BETTER SURFACE REPRESENTATIONS BY DELAUNAY TETRAHEDRONIZED IRREGULAR NETWORKS

Edward Verbree and Peter van Oosterom
E.Verbree@CiTG.TUDelft.NL and P.J.M.vanOosterom@CiTG.TUDelft.NL
Department of Geodesy
Delft University of Technology
Thijsseweg 11
2629 JA Delft
The Netherlands

KEY WORDS:

Conforming Delaunay Tetrahedronized Networks, Surface Reconstruction, Data Dependent Triangulations

ABSTRACT:

The Triangulated Irregular Network (TIN) created by the 3D-Analyst extensions of ArcGIS 8.x and ArcView 3.x are based on the Delaunay empty circum circle criterion, which is believed to optimize how faces model a surface. This is not the case as this algorithm ignores the Z-value of the surface points. To get a better representation of the surface this papers describes a method based on the Delaunay Tetrahedronized Irregular Network (TEN). This TEN stores the measurements (lines-off-sight to the surface-points). An ArcView algorithm is applied on this TEN to retrieve the TIN that represents the surface as good as possible.

1. INTRODUCTION

1.1 Background and motivation

A proper representation of the surface of the earth and what is build upon it is needed as a data source for environmental modelling and planning. Especially Virtual and Augmented Reality applications require an appropriate representation of the actual terrain and man-made objects. One way to represent the terrain given by a set of surface points is to construct a Delaunay Triangular Irregular Network (DTIN). This DTIN is believed to give the 'best' triangular tessellation as the Delaunay empty circle criterion opts for well-formed 'fat' triangles and the resulting triangulation maximizes the smallest angle. This idea is true for many applications, but it is not valid for visual and analytical queries dependent on the height of the surface. This limitation is given by the fact that the distribution of the triangular mesh is defined in the two-dimensional XY-plane and the Z-value of the surface points is not taken into account by the Delaunay empty circle criterion at all. Alternatively, Data Dependent Triangulations (DDTINs) aim to identify which triangulation over a given set of points will optimize some quality, i.e. the minimal spatial area of the surface or the volume below the resulting surface. The Z-value of the surface points is now taken into account, but still no certainty can be given that the derived TIN represents the actual surface.

Hence, the reconstruction of the surface given by only the set of surface points is not unambiguous.
This paper describes a surface reconstruction method based on the Delaunay Tetrahedronised Irregular Network (DTEN), which tessellates the 3D-space with non-overlapping, adjacent, tetrahedrons. The DTEN is constructed by the Delaunay criterion, resulting in a tessellation where the circumscribing sphere of each tetrahedron is empty. The approach presented in this paper is new in that not only the surface points are included into the DTEN, but also the observation lines, i.e. the lines-of-sight between the observer (the measurement platform) and the targets (the measured points). These observation lines add the information needed to extract the Surface TIN (STIN) from this DTEN. Afterwards the observation lines are discarded and they act for that purpose as a catalyst. Therefore the observation lines can also be artificial or simulated for this purpose. The STIN approach presented in this paper is a full 3D-implementation and refinement of the research presented in (Verbree, 2001).

1.2 Limitations on Delaunay Triangulations

TINs are commonly used for surface representation. Given target points on the surface an Irregular Network of Triangles is created. The Z-value of these features is stored as the Z-value of the nodes of the computed TIN. A Delaunay TIN fulfils the 'empty circle criterion'. This criterion opts for the triangulation with 'fat' triangles, such that the triangulation maximizes the smallest angle (Figure 1.1 and 1.2).

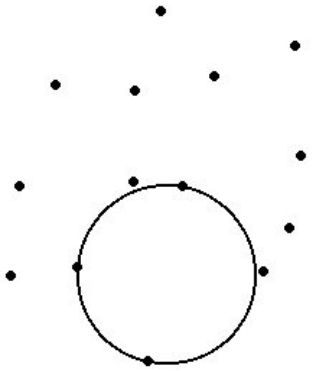

Figure 1.1
Empty circle criterion

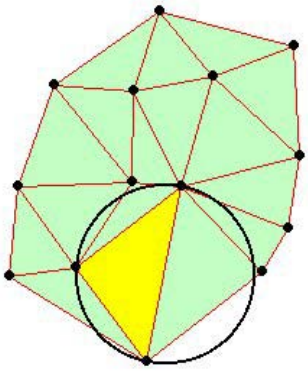

Figure 1.2
Resulting Delaunay TIN

This kind of TIN construction by the Delaunay criterion is used within the 3D Analyst extensions of both ArcView GIS and ArcGIS. We have to realise however that the 'empty circum circle criterion' does not take the Z-value of the features into account at all. This is clearly seen if the point distribution is square, as in the following example (Figure 1.3 and 1.4). In these figures 25 target points are given, with an alternating Z-value of 1 or 2.

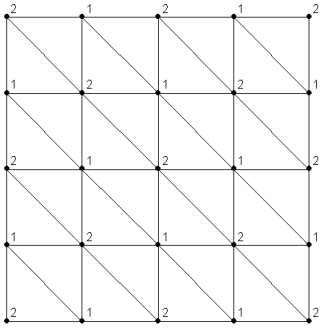

Figure 1.3
One possible Delaunay TIN

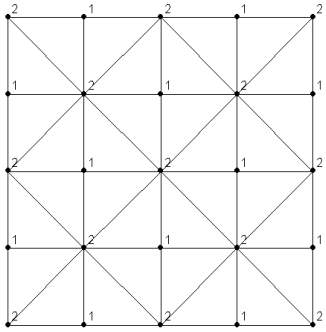

Figure 1.4
Another Delaunay TIN

In Figure 1.3 the diagonal of all triangles is directed northwest to southeast. As four points on a square are also on the same circle, it is with the Delaunay criterion in mind, free to choose a direction for the diagonals. This could be northeast to southwest for all diagonals or randomly distributed as in Figure 1.4. Which one to choose?

1.3 Limitation on Data Dependent Triangulations

The height values of the target points (or the Z-values of the nodes in the DTIN) do have consequences for derivatives like slope and aspect, visualization (hill-shading) and volume statistics (view sheds, and cut and fill calculations). One can argue that the 2D-Delaunay TIN (the triangulation of a 'flat' surface) is just one of the possibilities to triangulate a set of points and lines. In fact, any triangulation can be a candidate for a 2.5D terrain surface representation.

A better approach is to take the Z-value of the target points into account in the triangulation process. Extensive research on Data Dependent Triangulations (DDTINs) proves this observation. The idea behind this concept is to maximize or to minimize some cost functions that express certain local, regional or global properties of the resulting surface (Dyn, 1990; Lenk, 2000). Possible options for this cost functions are: minimize the surface area, minimize the volume, minimize the maximum angle of the surface triangles, etc. But it has to be stated that pure DDTINs do not perform well on terrain surface data as they tend for large amounts of slivers and steep triangles. Furthermore the local and global criteria could disregard certain phenomena, like ridges and faults.

1.4 The STIN-method; in search for improvement

Both Delaunay and Data Dependent Triangulations are 2.5D surface reconstruction techniques given a discrete 2.5D data set. This limitation is suitable for most terrain applications, but no overhanging cliffs or other disturbances are possible. Reconstructing the surface of caves, buildings or other full 3D-phenomena are only possible for parts of the data set, which first has to be projected to a suitable XY-plane. Therefore a full 3D representation, like the Delaunay Tetrahedronized Irregular Network (DTEN), could be considered. Within this DTEN many, many surfaces through the data points are embedded. A little trick is needed to select the 'best' surface.

One way to retrieve this surface is examining the data acquisition process. The position of a surface point is determined on the position of the observer and the direction and distance of the measurement or observation. So, for each data site (or target point) the position of the observer (observation point) is known. And of course the line of sight (observation line) between target and observer should be free of obstacles, otherwise no measurement can be made. The STIN method takes these observation lines into account in the surface reconstructing process. The observation lines are split with Steiner points until each part of the observation line recurs as an edge in the DTEN. This so-called conforming DTEN (Shewshuk) gives in conjunction with the Steiner points enough information to reconstruct the surface.

1.5 Overview of paper

In exploring the problem to retrieve a 2.5D surface within a Tetrahedronized Irregular Network within 2.5D and within 3D the procedure is given for the 1.5D situation. Here we want to retrieve a 1.5D surface within a Triangulated Irregular Network in 2D. The approach is given in chapter 2 and illustrated by some Avenue code. In the next chapter the 2.5D surface reconstructing process based on these ideas are described by many figures and examples. Chapter 4 gives applications for the STIN algorithm for 3D surface reconstructions. Chapter 5 gives some explanations and consideration in using ArcView and Avenue scripting while prototyping the STIN method. Chapter 6 will end up with conclusions and recommendations for further research.

2. SURFACE REPRESENTATION: THE 1.5D CASE

In exploring the problem to retrieve a 2.5D surface within a Tetrahedronised Irregular Network in 3D we explain this procedure first for the 1.5D situation. Here we want to retrieve a 1.5D surface within a Triangulated Irregular Network in 2D. This seems to be quite trivial to do, because we can order the target points on X-value. But as, for our goal, this is not straightforward in two dimensions, we have to use an algorithm, which will not take this ordering as a precondition.

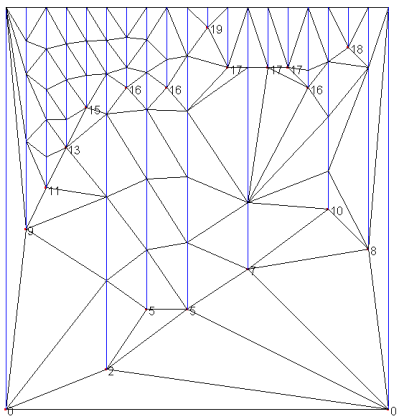

Figure 2a:
Delaunay TIN constrained by observation lines

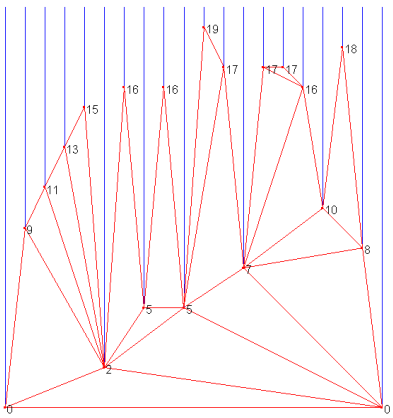

Figure 2b:
Derived '2D-Volume' and '1.5-Surface'

The aim is to find the '2D-volume' and the '1.5D-surface' defined by a set of target points and observation lines. The observation lines are shortened to a given value above the most extreme height value, and in this case dropped as perpendiculars.

We will give the algorithm by some abstracts of used Avenue scripts

Step 1: Initialisation

```
` Read target points
` Define observation lines
```

First a DTIN is created by a set of target points and observer-points. The ArcView 3D Analyst method of constructing a conforming DTIN forces each observation-line not being a TIN_Edge to subdivide into parts at a Steiner point. These Steiner points are included into the DTIN. This step finishes when all observation lines are represented by TIN_Edges in the Delaunay TIN. The addition of Steiner points is a powerful concept and is used in this approach as mean to find the Surface TIN.

Step 2: Construct conforming DTIN - see figure 2a

```
' Add ScanPoints
Tin_Data.AddShapesFromFTab (FTab_ScanPoints,Field_ScanPoints_Id,
  #SURFACEFEATURE_TYPE_MASS,Field_ScanPoints_Id,Prj.MakeNull)
' Add ScanLines
Tin_Data.AddShapesFromFTab (FTab_ScanLines,Field_ScanLines_Id,
  #SURFACEFEATURE_TYPE_HARDLINE,Field_ScanLines_Id,Prj.MakeNull)
```

The '2D Volume' and the '1.5D Surface' are found by the procedure, given by the code in step 3. All TIN_Edges are examined. If within a TIN_Edge a Steiner point is present the Target point at the end of the observation-line replaces this one. These newly constructed TIN_Edges are not Delaunay anymore, but are needed to obtain the 1.5D Surface and 2D Volume. All TIN_Edges (partly Delaunay, partly not) with two target points are maintained, the other TIN_Edges are discarded. The remaining TIN_Edges compose a complete and valid TIN.

Step 3: Transform TIN_Edges to Volume_Edges – see figure 2b

```
' Populate Surface Theme
FTab_SurfaceLines.SetEditable(True)
for each Rec_the in FTab_ScanPoints
  Point_ScanPoint = FTab_ScanPoints.ReturnValue(Field_ScanPoints_Shape,Rec_the)
  Number_ScanPoint_Id = FTab_ScanPoints.ReturnValue(Field_ScanPoints_Id,Rec_the)
  List_NaturalNeighbors_Indeces = TIN_Data.GetNaturalNeighbors(Point_ScanPoint)
  List_ScanLines_Ids = List.Make
  for each Number_NaturalNeighborIndex in List_NaturalNeighbors_Indeces
    if (Number_NaturalNeighborIndex <> 0) then
      PointZ_The = TIN_Data.ReturnNode(Number_NaturalNeighborIndex)
      Number_Triangle = TIN_data.Triangle(PointZ_The)
      If (TIN_Data.IsTriangleMasked(Number_Triangle).Not) then
        ' Find Scanline
        ' Trick: Z_coordinate is used as id!
        Number_ScanLines_Id = PointZ_The.GetZ
```

```
      If (Number_ScanLines_Id <> Number_ScanPoint_Id) then
        List_ScanLines_Ids.Add(Number_ScanLines_Id)
      end
    end
  end
  List_ScanLines_Ids.RemoveDuplicates
end

for each Number_ScanLines_Id in List_ScanLines_Ids
  Number_Record_Id = Number_ScanLines_Id
  Point_ScanLine = Tab_ScanPoints.ReturnValue(Field_ScanPoints_Shape,Number_Record_Id)
  PolyLine_SurfaceLine = PolyLine.Make({{Point_ScanPoint,Point_ScanLine}})
  Number_NewRecord = FTab_SurfaceLines.AddRecord
  FTab_SurfaceLines.SetValue (Field_SurfaceLines_Shape,Number_NewRecord,PolyLine_SurfaceLine)
  FTab_SurfaceLines.SetValue (Field_SurfaceLines_FromNode,Number_NewRecord,Number_ScanPoint_Id)
  FTab_SurfaceLines.SetValue (Field_SurfaceLines_ToNode,Number_NewRecord,Number_ScanLines_Id)
 end
end
FTab_SurfaceLines.SetEditable(False)
```

Finally a hidden line removal algorithm retrieves the Surface_Edges. A TIN_Edge is considered as 'below' another TIN_Edge if it has a target point in common and the Z-value of the mid of the TIN_Edge is less than the Z-value of the comparing TIN_Edge. In this 1.5D example the hidden line removal algorithm is quite trivial, which can be easily extended to a 2.5D hidden face removal, as all observation lines are dropped perpendicular to the Target_Points. For 'real' 3D cases a more demanding hidden face removal algorithm has to be applied, as the observer point could be anywhere in the scene.

3. THE STIN METHOD FOR 2.5D SURFACES

3.1 Input target points and observation lines – Figure 3.1

The STIN-method for 2.5D surfaces is illustrated by a simple example where the observation lines of the 2.5D target points are dropped perpendicular from a certain height, see figure 3.1.
The algorithms consist of several steps, described in the following sections. These steps are:

Step 1: Input target points and observation lines
Step 2: Construct Conforming DTEN
Step 3: Transform TIN_Edges to Volume_Edges
Step 4: Find STIN_Edges on Surface
Step 5: Create STIN_Faces

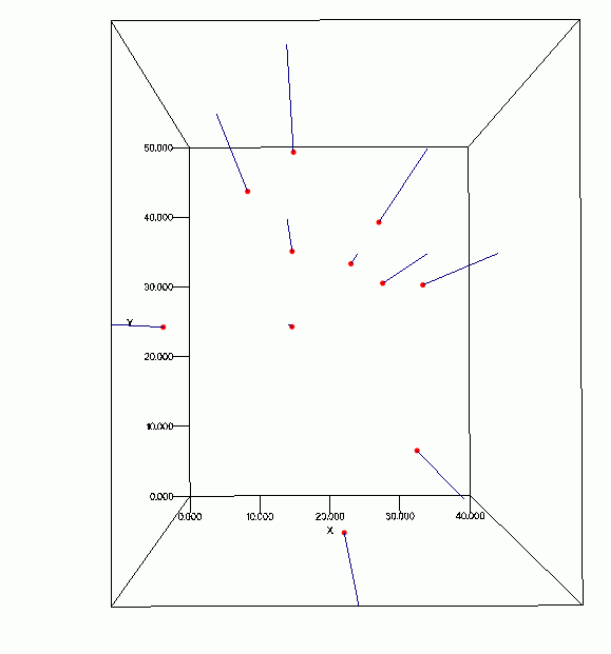

Figure 3.1
Target points and their observation lines – 2.5D case

3.2 Construct Conforming DTEN – Figure 3.2

The target points and their observation lines are included into a conforming Delaunay TEN. This 3D-network should result in a set of non-overlapping adjacent tetrahedrons, which should adhere to the following rules:

1) For each of the tetrahedrons in a Delaunay TEN the circumsphere should not contain any other point of the data set.
2) All observation lines are identified as edges in the Delaunay TEN.

The Delaunay TEN is calculated based on the incremental algorithm (Watson, 1981) and (Bowyer, 1981). This algorithm adds one point at a time to an (initial) valid Delaunay Triangulation. This algorithm is also known as the cavity algorithm, since its deletes all tetrahedrons that are not longer empty after the intersection of the new point. The cavity is then tetrahedronized by connecting the newly inserted point to all vertices on the cavity boundary. This procedure is available as an independent TEN-constructing program, as used in (Kraak, 1992).

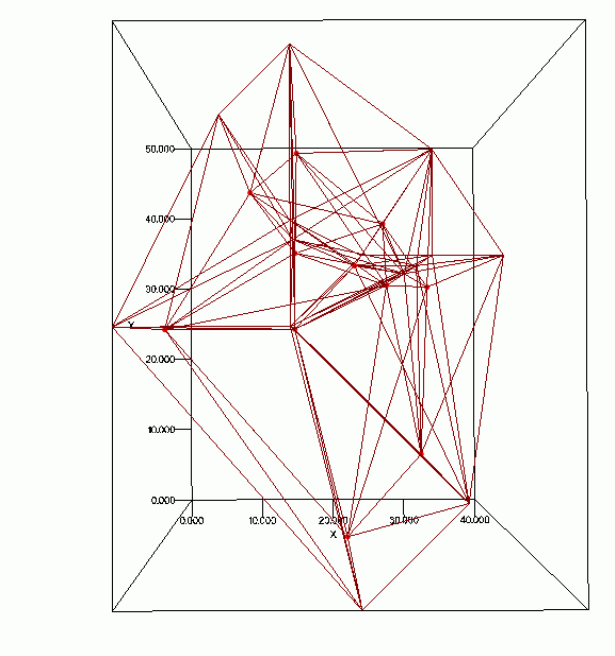

Figure 3.2
Conforming Delaunay TEN

To remain the observation lines within the TEN two possibilities are well-known, the constrained TEN and the conforming TEN. Within the constrained TEN the empty circumsphere requirement is loosed to allow an incorporation of the lines within the TEN as an entity. Conforming TENs on the other hand allow the insertion of so-called Steiner points. These extra points are iteratively added on the midpoints of the observation lines until each (part of) the observation line can be identified by an edge of the Delaunay TEN. Here is chosen for the Conforming TEN procedure, because of its minimalist approach (Calvalcanti, 1999), but also because of the use of these Steiner points in reconstructing the surface.

The 2.5D Volume beneath the target points and thus the Surface are found by the procedure, given by the code in the next step. Within this step a little trick is applied to derive the volume edges.

3.3 Transform TIN_Edges to Volume_Edges – Figure 3.3

All TEN_Faces are examined. If a TEN_Face has one Steiner point and two target points the algorithm will replace the Steiner point by the target point at the end of the observation line. The TEN_Edges of this TEN_Face are stored as Volume_Edges. Also the TEN_Edges of the TEN_Faces with three target points are stored as Volume_Edges and the remaining TEN_Faces are discarded.

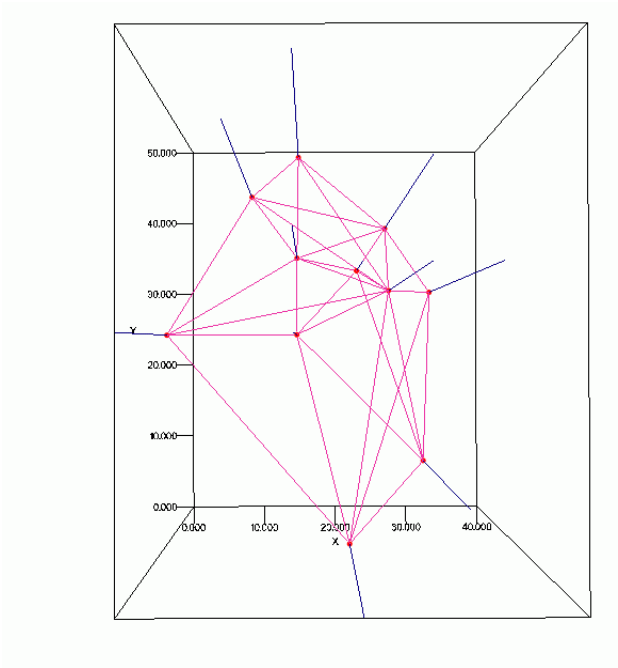

Figure 3.3
Volume_Edges of Conforming DTEN

3.4 Find STIN_Edges on Surface – Figure 3.4

In this step a hidden edge removal algorithm is applied on the Volume_Edges to retrieve the STIN_Edges. The algorithm applied projects the Volume_Edges to 2D and tests each one with the intersecting Volume_Edges. The intersection point is calculated in 2D, and the algorithm continues with calculation of the Z-value of the Volume_Edges at the intersection point. The Volume_Edge with the lowest Z-value is the furthest away from the observer and therefore not at the surface. This one is removed from the Volume_Edges. All remaining edges are declared to be STIN_Edges.

A problem arises in that some removed Volume_Edges are to be considered as STIN_Edges to obtain a complete and valid STIN_Surface. The removed Volume_Edges that have no 2D-intersection with another removed Volume_Edge are therefore promoted to STIN_Edges.

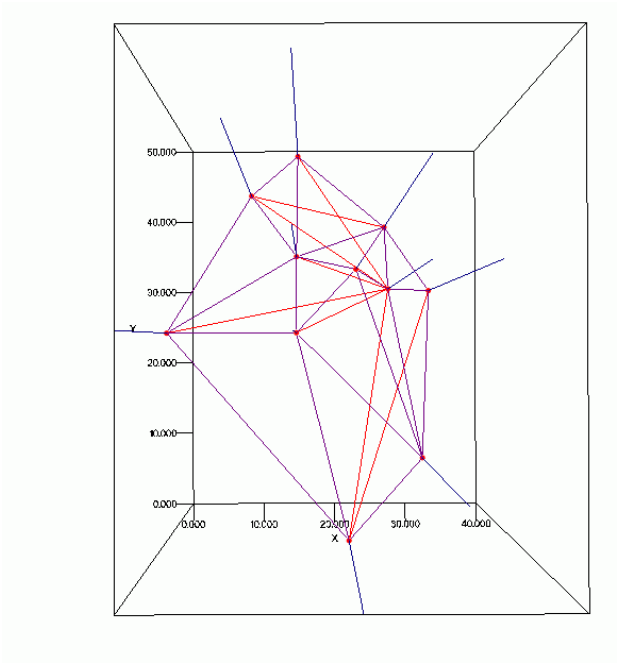

Figure 3.4
TEN_Edges on Surface

3.5 Create STIN_Faces – Figure 3.5

Finally the STIN_Faces (Surface triangles) are constructed. The STIN_Edges on the surface gives a complete and non-overlapping triangulated partitioning of the surface. The internal numbering of the nodes of the STIN_Edges gives enough information to construct the STIN_Faces.

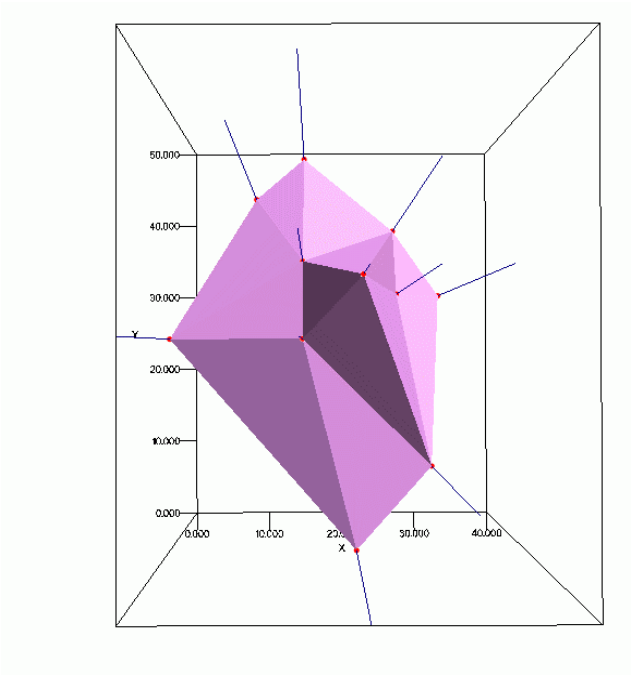

Figure 3.5
TEN_Faces on Surface

4. THE STIN METHOD FOR 3D SURFACES

4.1 Extending the STIN method to 3D – figure 4.1

The STIN method is described and explained with an example data set in 2.5D. The only step in the STIN method, which uses this property, is the hidden edge algorithm to find the STIN_Edges on the surface (Section 3.4). That fast algorithm could be applied because the observer was thought to be far above the scene and thus could the observation lines, like airborne laser scanning, be considered perpendicular.

If the observer is more or less within the scene, like in the case of terrestrial laser scanning, another hidden edge algorithm should be applied. This modification is necessary, because the target points are now really distributed in 3D. But that is the only modification. The remaining algorithm is not affected. This is demonstrated by the following example, where one observer from within the object scans several target points around it in 3D space.

4.2 Construct Conforming DTEN – Figure 4.2

The observation lines has to be cut off at a certain distance from the observer, for sake of unwanted site-effects (unlimited addition of Steiner points) in the calculation of the conforming DTEN.

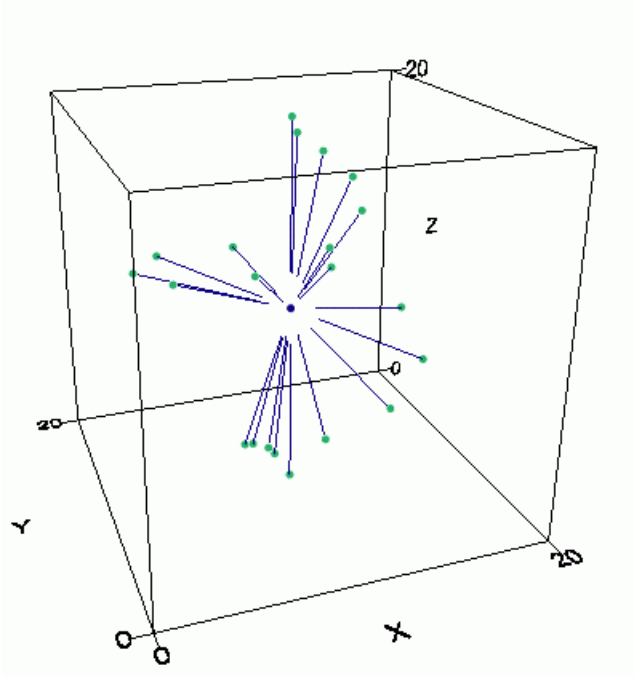

Figure 4.1
One observation point, many target points - 3D case

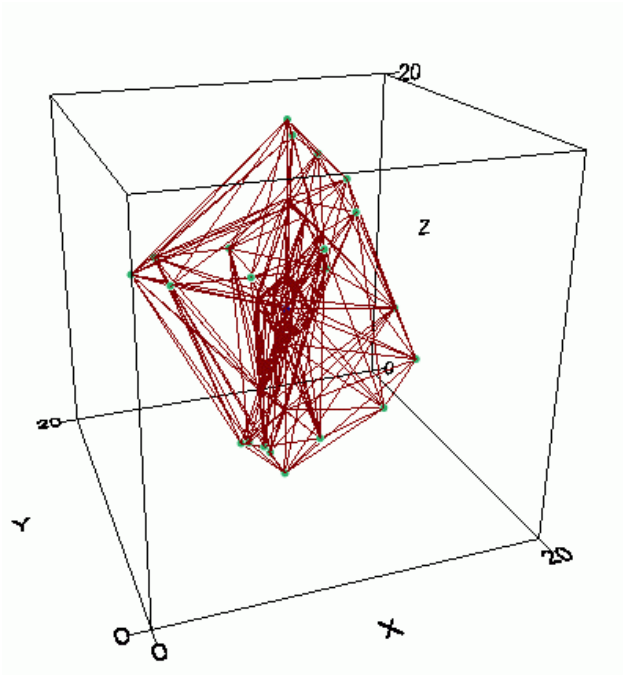

Figure 4.2
Conforming Delaunay TEN

4.3 Transform TIN_Edges to Volume_Edges – Figure 4.3

The same algorithm as in Section 3.3 is applied. Again all TEN_Faces are examined and transformed if one of the nodes is an added Steiner point. If that is the case the target point at the end of the observation line replaces this node. All TEN_Edges of the TEN_Faces are now stored as Volume_Edges.

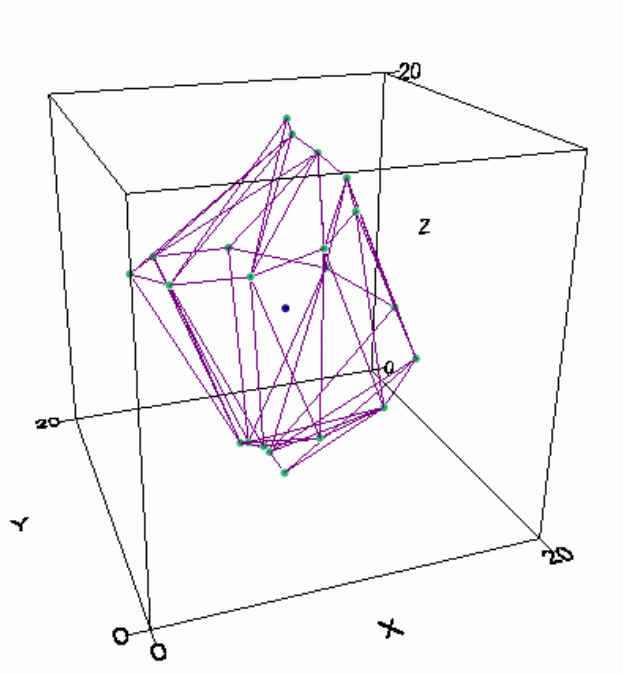

Figure 4.3
Volume Edges of Conforming DTEN

4.4 Find STIN_Edges on Surface – Figure 4.4

To determine which TEN-Edges are on the surface a full 3D hidden edge algorithm had to be applied. This algorithm is a straightforward three dimensional generalization of the 2D method presented in (Aftosmis) and described by (O'Rourke, 1994). Figure 4.4 gives an illustration of this method.

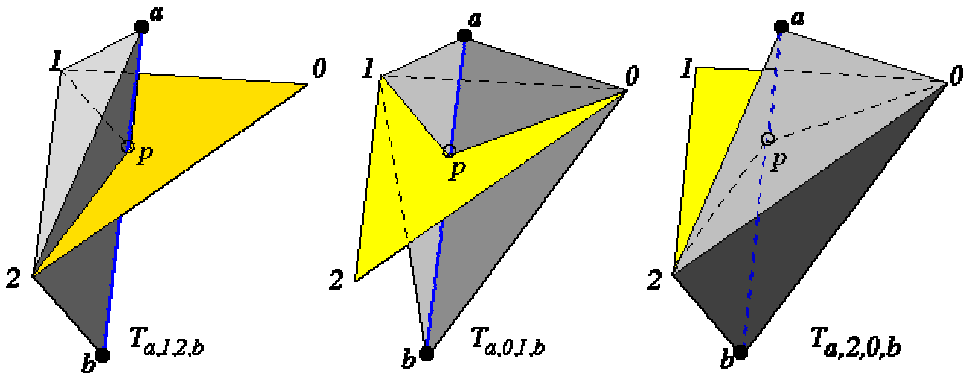

Figure 4.4
Test visibility Edge (a,b) and Edge (1,2) from Observer (0)

Each Volume_Edge is tested against all other Volume_Edges. To determine whether or not Volume_Edge (a,b) is in front of Volume_Edge (1,2) given the observation from point (0) three tetrahedrons T(a,1,2,b), T(a,0,1,b) and T(a,2,0,b) are constructed. The Volume_Edge (a,b) is in front of Volume_Edge(1,2) if the sign of the determinant of these tetrahedrons is the same (all positive or all negative). These Volume_Edges are eliminated and the remaining Volume_Edges are on the surface and declared as STIN_Edges.

```
' Description: IntersectEdges
' Return True if Triangle (PointZ_0,PointZ_1,PointZ_2) is intersected by Edge (PointZ_q, PointZ_r)

PointZ_0 = SELF.Get(0)
PointZ_1 = SELF.Get(1)
PointZ_2 = SELF.Get(2)
PointZ_q = SELF.Get(3)
PointZ_r = SELF.Get(4)

Boolean_Intersect = False

'check p(q) is left of V(123) and p(r) is right of V(123) or p(q) is right of V(123) and p(r) is left of V(123)
Number_SignA = Av.Run("AvTenTin.VolumeSign",{PointZ_0,PointZ_1,PointZ_2,PointZ_q})
Number_SignB = Av.Run("AvTenTin.VolumeSign",{PointZ_0,PointZ_1,PointZ_2,PointZ_r})
if (((Number_SignA < 0) and (Number_SignB > 0)) or ((Number_SignA > 0) and (Number_SignB < 0))) then
  Number_Sign0 = Av.Run("AvTenTin.VolumeSign",{PointZ_q,PointZ_0,PointZ_1,PointZ_r})
  Number_Sign1 = Av.Run("AvTenTin.VolumeSign",{PointZ_q,PointZ_1,PointZ_2,PointZ_r})
  Number_Sign2 = Av.Run("AvTenTin.VolumeSign",{PointZ_q,PointZ_2,PointZ_0,PointZ_r})
  if (((Number_Sign0 > 0) and (Number_Sign1 > 0)) and (Number_Sign2 > 0)) then
    Boolean_Intersect = True
  end
  if (((Number_Sign0 < 0) and (Number_Sign1 < 0)) and (Number_Sign2 < 0)) then
    Boolean_Intersect = True
  end
end 'if
```

```
return (Boolean_Intersect)
```

```
' Description: VolumeSign
' Returns sign of volume of tetrahedron given by four points
' See: Computational Geometry in C; code 4.16; Also: Equation 1.16; and Exercise 1.3.9.1

PointZ_0 = SELF.Get(0)
PointZ_1 = SELF.Get(1)
PointZ_2 = SELF.Get(2)
PointZ_3 = SELF.Get(3)

Vector_A = Vector.Difference(PointZ_0,PointZ_3)
Vector_B = Vector.Difference(PointZ_1,PointZ_3)
Vector_C = Vector.Difference(PointZ_2,PointZ_3)

' calculate determinant
Volume = Vector_A.DotProduct(Vector_B * Vector_C)

if (Volume > 0) then
  VolumeSign = 1
else
  if (Volume < 0) then
    VolumeSign = -1
  else
    'Volume = 0
    VolumeSign = 0
  end
end

Return (VolumeSign)
```

Again (as in the 2.5D example) some of the removed Volume_Edges are needed to obtain a complete and valid STIN_Surface. Each removed Volume_Edge that is visible from the observation point given the set of STIN_Edges is added to this set.

4.5 Create STIN_Faces – Figure 4.5

The set of STIN_Edges gives a complete and non-overlapping partitioning of the Surface and all STIN_Faces are constructed. However, one extra test is necessary. It is possible that one or more of the (original) observation lines is intersecting a STIN_Face, in which case the intersected STIN_Face has to be deleted.

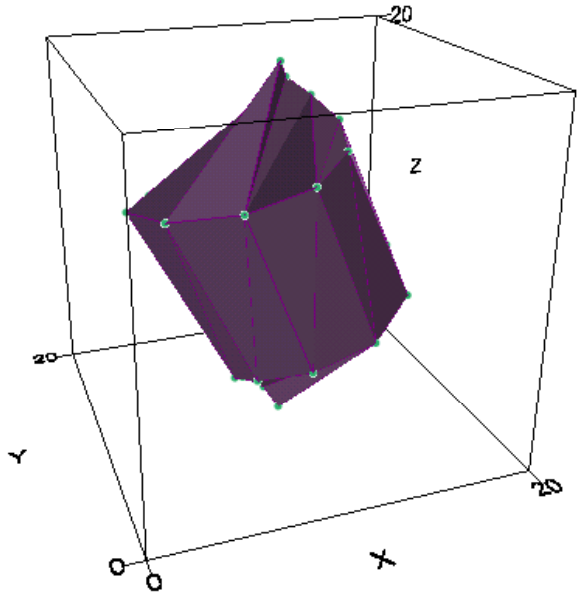

Figure 4.5
TEN_Faces on Surface

4.6 Examples – Figures 4.6 and 4.7

In this example all steps of the STIN method are made visible. Given are the eight target points on the corners of a cube and six target points slightly pushed inside the cube. The quest is to reconstruct the surface, given an observation point. In figure 4.6 this observer is in the mid of the cube and the STIN Surface is found given the procedure described in the former sections.

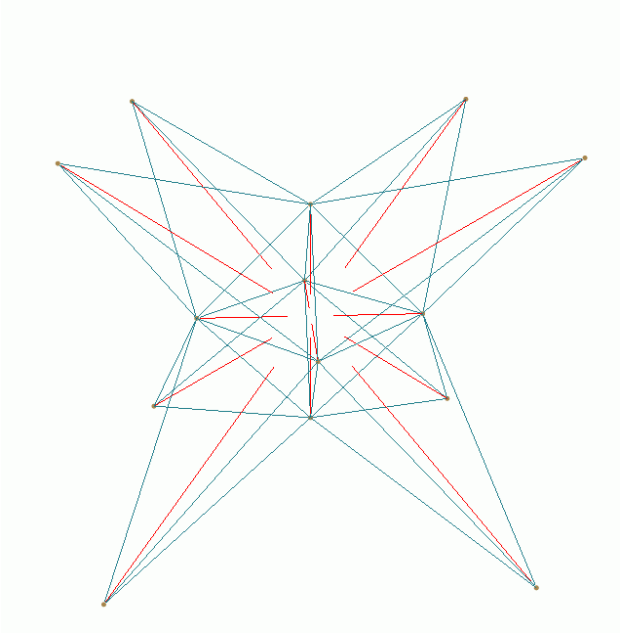

Figure 4.6
STIN Surface of 'Pushed' Cube with observation inside

Figure 4.7 shows the reconstructed STIN Surface of almost the same dataset (one of the pushed target points is eliminated to give sight inside the cube)

and the effect of the position of the observer (slightly below the position of the eliminated target point). The observer is now outside the object and a complete different - but still valid - surface is derived.

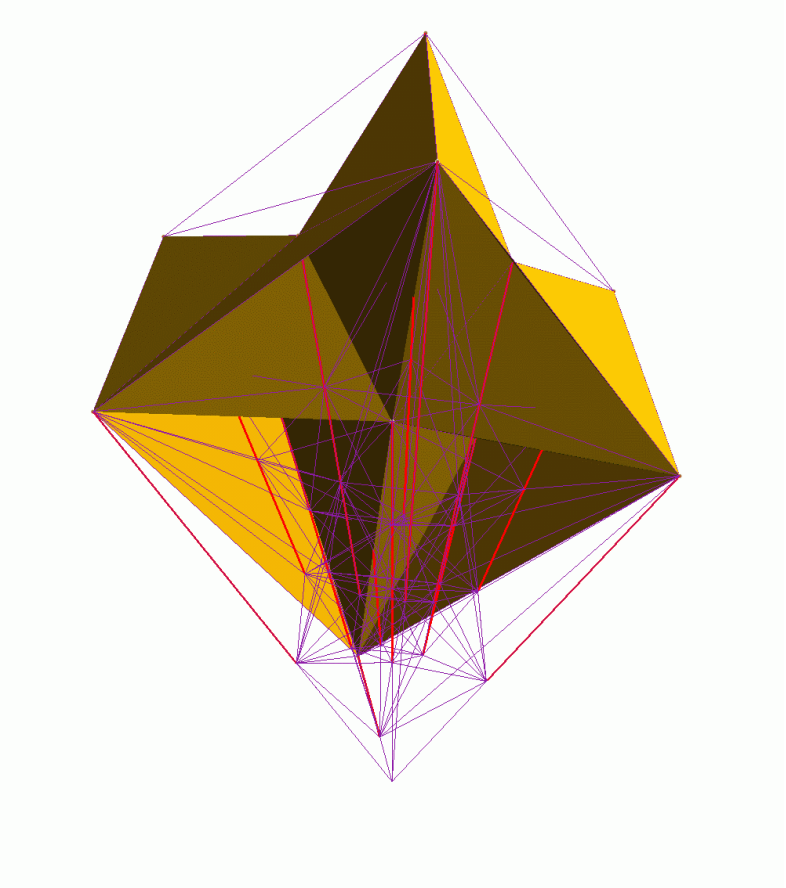

Figure 4.7
STIN Surface of 'Pushed' Cube with observation outside

5. CONSIDIRATIONS IN USING ARCVIEW AND AVENUE

5.1 Why still developing in ArcView and programming in Avenue?

The STIN method is developed within ArcView 3.2 and programmed with its language Avenue. The reasons for this environment are manifold. When Avenue came available without restrictions within ArcView 2, it has been since the choice in writing geo-computational prototype programs by the first author of this paper. That is because Avenue offers a rich set of spatial datatypes among the available classes, in combination of a quite broad implementation of datastructures and the storage of the spatial data within easily accessible shapefiles.

The second reason to use ArcView is its 3D-Analyst extension, which gives full access to Triangulated Irregular Networks (TINs). This extension creates TINs according to the Delaunay criterion, and is able to deal with constrains. These breaklines and polygons are maintained by adding as many nodes (Steiner points) as needed to fulfill the empty circle criterion, where each circum circle through the three nodes of a triangle should be empty. These added Steiner points in this so-called conformal Delaunay TIN are used as a kind of anchor in the STIN method and are therefore very useful. The created TIN is

maintained in a kind of winged edges datastructure, which can be inquired by the instance requests for getting the attributes of the TIN and the TINEdge classes. Therefore it is possible to build FThemes of feature classes PointZ, PolylineZ and PolygonZ to store the TIN nodes, edges and faces within FThemes with all their attributes.

The third motive in using ArcView and 3D-Analyst is the 3D-Scene. It has very basic functionality in the sense of manipulation of the point of view and the scene itself, but it has one big advantage in querying the displayed TIN Shapefiles. One can easily identify and select the nodes, edges and faces of the FThemes within the 3D-Scene. When writing an algorithm which builds a STIN a lot of checking is necessary to test the correctness and completeness of the derived STIN. And to improve the limitations of the scene manipulation capabilities some very fine extensions are available through the ArcScripts.

There are however some main disadvantages in the implementation of the TINs and the features classes with Z-values. First of all the TINs are stored in an ESRI priority data format and not in shapefiles. This is probably because the TIN implementation is a port of the ArcInfo TIN construction method. And although it is as mentioned possible to build the FThemes yourself a TIN export to PointZ (nodes), PolylineZ (edges) and PolygonZ (faces) shapefiles should be nevertheless as class requests.

The shape classes that represent (X,Y,Z) points inherit from the regular (X,Y) shape classes. One could argue that in relating two shapes classes the Z-value of the features should be taken into account. But in the calculation of the intersection of two PolyLineZ shapes only the planimetric part is taken into account and the Z-value is forgotten. This limitation of the use of the 3D shape classes given by the 3D-Analyst is the second disadvantage.

The third disadvantage in creating TINs by the 3D-Analyst is the limited link between the created TIN and the features it originated from. One can specify an aValueField within the AddShapesFromFTab request. However this ValueField is only used by Mass Points and Polygons, but not for PolyLines. So the ValueField is maintained within the Nodes and Triangles of the TIN only, but not for the TIN_Edges. This limitation in linking the attributes values of the PolyLines used as constraints to the derived TIN_Edges is difficult to handle. And the linkage with an aValueField is of course a kind of a work around. There should be by preference a more sophisticated implementation available in maintaining the link between the original FThemes and the TIN.

5.2 The 3D Analyst and TENs

The TIN construction capabilities of ArcView 3D Analyst are used in the STIN Method for 1.5D Surfaces (see chapter 2). As this 1.5D Surfaces are only created to give a kind of perception to the STIN method it has no practical use. The 2.5D and 3D Surfaces however are based on Tetrahedronized Irregular Networks (TENs). At the moment no extension whatsoever is available within ArcView to construct this kind of TENs. That is argumentative because the use of TENs within GIS is still in development.

But is makes life a little bit complicated. That is because an extensive reading and writing of input and output files is needed to an external TEN-algorithm, available as an old-fashion standalone program. This way to communicate with external programs has off course some main drawbacks. Reading an external file and creating FThemes out of it is a taskforce, because the geometry of the shapes has to be generated. And as TENs have a tendency to be huge (thousands of tetrahedrons) this materialization takes a while. A better approach should be to write the shapefiles directly within the TEN-construction program.

In conjunction to these troubles no tetrahedron feature type exists. A tetrahedron could be modeled as a PolygonZ with multiple (thus four) rings. But the 3D Scene has quite some difficulties in rendering this kind of shapes properly. For that reason tetrahedrons are modeled based on their TEN_Edges, so each unique (where the from_node is literally less than the to_node) TEN_Edge gives one PolyLineZ in the FTheme. This approach complicates the identification and selections of tetrahedrons within a 3D Scene.

Finally one could argue to use the 3D Analyst and ArcScene environment of ArcGIS. Although this environment offers better visualization and navigation tools, it uses the same geometric datatypes as the 'old-fashion' ArcView GIS. These advantages are not enough to switch to the ArcGIS environment at the moment, in conjunction with the loss of the experience and available scripts gained in the Avenue language and the need to program in VBA.

6. CONCLUSIONS AND RECONMANDATIONS

The standard Delaunay TIN (DTIN) method has to be handled with care for surface reconstruction purposes, as the Z-value of the target points is not considered in the construction. Within the Surface TIN (STIN) method the Z-value of the target points is taken into account along the position of the observer and the observation lines. The surface is created and derived within a Tetrahedronised Irregular Network (TEN) in three dimensions. This method lines up with all kinds of Data Dependent Triangulations (DDTINs). The STIN method is capable to reconstruct surface out of a given point cloud in 3D as long as the observation point is known.

Current research is undertaken to extent the method to:

- Handle large data sets. The STIN method is now available in a prototype environment written in the scripting language Avenue of ArcView 3.2a and own TEN-constructing software. It is possible to create surfaces to 1000 points in reasonable time. The use of a more robust and scalable environment as the Computational Geometry Algorithm Library (CGAL) should be considered.
- Handle more observation points for the 3D-surface reconstruction procedure.
- Combine 2.5D surfaces with full 3D objects for terrain modelling applications.

Furthermore research is undertaken to:

- Give a formal proof of the correctness of the reconstructed surfaces (no holes or overlapping parts).
- Compare the 2.5D STIN surfaces with results from Data Dependent Triangulations.

REFERENCES

Aftosmis, M.J., Intersection of Generally Positioned Polygons in R3, http://people.nas.nasa.gov/ ~aftosmis/cart3d/bool_intersection.html

Bowyer, A., Computing Dirichlet tessellations, The Computer Journal, 24(2):162–166, 1981.

Calvalcanti, P.R. and Ulisses T. Mello, Three-dimensional Constrained Delaunay Triangulation: a Minimalist Approach, Proceedings of the 8th International Meshing Roundtable, Lake Tahoe, California, October 1999, p.119-129

CGAL, www.cgal.org

Dyn, N., D. Levin and S. Rippa, Data dependent triangulations for piecewise linear interpolation. IMA J. Numer. Anal. 10 (1990), pp. 137-154.

ESRI, www.esri.com

Kraak, M.J., and E. Verbree, Tetrahedrons and Animated Maps in 2D and 3D Space, in Proceedings 5th International Symposium on Spatial Data Handling, 1992, pp. 63-71.

Lenk, U., Optimisation Criteria for Degenerated Delaunay Triangulations, proceedings GIScience 2000, Savannah, Georgia, USA, October 28-21, 2000.

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

Shewchuk, http://www-2.cs.cmu.edu/~quake .defs.html#cdt

Verbree, E. and P.J.M. van Oosterom, 2001, Scanline Forced Delaunay TENs for surface representation, ISPRS Workshop on Land Surface Mapping and Characterization using Laser Altimetry, Annapolis, Maryland, USA, pp. 45-51.

Watson, D.F., Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes, The Computer Journal, 24(2):167–172, 1981.