1, the memory requirement for the edge-based data structure is estimated at

$$3 \times \#vertices + 2 \times \#half\text{-}edges + 3 \times \#triangles = 21 \times \#vertices\ .$$

The edge-based data structure requires nearly half again as much memory space as the triangle-based data structure.

5.2.3 Conditions for Search-Free Data Structures

At the beginning of Section 5.2, the need for convenient access to attributes and adjacency relationships was pointed out. In this section, the goal is to formalize those requirements by introducing the concept of a

"search-free"

data structure for 2D triangulations. The focus of this exposition will be on search-free data structures.

To start with, the notion of "convenient access" needs to be clarified. Here, it means that either an explicit listing – an array in the computer program – of the desired item is available, or the item can be retrieved by a accessing a suitable combination of such lists – in a sense, a concatenation of functions – or failing all of those provisions, a search of a number of items accessed by the previous two means. That number, however, should be predetermined, in other words, independent of the task at hand, and could therefore be hard-coded in a computer program. In the language used in formulating the two conditions below: the computational effort of accessing certain information should not depend on the parameters of a particular task. The first condition for a search-free data structure is to ensure that it provides for the assignment and ready retrieval of attributes.

> **Attribute Access Condition:** *A search-free TIN data structure should provide list structures for vertices, edges and triangles, respectively, so that the computational effort of retrieving an attribute of any vertex, edge or triangle should not depend on the parameters of a particular TIN application.*

Next, the retrieval of adjacency relationships needs to be defined. The second condition for a search-free data structure identifies the look-up tasks involved in establishing adjacencies.

> **Adjacency Access Condition:** *A search-free TIN data structure should provide list functions such that the following tasks require efforts that do not depend on the parameters of a particular TIN application. For each*
> 1. *vertex v , find a triangle trefv(v) of which v is a vertex;*
> 2. *vertex v , find a half-edge hrefv(v) which originates at vertex v ;*
> 3. *vertex v , find a half-edge htrmv(v) which terminates at vertex v ;*
> 4. *half-edge h , find its unique origin vertex vorgh(h) ;*
> 5. *half-edge h , find its unique destination vertex vdesh(h) ;*
> 6. *half-edge h , find its mirror hmirh(h) ;*

7. *half-edge h , find the unique triangle tlefh(h) to its left;*
8. *half-edge h , find its unique successor hsuch(h) in its triangle;*
9. *half-edge h , find its unique predecessor hpreh(h) in its triangle;*
10. *triangle t , find its three vertices vertit(i,t), i = 1, 2, 3, in positive order;*
11. *triangle t , find its three half-edges hedgit(i, t), i = 1, 2, 3, in successive order;*
12. *triangle t , find its three adjacent triangles tadjit(i, t), i = 1, 2, 3.*

The above listed tasks of adjacency determination are not independent. For instance, tasks 2 and 3 for vertex v can be accomplished by finding triangle $t = treft(v)$, and then examining the half-edges $h(i) = hedgit(t)$, $i = 1, 2, 3,$ as to whether $vorgh(h(i)) = v$ or $vdesh(h(t) = v$, respectively. Either of the first three look-ups will enable the star sweep task mentioned in the previous section.

Note that access condition 11 as well as either task 4 or 5 are also invoked. In fact, if those adjacencies are retrievable, then any of the first three conditions implies the remaining two. The procedures do require search, but that search is limited to three steps independently of the application at hand. In the same vein, if the mirror mapping of condition 6 is available, then conditions 4 and 5 as well as 8 and 9 are pairwise equivalent. Thus if either $vorgh(h)$ or $vdesh(h)$, can be retrieved, then condition 10 follows from condition 11.

It will now be shown that, in particular,

the triangle-based data structure can be interpreted as being search-free.

As a first step, attribute access is to be verified. Direct lists are provided for both triangles and vertices. Thus any desired attributes of those items will be directly accessible. Edges are not listed directly, but edge attributes can be attached to the triangle list, since every one of three half-edges belongs to a unique triangle. Thus the index pairs

$$(t\ ,\ 1)\ ,\ (t\ ,\ 2)\ ,\ (t\ ,\ 3)\ ,\quad t = 1,\ \ldots,\ \#triangles\ ,$$

are the half-edges with the respective origins $v_1(t)$, $v_2(t)$, $v_3(t)$ and corresponding destinations $v_2(t)$, $v_3(t)$, $v_1(t)$. They account for all the half-edges, as all triangles t are considered, and thus permit storing attributes in a corresponding 2-dimensional array. This establishes the attribute access condition.

The above convention immediately yields for the half-edge $(t\ ,\ i)$ the relationship

$$vorgh(t\ ,\ i) = v_i(t),\ \ i = 1, 2, 3\ ,$$

and the analogous expression for $vdesh(t\ ,\ i)$. This accounts for adjacency access conditions 4 and 5. Since the vertices $v_i(t)$ are listed in positive orientation,

$$tlefh(t\ ,\ i) = \ t\ ,\quad i = 1, 2, 3\ ,$$

accounting for condition 7. This then establishes conditions 8 and 9 as

$$hsuch(t\ ,\ 1) = (t\ ,\ 2)\ ,\ \ hsuch(t\ ,\ 2) = (t\ ,\ 3)\ ,\ \ hsuch(t\ ,\ 3) = (t\ ,\ 1)\ ,$$

and $hpreh(t\ ,\ i)$ is analogously defined. As to adjacency access condition 6, recall that the triangles in the triangle-based data structure have been arranged so that triangle $t_i(t)$ lies to the right of the half-edge $(t\ ,\ i)$. The mirror $hmirh(t\ ,\ i)$ to half-edge $(t\ ,\ i)$ can then be found by searching among the three half-edges of triangle $t_i(t)$ for the half-edge that originates at the destination of $(t\ ,\ i)$. Conditions 10, 11, and 12 are again trivially satisfied. As mentioned before, anyone of the first three adjacency access condition will imply the remaining two, provided several other conditions hold. Since all but the first three conditions have been verified, and since the function $trefv(v)$ is explicitly provided, the full complement of adjacency access conditions has been verified.

The edge-based data structure of Section 5.2.2 is also readily seen to be search-free. For instance, the retrieval of the vertices $v_1(t)\ ,\ v_2(t)\ ,\ v_3(t)$ of the triangle t as well of its adjacent triangles $t_1(t)\ ,\ t_2(t)\ ,\ t_3(t)$, based on its edges

$$h_1(t) = hfirt(t)\ ,\ \ h_2(t) = hsuch(h_1(t))\ \ ,\ \ h_3(t) = hsuch(h_2(t))\ ,$$

is demonstrated below:

$$v_1(t) = hdesh(h_1(t))\ \ ,\ \ v_2(t) = hdesh(h_2(t))\ ,\ \ \ v_3(t) = hdesh(h_3(t))\ ,$$
$$t_1(t) = tlefh(h_1(t))\ ,\ \ t_2(t) = tlefh(h_2(t))\ ,\ \ t_3(t) = tlefh(h_3(t))\ .$$

5.3 EXACT ARITHMETIC

The calculations based on the data structures described above need to maintain the combinatorial integrity of such data structures. Otherwise the NIST programs will crash. However, floating point arithmetic, with its built in rounding or truncating mechanisms, does not maintain combinatorial integrity. One of the problems is the inability to consistently recognize whether three points

$$p_i = (\ x_i\ ,\ y_i\ ),\ \ i = 1,\ 2,\ 3,$$

are in line. A criterion for this is that the

$$\text{determinant} \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix} \ = \ 0\ .$$

However, even if the points are in line, evaluation of this determinant with floating point arithmetic may result in a small non-zero value due to round-off. A tolerance $\delta > 0$, is frequently

specified below which absolute values are considered not to differ significantly from zero. If such a "zero tolerance" is chosen too small, it may not be realized that the three points are, in fact, in line. On the other hand, if the zero tolerance is chosen too large, say, in the course of applying the insertion method, inserting a data point close to an edge of the triangulation may result in a triangle of negative orientation due to round-off, as in the following example.

Consider four vertices of a current triangulation, indexed for convenience from 1 to 4,

$$p_1 = (0\ ,\ 1)\ ,\ \ p_2 = (2\ ,\ 1.000001)\ ,\ \ p_3 = (1\ ,\ 0),\ \ p_4 = (11\ ,\ 1.000007)\ .$$

Assume that the triangles

$$t_1 = (p_1\ ,\ p_2\ ,\ p_3)\ ,\quad t_2 = (p_2\ ,\ p_1\ ,\ p_4)$$

are part of the current triangulation. Obviously, they are adjacent to each other along the edge $(p_1\ ,\ p_2)$. Assume further that the point

$$p_0 = (1\ ,\ 1)$$

is to be inserted, and that a zero tolerance

$$\delta\ =\ 0.000002$$

has been specified. Note that the determinants which indicate the orientation of the triangles t_1 and t_2 are both significantly positive, as they should be:

$$\text{determinant} \begin{vmatrix} 0 & 1 & 2 \\ 1 & 0 & 1.000001 \\ 1 & 1 & 1 \end{vmatrix} \ = 2.000001 > \delta\ ,$$

$$\text{determinant} \begin{vmatrix} 0 & 2 & 11 \\ 1 & 1.000001 & 1.000007 \\ 1 & 1 & 1 \end{vmatrix} \ = \ 0.000003\ >\ \delta\ .$$

Observe then that the determinant

$$\text{determinant}\begin{vmatrix} 0 & 1 & 2 \\ 1 & 1 & 1.000001 \\ 1 & 1 & 1 \end{vmatrix} = 0.000001 < \delta \, .$$

is within zero tolerance, so that point p_0 is to be considered to lie on the edge $(p_1 \, , \, p_2)$, separating triangles t_1 and t_2. In this case, the first step of inserting p_0 creates (Section 3.1.1) four new triangles,

$$t_3 = (p_3 \, , p_0 \, , p_1) \, , \;\; t_4 = (p_3 \, , p_2 \, , p_0) \, , \;\; t_5 = (p_4 \, , p_0 \, , p_2) \, , \;\; t_6 = (p_4 \, , p_1 \, , p_0) \, ,$$

where the triangles t_3 and t_4 split the triangle t_1, the triangles t_5 and t_6, the triangle t_2. The triangle t_6, however, is now significantly negatively oriented:

$$\text{determinant}\begin{vmatrix} 11 & 1 & 2 \\ 1.000007 & 1 & 1.000001 \\ 1 & 1 & 1 \end{vmatrix} = -0.000003 < -\delta \, .$$

In order to avoid round-off and tolerance selection problems, NIST transforms point cloud data into integers by rounding to a fixed number of digits after the decimal point. The resulting coordinates are then multiplied by a common power of 10 in order to transform them into integers. Integer arithmetic is then used to process the determinants arising in the examination of the various criteria required by the respective algorithms. This is possible since the determinants in question can be evaluated without recourse to division.

In almost all practical applications, however, the word length of single precision integer arithmetic will be insufficient. Double precision floating point arithmetic applied to integers represented as floating point numbers will sometimes be applicable. NIST has chosen the route of implementing multi-precision integer arithmetic. The most promising way to use this high level tool is to employ it to resolve zero/non-zero questions only if results calculated in floating point arithmetic are close to zero.

5.4 "BOTTOM-UP" VERSUS "TOP-DOWN"

The issue of "top-down" versus "bottom-up" arises if a partial triangulation is desired. This issue is to be distinguished from the question of which triangulation algorithm is more efficient when computing the full Delaunay triangulation of a planar point set. An efficient algorithm is, for instance, the sweep line algorithm by [Fortune 1987]. This method starts with sorting by one planar variable – an $O(n \, log \, n)$ effort – and requires only linear, i.e., $O(n)$, effort thereafter. The insertion method stands to be less efficient, even when implemented without added adaptive selection procedures, because it requires locating each new insertion in a triangle of a current triangulation (Section 3.1.5). However, the authors do not know of a systematic computational

comparison between those methods. Moreover, much depends on the data, whose inherent patterns may favor one method over the other.

What matters, however, is the fact that methods which do not involve adaptive selection processes are more efficient provided the goal is to find a full triangulation. The top-down approach takes advantage of this edge in efficiency. It generates the full triangulation first and, should a partial triangulation be required, it reduces the triangulation by adaptively selecting vertices of the triangulation for deletion and subsequent restoration of the Delaunay triangulation. Frequently, the purpose of determining a partial triangulation is to reduce or "thin" the point cloud in order to reduce the computational effort of follow-on analyses. In this case, a commonly used criterion is to delete a vertex if the surface patch formed by the triangles adjacent to the vertex is close to planar [Kragovi and Tartalja 1996]. The rationale is that the deletion of such a vertex affects the surface in only a minor way. Most commercial implementations of TIN methods follow the top-down approach.

Nevertheless, if the partial triangulation represents only a minor portion of the full data set, say, 50 % or less, then the authors expect – without actual evidence – that the "bottom-up" adaptive approach is computationally comparable to an adaptive top-down approach, in particular, if the point cloud follows some sequential pattern (Section 3.1.5).

What about memory requirements? Suppose the top-down approach is based on the sweep line algorithm. The sorted file is used as input, with each point being processed as a vertex upon being read. Only the data structure which represents the triangulation per se needs to be stored. According to the estimate developed in Section 5.2.2 for the triangle-based data structure, the total memory requirement amounts to

$$15\,n, \quad n = \#\textit{data points}.$$

On the other hand, the bottom-up approach using the insertion method with point-to-triangle linking (Section 3.2.1) has an immediate memory requirement of $4n$, because it needs to store the entire point cloud with its three coordinates plus the array that links the data points. In addition, the data structures for the intermediate final triangulations have to be accommodated as the algorithms progresses, leading up to the final partial triangulation of m vertices. The total memory requirement of the bottom-up approach may thus be estimated at

$$4\,n + 15\,m, \quad n = \#\textit{data points}, \quad m = \#\textit{vertices}.$$

This suggests that a bottom-up approach requires less memory for partial triangulations of about 70 % or less of full triangulations. For partial triangulations of 25 %, the memory requirement is cut in half. The above analysis is, of course, based on assumptions that may not hold for individual implementations, but should nevertheless indicate a general trend.

There is demand for partial triangulations of low percentage. Meaningful data reduction would be for 50 % or less. Experimental assessment of volume determination to be reported in [Cheok and Witzgall 2004] found optimal accuracy to occur for partial triangulations at 30 %. It is not

known, whether this represents a general phenomenon; one would expect it to depend on the noise level of the data. Partial triangulations are also required for the elevation and triangulation adjustments described in Chapter 4. Here the data points that have not been selected as vertices guide the adjustments. It is recommended for those adjustments that there should be at least one of those non-vertex data points per triangle on average. As there are roughly twice as many triangles as vertices in a triangulation (Section 5.2.1), this amounts to $n - m = 2\ m$ or $3\ m = n$, where again n is the number of all data points with m the number of vertices in the partial triangulation. In other words, if adjustments are required, then partial triangulations of no more than 33 % of full triangulations are called for, and this represents an upper limit.

To sum up, there are strong indications that, for partial triangulations of 50 % or less, the bottom-up approach with adaptive selection processes is computationally less costly than the top-down approach, and there is demand for such partial triangulations. More importantly, the bottom-up approach holds the promise of more adaptive options since the full data set is always accessible to guide the triangulation process.

6. ALTERNATE TRIANGULATIONS

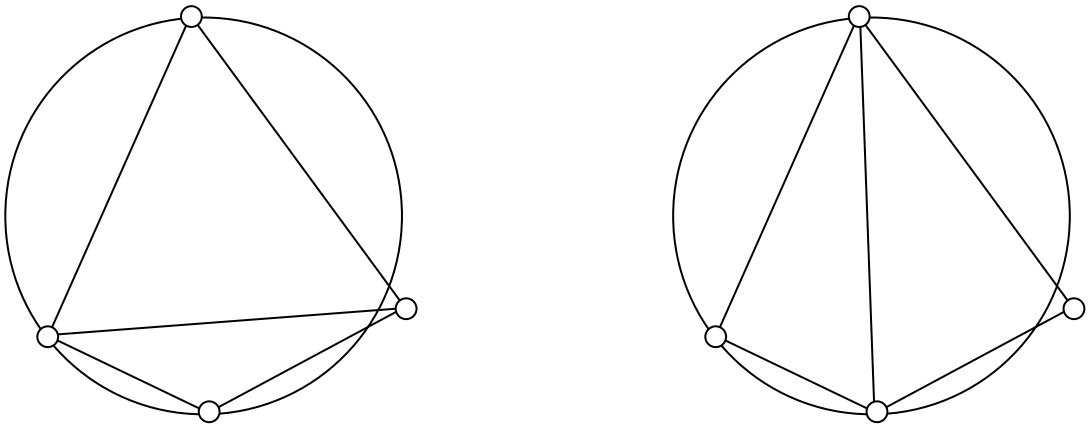
There are many different triangulation procedures in use, and not all of them are Delaunay based, For instance, the area covered by the triangulation may by consecutively expanded [Silva and Mitchell 1998]. In this report, only two alternate paradigms are discussed, the greedy triangulation and the triangulation of regular grids.

6.1 GREEDY TRIANGULATION

The concept of the

"greedy triangulation"

has played a role [e.g., Duppe and Gottschalk 1970, Lloyd 1977, Lingas 1986, Goldman, 1989] and is occasionally useful. It was inspired in part by the idea of an "optimal" triangulation which minimizes total edge length. In this vein, the pairs of points $p_i = (x_i , y_i)$, $p_j = (x_j , y_j)$ of a planar point set are sorted by increasing distance from each other. For a first edge, connect the points closest to each other by a straight line segment. Then connect the next closest pair, and continue in this fashion, but ignore those pairs of points whose connecting line segment would cross previously established edges. This process terminates when there are no pairs left which are eligible to be connected. At this point, the convex hull of the planar point set has been partitioned into the triangles of the greedy triangulation. The greedy triangulation is similar and, in some cases, identical to the Delaunay triangulation. A simple example of points for which the two triangulations differ is given in Fig. 6.1.



Shortest DistanceTriangulation Delaunay Triangulation

Figure 6.1. Shortest distance and Delaunay triangulations may differ.

Not counting the considerable effort involved in avoiding crossovers, the triangulation procedure based on sorting edges by length is of complexity $O(n^2 \log n)$, where n denotes the number of points to be triangulated. Efforts have been made to make greedy triangulation more efficient [Levcopoulos and Lingas 1992, Dickerson et al. 1997]. It may also be more efficient to proceed along the lines of the insertion method (see Chapter 3) using, instead of the empty circle criterion, the following

> ***Shorter Diagonal Criterion:*** *For any two adjacent triangles that form a convex quadrangle, a diagonal of equal or shorter length than the other diagonal of this quadrangle is an edge in the triangulation.*

Again, the efficiency of this method depends on the efficiency with which a data point can be located in a triangle of the intermediate triangulation. An advantage of the insertion approach to constructing the shortest distance triangulation is that it may be terminated with a partial triangulation, and that it is therefore amenable to adaptive techniques. The shorter diagonal criterion has been used, for instance, as part of a triangulation method based on sweeping the plane in a radial fashion [Mirante and Weingarten 1982].

While the greedy triangulation must satisfy the shorter diagonal criterion, it is not necessarily determined by it. Figure 6.2 provides an example of two different triangulations of the same point set, both of which satisfy the shorter diagonal criterion, but only the triangulation on the right is greedy. Indeed, the diagonal (p_1, p_4) is shorter than the diagonal (p_3, p_5) , so the former will be selected before the latter in the greedy process.

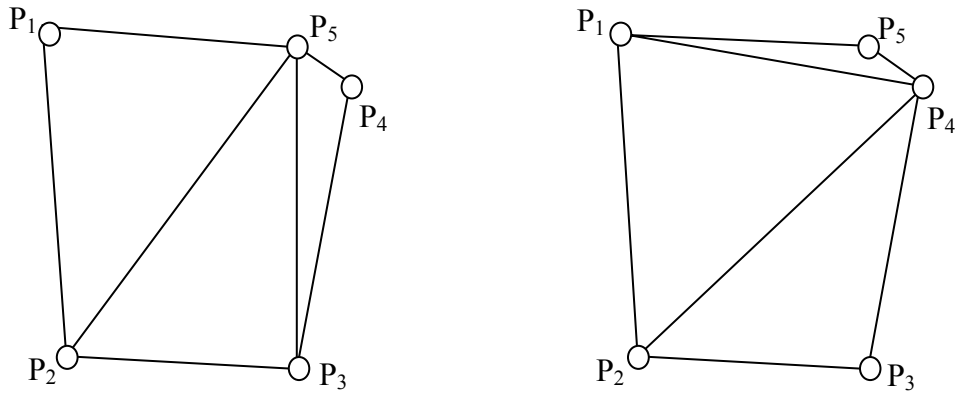


Figure 6.2. The triangulation on the right is greedy, the one on the left is not.

6.2 GRIDDED DATA SETS

In many applications, point data are provided as elevations over a regular rectangular grid. That is, the footprints (x_i , y_j) of a point cloud are grid points, regularly spaced in x and y, respectively, with a single elevation z_{ij} assigned to each grid point. Numerous methods are available to approximate irregular point clouds by such gridded data sets. They will not be discussed here. It should be pointed out, however, that meshing an irregular data set, and then sampling the surface elevations at the grid coordinates, is a fast and oscillation-free procedure for extracting a gridded data set from an irregular point cloud. Many methods, also, are in use for constructing a 2.5D surface from such a gridded data set: bilinear and bicubic interpolation in each grid cell are popular options, as are various splining procedures. All these procedures may be based on an extremely simple data structure. Indeed, the number of rows and columns, the two cell sizes in x and y, and the "Southwest" corner of the full grid rectangle are sufficient to retrieve the value of the coordinates x_i , y_j from their respective row and column numbers. These coordinates need, therefore, not be stored. This leaves the elevations z_{ij}, which may be listed in conventional matrix formats.

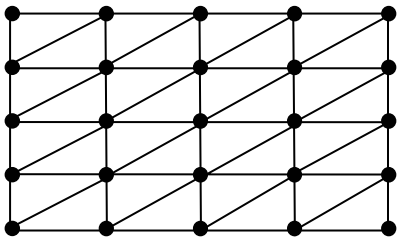


Figure 6.3. Triangulation of a gridded data set.

The triangulation shown in Fig. 6.3 suggests itself and is widely used. The compactness of the data structure that suffices to encode a gridded data set and its regular triangulation is the key advantage of a gridded data set. General display packages may require representation in a more complex triangulated data structure. The computational effort of restructuring the information in terms of, say, the data structures discussed in Section 5.2.2 is minimal because it can be based on straightforward index manipulations. However, the decisive advantage of the compact data structure is lost.

Gridded data sets are often incomplete, having so-called

"no-data values"

attached to grid locations for which no data are available. If they occur in small isolated clusters, then it is recommended to use local interpolation schemes to assign values to grid locations, thereby completing the grid. In this fashion the regularity of the data structure is preserved. Four-way interpolation, illustrated in Fig. 6.4, is typical. A more expensive approach is to treat

the data as an irregular data set and create a meshed surface, which then defines elevations at the missing grid points. Greedy triangulation has a role there.

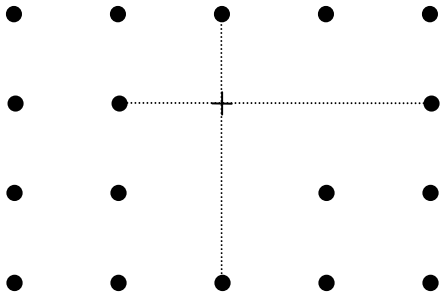


Figure 6.4. Four-way interpolation to no-data values for a gridded data set.

7. MESHING OVER SPHERES

Some LADARs produce a panoramic data set, that is, their point clouds are arranged by polar coordinates, and are thus more naturally associated with locations on a sphere than with locations on a footprint plane. In particular, the transformation to x, y, z-coordinates with the usual interpretation of z representing actual vertical elevation and of x and y varying over a horizontal footprint plane, often leads to unrealistic neighbor specifications. This problem is in part remedied by stipulating a single "scan direction" with a footprint plane perpendicular to it, so that "elevation" is defined in that scan direction. This device, however, limits viewing angles and is certainly not feasible for panoramic views. In this chapter, methods are examined that project footprints onto spheres.

7.1 THE ROLE OF CONVEX HULLS

Of particular interest in this section are finite sets of 3D points located on a sphere,

$$\{P_1 , P_2 , \dots , P_n\} .$$

The convex hull of such a spherical point set is a convex polytope with the above points P_i as its vertices and with facets that are generally triangles. For a facet not to be a triangle as, for instance, the facets of a cube, then that facet may be partitioned – in several ways – into triangles by drawing diagonals. A convex hull of points that lie on a sphere thus gives rise to a

"convex triangulation"

of the sphere. This triangulation is unique if no two edge-adjacent triangles are coplanar.

7.1.1 Connection to Delaunay Triangulation

It is well known that Delaunay triangulations in a plane correspond to convex triangulations on a sphere. To establish this correspondence, the sphere is placed on a horizontal plane. Let P_0 denote the top point or "North Pole" of the sphere. A projection of the sphere – leaving out P_0 – onto the plane may be defined as follows. If $P \neq P_0$ is a point on the sphere, then the line through P_0 and P intersects the plane at the point p, which is the projection of the point P on the sphere. Note that the projection can be reversed: if p is a point in the plane, then the line through p and P_0 intersects the sphere in the previous point P (Fig. 7.1).

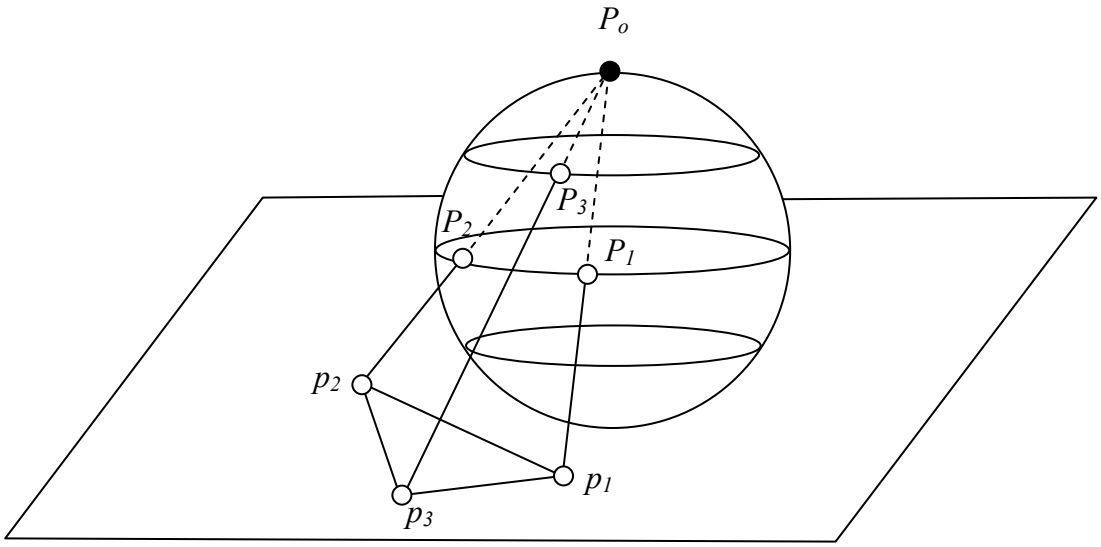


Figure 7.1. Projecting a triangle on a sphere into a plane.

The connection with the Delaunay triangulation is as follows. Given a set of n points $P_i \neq P_0$, on the sphere, consider the triangulation of the sphere determined by the convex hull of the North Pole P_0 together with the points P_i. This convex hull defines a triangulation of the sphere. It can be shown that the projection defined above carries that triangulation of the sphere into a triangulation of the projections p_i in the plane. This planar triangulation is Delaunay, because the projection carries circles on the sphere onto circles in the plane. Conversely, the triangles of the Delaunay triangulation are, in reverse, projected onto triangles of a convex hull of points on the sphere.

7.1.2 Center Projections – Footprints on a Sphere

The use of *x, y, z* – coordinates with *x* and *y* the coordinates of a horizontal plane, and z indicating vertical elevation, as described in the main body of the report, is frequently not advantageous. The direction from the instrument to the object may be a better choice as a "vertical" direction since the range is measured in this direction, and deviations in this direction from the object surface due to noise are a more natural error measure.

However, the instrument direction varies as each range measurement has its own bearing. As mentioned at the beginning of the chapter, the assumption of a fixed instrument direction may be an acceptable approximation in some applications, but it does not work for the panoramic scans, say, by a rotational LADAR. For those data, the concept of a footprint plane needs to be

abandoned in favor of a "footprint sphere" centered at the instrument, if the advantages of meshing in the instrument direction are to be maintained.

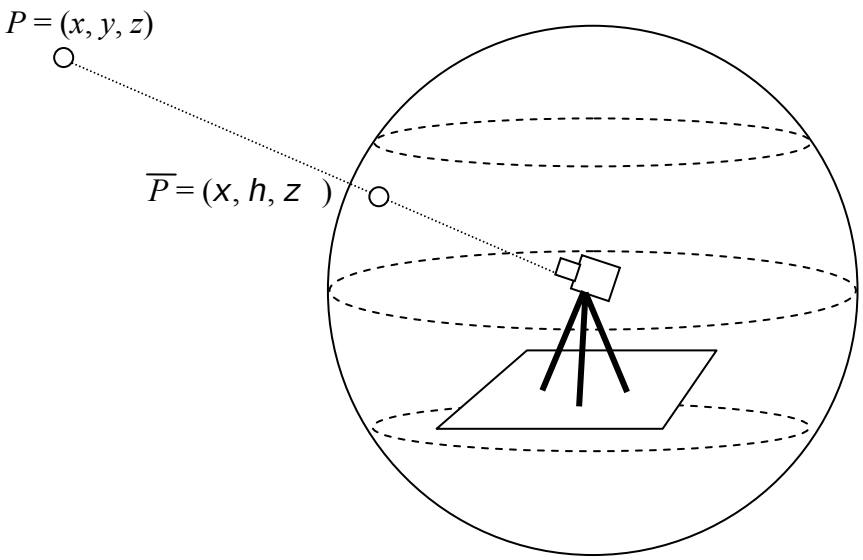


Figure 7.2. Center projection of data points onto a sphere centered at the instrument.

In this and the subsequent section, the term "sphere" will always refer to a sphere of radius 1 centered at the instrument, which is also the origin of any frame in which the original data points are collected (Fig. 7.2). These points P_i are expressed either in a Cartesian *x, y, z*-coordinate system or in a "polar" *r, φ, θ*-coordinate system:

$$P_i = (x_i , y_i , z_i) = (r_i , \varphi_i , \theta_i).$$

The footprint p_i of that data point P_i with respect to the sphere is the intersection of the sphere and the ray that starts at the origin and passes through the point P_i :

$$p_i = (\varphi_i , \theta_i) = (\xi_i , \eta_i , \zeta_i),$$

Where,

$$\xi_i = \cos \varphi_i \cos \theta_i = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}},$$

$$\eta_i = \sin \varphi_i \cos \theta_i = \frac{y_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}},$$

$$\zeta_i = sin\ \theta_i = \frac{z_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}},$$

so that

$$\xi_i^2 + \eta_i^2 + \zeta_i^2 = 1 .$$

Instead of meshing by triangulating footprints of data points in an *x,y*-plane, footprints of data points P_i are triangulated on the surface of the sphere. Such an

"origin-centered triangulation"

of a given set of spherical footprint points p_i consists of the flat triangles whose vertices are spherical footprint points and which are facets – or which partition polygonal facets of more than three vertices – of the convex hull of the footprints and the origin 0. More precisely, there are two cases to consider, depending on whether the origin lies in the interior – the fully panoramic case – or on the boundary of the convex hull. In the former case, all triangles which partition the convex hull constitute a triangulation. In the latter case, the origin may be a vertex, lie on an edge, or in the interior of a facet. Then triangles which cover facets containing the origin are excluded from the triangulation.

Just as in the case of a footprint plane, the triangles of the triangulation have to be positively oriented. Three points $p_1 = (\xi_1, \eta_1, \zeta_1)$, $p_2 = (\xi_2, \eta_2, \zeta_1)$, $p_3 = (\xi_3, \eta_3, \zeta_3)$ on the sphere are

"positively oriented"

if the tetrahedron $(0, p_1, p_2, p_3)$ is positively oriented, that is, it has a positive volume. In terms of determinants one then has

$$\text{determinant}\begin{vmatrix} 0 & \xi_1 & \xi_2 & \xi_3 \\ 0 & \eta_1 & \eta_2 & \eta_3 \\ 0 & \zeta_1 & \zeta_2 & \zeta_3 \\ 1 & 1 & 1 & 1 \end{vmatrix} > 0 \qquad \text{or, equivalently,} \qquad \text{determinant}\begin{vmatrix} \xi_1 & \xi_2 & \xi_3 \\ \eta_1 & \eta_2 & \eta_3 \\ \zeta_1 & \zeta_2 & \zeta_3 \end{vmatrix} < 0 .$$

Consider the "spherical quadrangle" arising from two adjacent positively-oriented spherical triangles

$$(p_1, p_2, p_3),\ (p_2, p_1, p_4) .$$

Then the "complementary" triangles (Fig. 7.3)

$$(p_1, p_4, p_3),\ (p_2, p_3, p_4)$$

are again positively oriented.

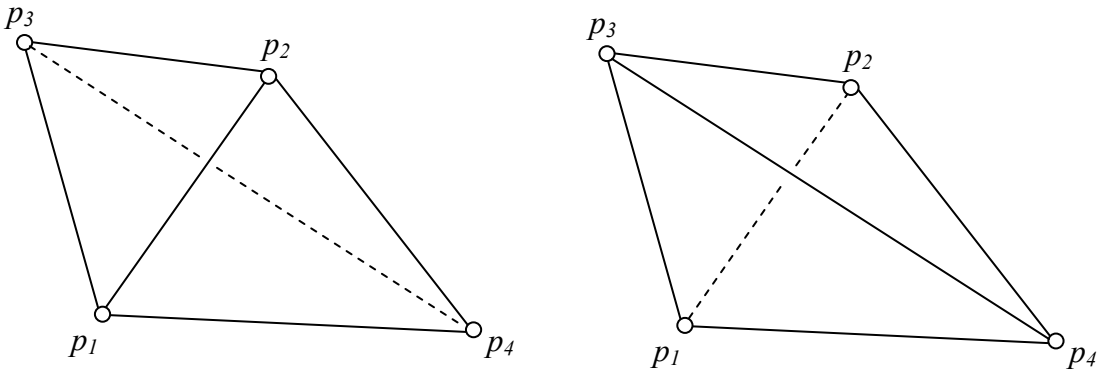


Figure 7.3. Four spherical points may be represented in two ways as adjacent triangles.

The two adjacent, positively-oriented, spherical triangles at left in Fig. 7.3,

$$(p_1, p_2, p_3), \ (p_2, p_1, p_4).$$

are part of the boundary of a convex hull if and only if

$$\text{determinant} \begin{vmatrix} \xi_1 & \xi_2 & \xi_3 & \xi_4 \\ \eta_1 & \eta_2 & \eta_3 & \eta_4 \\ \zeta_1 & \zeta_2 & \zeta_3 & \zeta_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} \leq 0 .$$

It can be shown that if the above determinant is negative, then the complementary configuration, shown at left in Fig. 7.3, gives rise to a positive determinant. This suggests the kind of "diagonal flipping" analogous to the insertion method for Delaunay triangulations for constructing origin-centered spherical triangulations.

An analog to data sets defined on a planar grid exists on the sphere with rows of points of equal latitude in equiangular position, and with columns of points of equal longitude, stepping with equal latitudinal increments. The corresponding spherical triangulation depicted in Fig. 7.4 describes a convex polytope and therefore a center-oriented triangulation. Such a triangulation can again be presented by an extremely compact data structure.

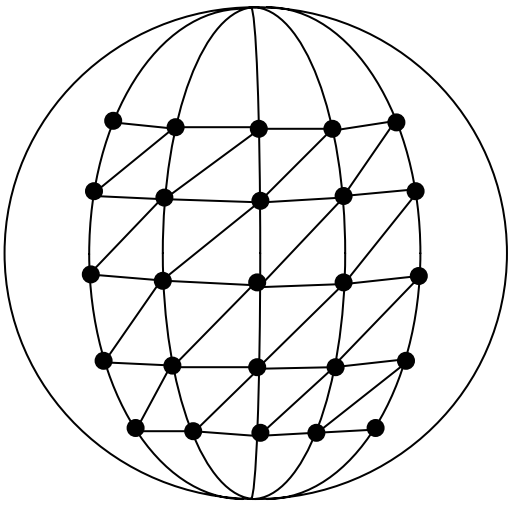


Figure 7.4. A regular center-oriented triangulation of the sphere.

8. REFERENCES

Amenta, N., and M. Bern [1999], "Surface Reconstruction by Voronoi Filtering," *Discr. Comput. Geom*., 22, pp. 481-504.

Amenta, N., S. Choi, T. K. Dey, and N. Leekha [2000], "A Simple Algorithm for Homeomorphic Surface Reconstruction," *Proc. 16th ACM Sympos. Comput. Geom. (2000)*, pp. 213-222.

Aurenhammer, F. [1991], "Voronoi Diagrams – a Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys 23*, pp 345-405.

Beichl, I., J. Bernal, F. Sullivan, and C. Witzgall [1996], "Voronoi Constructs," *Encyclopedia of Operations Research and Management Science*, Kluwer Academic Publishers, pp.726-729.

Bernal, J. [1988], "On Constructing Delaunay Triangulations for Sets Constrained by Line Segments," *NIST Technical Note 1252*, National Institute of Standards and Technology, Gaithersburg, MD, September.

Bernal, J. [1993], "Bibliographic Notes on Voronoi Diagrams," *NISTIR 5164*, National Institute of Standards and Technology, Gaithersburg, MD, April.

Bernal, J. [1995], "Inserting Line Segments into Triangulations and Tetrahedralizations," *NISTIR 5596*, National Institute of Standards and Technology, Gaithersburg, MD, March.

Bernal, J. [2001], "REGTET: A Program for Computing Regular Tetrahedralizations," *NISTIR 6786,* National Institute of Standards and Technology, Gaithersburg, MD, September.

Bernal, J., and C. Witzgall [1999], "Triangulation-based L_1-fitting of Terrain Surfaces," *NISTIR 6346*, National Institute of Standards and Technology, Gaithersburg, MD, June.

Cheok, G. S., Lipman, R. R., Witzgall, C., Bernal, J., and Stone, W. C. [2000], "NIST Construction Automation Program Report No. 4: Non-Intrusive Scanning Technology for Construction Status Determination," *NISTIR 6457*, National Institute of Standards and Technology, Gaithersburg, MD, January.

Cheok, G. S., S. Leigh, and A. Ruhkin [2002], "Calibration Experiments of a Laser Scanner," NISTIR 6922, National Institute of Standards and Technology, Gaithersburg, MD, September.

Cheok, G. S., and C. Witzgall [2004], "Triangular Meshing and Volume Determination," *NISTIR* in preparation, National Institute of Standards and Technology, Gaithersburg, MD.

Delaunay, C. [1934], "Sur la sphere vide," *Bull. Acad. Sci. USSR* (VII), Classe Sci. Mat., pp. 793-800

Dey, T. K., and J. Giesen [2001], "Detecting Undersampling in Surface Reconstruction," *Proc. 17th ACM Symp. Comput. Geom. (2001)*, pp. 257-263.

Dickerson, M., R. L. S. Drysdale, S. A. McElfresh, E. Welzl [1997], "Fast Greedy Triangulation Algorithms," *Computational Geometry: Theory and Applications*, 8, pp. 67-86.

Duppe, R. D., and H. J. Gottschalk [1970], "Automatische Interpolation von Isolinien bei willkürlichen Stützpunkten," *Allgemeine Vermessungsnachrichten*, 77, pp. 423-426.

Edelsbrunner, H. [1987], "*Algorithms in Combinatorial Geometry*," Springer Verlag, New York, NY.

Edelsbrunner, H., D. G. Kirkpatrick, and R. Seidel [1983], "On the Shape of a Set of Points in the Plane," *IEEE Trans. Inform. Theory,* 29, pp. 551-559.

Edelsbrunner, H., and E. P. Mucke [1994], "Three-dimensional Alpha Shapes," *ACM Trans. Graphics,* 13, pp. 43-72.

Edelsbrunner, H., and N. R. Shah [1996], "Incremental Topological Flipping Works for Regular Triangulations," *Algorithmica* 15 (3), pp. 223-241.

Fortune, S. [1987], "Sweep Line Algorithm for Voronoi Diagrams," *Algorithmica*, 2, pp. 153-174.

Gass, S. I. [1985], "*Linear Programming*," 5th ed., McGraw-Hill, New York, NY.

Goldman, S. A. [1989], "Space Efficient Algorithm for the Greedy Triangulation", Information Processing Letters, Vol. 31, No. 4, pp. 191-196.

Guibas, L. J., D. E. Knuth, and M. Sharir [1990], "Randomized Incremental Construction of Delaunay and Voronoi Diagrams," Springer Verlag, *Lecture Notes in Computer Science*, Vol. 443, pp. 414-431.

Heckbert, P., and M. Garland [1995], "A Survey of Terrain Triangulation Algorithms," Technical Report, Carnegie Mellon University.

Kirkpatrick, D. G. [1983], "Optimal Search in Planar Subdivisions", *SIAM J. Comput.,* 12, pp. 28-35.

Kragovi, M., and I. Tartalja [1998], "An Approach to Surface Data Set Reduction," manuscript, PTT College and School of Electrical Engineering, University of Belgrade.

Lawson, C. L. [1977], "Software for C^1 surface interpolation," *Mathematical Software III*, J.R. Rice (Ed.), Academic Press, New York, pp. 161-194.

Lawson, C.L. [1986], "Properties of *n*-dimensional Triangulations," *Computer Aided Geometric Design* 3, pp. 231-246.

Lee, D.T., and F.P. Preparata [1977], "Location of a Point in a Planar Subdivision and its Applications," *SIAM J. Comput.,* 6, pp. 594-606.

Levcopoulos, C., and A. Lingas [1992], "Fast Algorithms for Greedy Triangulations," BIT 32, pp. 280-296.

Lingas, A. [1986], "The Greedy and Delauney Triangulations are not bad in the Average Case," *Inform. Proc. Letters* 22, pp. 25-31.

Lloyd, E. L. [1977], "On Triangulation of a Set of Points in the Plane," *Proc. 18th Annual IEEE Conf. on the foundation of Computer Science*, Providence, R.I., pp. 228-240.

Mandel, B., J. Bernal and C. Witzgall [1987], "Contour-to-Grid Interpolation with Nonlinear Finite Elements: A Feasibility Study," *ETL-0472,* U.S. Army Corps of Engineers, Engineer Topographic Laboratories, Fort Belvoir, VA 22060-5546.

Mirante, A., and N. Weingarten [1982], "The Radial Sweep Algorithm for Constructing Triangulated Irregular Networks", *IEEE Computer Graphics and Applications*, pp. 1-21.

Preparata, F.P. [1981], "A New Approach to Planar Point Location," *SIAM J. Comput.,* 10, pp. 473-482.

Preparata, F. P., and M. I. Shamos [1985], "*Computational Geometry*," Springer Verlag, New York, NY.

Peucker, T. K., R. J. Fowler, J. J. Little, and D. M. Mark [1976], "Digital Representation of Three Dimensional Surfaces by Triangulated Irregular Networks (TIN)" *Technical Report* 10, Office of Naval Research, Geography Programs

Peucker, T. K., R. J. Fowler, J. J. Little, and D. M. Mark [1978], "The Triangulated Irregular Network." *Proceedings of the Digital Terrain Model Symposium*, St. Louis, MO, pp. 516-532.

Polis, M. F., S. J. Gifford, D. M. McKeown Jr. [1995], "Automating the Construction of Large Scale Virtual Worlds" *IEEE Computer* 28 (7), pp. 57-65.

Samet, H. [1990], "*The Design and Analysis of Spatial Data Structures*", Addison Wesley, Reading, PA.

Silva, C. T., and J. S. B. Mitchell [1998], "Greedy Cuts: An Advancing Front Terrain Triangulation Algorithm," Proc. of the 6th ACM International Symposium on Advances in Geographic Systems, Washington D. C., pp. 137-144.

Stoyan, D., W. S. Kendall, and J. Mecke [1987], "*Stochasic Geometry and its Applications*", John Wiley , New York, NY.

Witzgall, C., and R. S. Karalus [1991], "Terrain Modelling: Shortest Path, Drain Patterns, and Interspersed contours," *TEC-003,* US Army Corps of Engineers, Topographic Engineering Center, Fort Belvoir, VA, December

Witzgall, C., and G. S. Cheok [2001], "Registering 3D Point Clouds: An Experimental Evaluation," *NISTIR 6743*, National Institute of Standards and Technology, Gaithersburg, MD, May.