# Using Constraints in Delaunay and Greedy Triangulation for Contour Lines Improvement

Ivana Kolingerová[1], Václav Strych[2], and Václav Čada[2]

[1]Department of Computer Science and Engineering
[2]Department of Mathematics
University of West Bohemia, Plzeň, Czech Republic
kolinger@kiv.zcu.cz, http://iason.zcu.cz/~kolinger

**Abstract.** Automatic computation of contour lines on a triangulation is a difficult problem because due to input data digitization and the type of triangulation used, some triangles can be a source of a strange behaviour of the contour lines. In this paper, we show what problems can appear in contour lines when Delaunay or greedy triangulations are used and how the contour lines can be improved using constraints in the triangulation. We improved contour lines by manually imposing constraints in a triangulation editing program. Automation of this process is a next step of our work.

## 1 Introduction

A computation of contour lines on a triangulation is a necessary part of GIS programs. However, automatically obtained contours are rarely satisfactory. For an experienced expert, it is easy to recognize the problems and to correct them manually, but at present it is nearly impossible to make this recognition and correction automatic.

This paper makes one step to a future fully automatic solution: it shows results of using constraints – constrained edges prescribed into the triangulation on the places where contour lines computed on the original triangulation are not satisfactory. Detection of these places is manual, local recomputation of triangulation and of contour lines is automatic.

Section 2 describes background. Section 3 shows main problems in contour lines on Delaunay and greedy triangulations. Section 4 shows how to improve the contour lines using constraints, Section 5 presents results, Section 6 concludes the paper.

## 2 Triangulations and Contour Lines Computation

**Def. 1. A triangulation.** A triangulation T($P$) of a set $P$ of $N$ points in the Euclidean plane is a set of edges $E$ such that

- no two edges in $E$ intersect at a point not in $P$
- the edges in $E$ divide the convex hull of $P$ into triangles.

There exist many types of triangulations. The most popular one - due to good properties of its triangles and simple and well-studied computation, usually in $O(N)$

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 123–130, 2004.

expected time and $O(N \log N)$ worst case – is the Delaunay triangulation. Another well-known triangulation is the greedy triangulation.

**Def. 2. Delaunay triangulation (DT).** The triangulation DT($P$) of a set of points $P$ in the plane is a Delaunay triangulation of $P$ if and only if the circumcircle of any triangle of DT($P$) does not contain a point of $P$ in its interior.

Delaunay triangulation ensures maximizing minimum angle of each triangle as well as of the whole triangulation. Therefore, it produces the most equiangular triangulation of all. Other positive features exist, as described in [2, 3, 9]. Algorithms for DT computation can be found in [6, 12, 13].

**Def. 3. Greedy triangulation (GT).** The triangulation GT($P$) of a set of points $P$ in the plane is a greedy triangulation if it consists of the shortest possible compatible edges where a compatible edge is defined to be an edge that crosses none of those triangulation edges which are shorter than this edge.

The main problem connected to GT is its time complexity. It can be computed in $O(N^2 \log N)$ time using demanding algorithms. $O(N)$ expected time solution is possible only for uniform data. Effective algorithms can be found in [4, 8, 10].

DT and GT use for computation only planar information given by x and y coordinates of the input points, however, resulting meshes are of good quality, with one exception: a terrain with a steep slope. Such a kind of data needs a special triangulation, taking into consideration also heights of points (z-coordinates), angles between triangle normals, etc. These triangulations are called **data dependent triangulations (DDT)** and were established in [5].

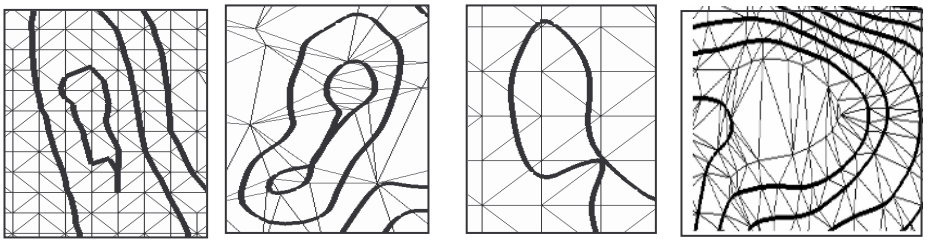
One more concept used in triangulations is a **constrained edge** (a **constraint**). It is an edge prescribed to be used in a triangulation. This approach is useful to adjust a triangulation to some domain limitations, such as a break, a polygonal boundary, etc. DT as defined in Def. 2 is not able to consider constraints - its definition has to be modified: in the **constrained Delaunay triangulation (CDT)**, only non-constrained edges are checked on the empty circumcircle property. For the CDT algorithms see [11, 1]. It is very easy to use constrained edges in GT: such edges are accepted into the triangulation first, then a usual greedy construction continues.

There exist more possible algorithms how to compute **contour lines** on a triangulation, e.g. [7]. As the approach described in this paper is independent of the contour line computation method, we will not explain details. Generally, we are looking for intersections of the triangulation with the height range $<z_{min};z_{max}>$ on which the contour lines are to be computed. Contour lines computed on triangulation are piecewise linear - polylines, later they are usually smoothed out by an interpolation. For one particular $z=z_h$, the contour lines may be formed by open segments – polylines starting and ending at the triangulation boundaries, and by closed segments - polylines surrounding some extreme sites in the terrain model.

## 3 Main Problems in Contour Lines on DT and GT

One of the main criteria when producing triangulations for contour lines is avoiding too long and skinny triangles. As GT does not consider shape and angles of the triangles, we expected problems with the contour lines computed on this triangulation. Surprisingly, we came across the situations where the contour lines on GT were better than on DT. Now we will present the problems which we identified in the countour lines computed on DT. We should note that for more pleasant figures, the contour lines have been smoothed by splines, however, the described effects are present also on non-smoothed contour lines, so they are not caused by the interpolation.

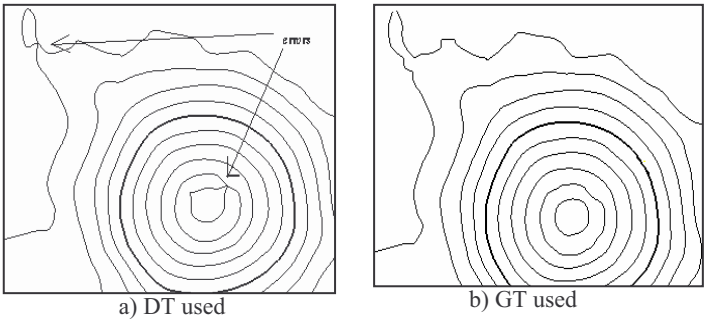
While testing DT, we found the following types of errors in contour lines. Fig. 1a) shows the situation where the contour lines lie on one edge of the triangle and then return back on the same edge. Fig. 1b) shows bad contour lines where two tops of hills were connected into one contour line. Fig. 1c) shows a problem of „too close lines". An interesting type of detected errors is in Fig. 1d). This error is usually caused by the triangles which have all their vertices in the same height.



a) Identical parts b) Wrong connection c) Improper closeness d) An error obtained due to horizontal triangles

**Fig. 1.** Errors detected on contour lines when using DT

Fig. 2 shows contour lines generated from a regular grid. It can be seen that in case of DT there appear two problematic places which are correct when using GT. Fig. 3a) documents again a problem with nearly horizontal triangles. Contour lines on GT in Fig. 3b) are a bit better but not completely correct, either.




a) DT used b) GT used

**Fig. 2.** Comparison of contour lines on DT and GT

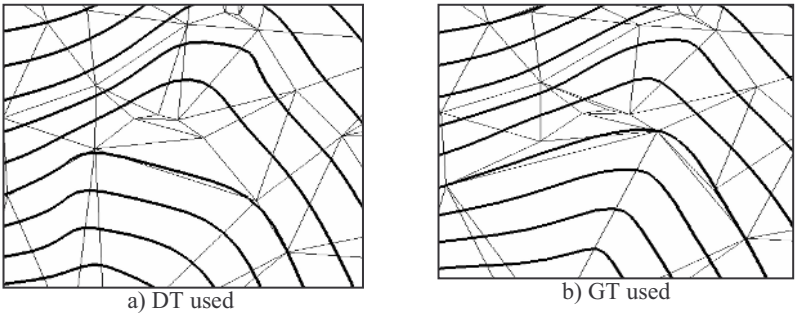



**Fig. 3.** Another problematic situation with nearly horizontal triangles

According to these two cases, it might seem that GT enables to create better contour lines. Not each time this is true. In most cases, contour lines are similar in both triangulations. There appears a problem for GT if a narrow triangle is created. In such a case the contour lines twist unpleasantly, see Fig. 4b), or break, see Fig. 4d), Figs. 4a), 4c) show DT for comparison.

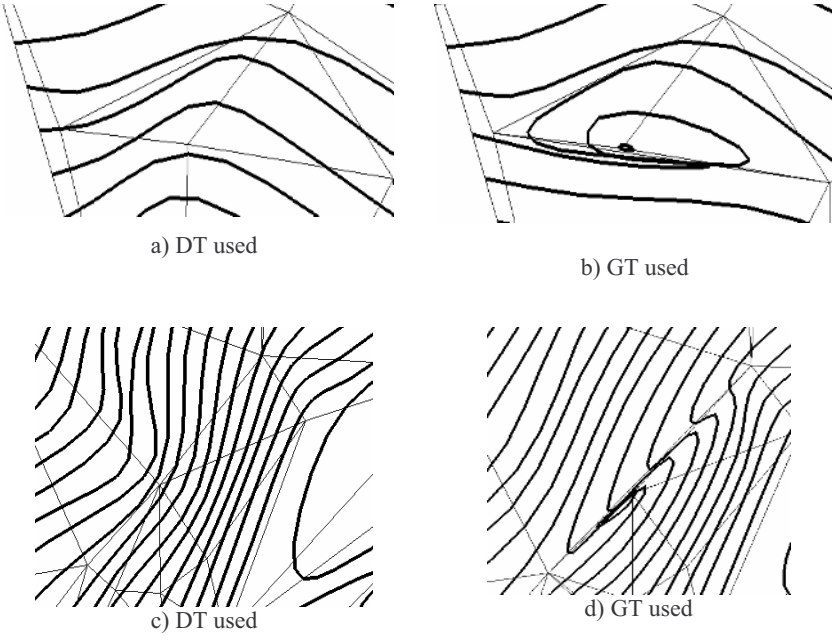



**Fig. 4.** Also GT may cause contour line problems: narrow triangles and breaks

If we want to evaluate the triangulations according to their handling of problem places, it is necessary to say that GT creates less of such errors. The reason is that DT triangles are usually oriented by their longer edges against the slope direction. It causes a creation of horizontal or nearly horizontal triangles. In GT, some triangles are oriented by their longer edges into all possible directions and vertices with different

height can be connected. It prevents from horizontal triangles creation. This is not the only problem that may appear but it is the most visible one. This visibility is on one hand a disadvantage because several such places cause bad appearance of the whole map; on the other hand, it allows to detect such places easily and to edit them. Skinny triangles, more often appearing on GT than DT, have the opposite property; they are not that much visible at first sight, and, therefore, also more difficult to find, and, what is more important, they are more often than horizontal triangles.

For comparison, Fig. 5-7 show contour lines computed by the Erdas Imaging software, by DT and GT in our implementations. It can be seen that even a professional software sometimes produces problems, see a 'tooth' on Fig. 5. Boundaries of Figs. 6 and 7 are wrong due to convex hulls in the triangulation. For good contour lines, convex hulls should be avoided and replaced by domain boundary preservation over constraints. If we do not take into account boundaries, the contour lines obtained on DT are comparative to the professional software and they do not contain the 'tooth effect'. GT results are worse – there are two places where thin triangles spoil the shape of contour lines.

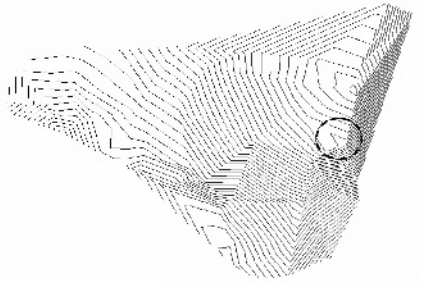


**Fig. 5.** Erdas

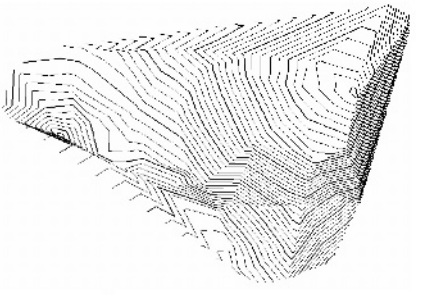


**Fig. 6.** DT

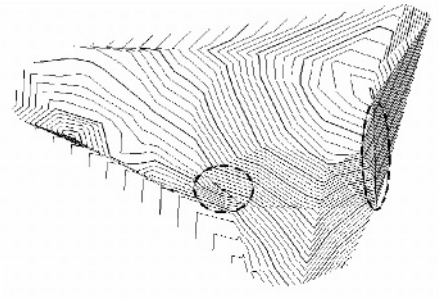


**Fig. 7.** GT

As according to our experiments, neither GT nor DT can provide perfect contour lines, DT should be recommended as a better choice as its computation is cheaper and

some of problems on contour lines are less often and easier to be corrected. However, it is visible that none of the considered triangulations is perfect; therefore, other ways to improve the contour lines have to be searched. Next section will present how the contour lines quality can be improved using constrained edges.

## 4 Improving Contour Cines by Constraints

Let us recall that constrained edges (or constraints) are edges which are prescribed in advance to be used within a triangulation. They are necessary to achieve an outer shape of the triangulated area different from the convex hull of the triangulated points or to include some breaks or ridges into the triangulation. We will show now that such 'enforced' edges are also important to increase a quality of contour lines.

The constraints are to be used to change the triangulation on the places where contour lines are incorrect or improper. Such places must be identified manually by inspecting the (automatically computed) contour lines. It would be also possible to detect wrong contour lines automatically by a set of general criteria, however, improper places are for experienced cartographic experts easy to be seen but difficult to be quantified, and formulation of criteria for automatic processing needs quantification. This is still an open problem for our future contour lines research. Therefore, our solution uses an editing program which displays the triangulation together with contour lines computed on it. The user – a cartographic expert – inspects the contour lines and if he detects some problems as described in Section 3, he may prescribe a constrained edge between a pair of triangulation vertices.

Insertion of a constrained edge means that the triangulation edges intersecting the newly inserted edge have to be deleted and the hole in the triangulation retriangulated. Then, in case of DT, the newly obtained triangles have to be checked whether they hold empty circumcircle property; eventual changes may spread into the whole triangulation, although usually they are only local. After the triangulation changes, also contour lines on changed triangles have to be recomputed.

A fundamental step for constraints insertion is an edge flip. Let us suppose we have two triangles ($v_1v_2v_3$), ($v_2$,$v_4$,$v_3$) sharing an edge ($v_2v_3$) and we need to replace this edge by the constraint $v_1v_4$. If ($v_1v_2v_4v_3$) is a convex quadrangle, the flip is possible, and vice versa. This condition ensures that the new edge will not intersect other edges, not participating in the flip operation.

Retriangulation of the area intersected by a constrained edge according to [11] has steps as follows (see also an example in Fig.8):

1. Find a triangle where the constrained edge starts (the triangle which has one of the endpoints of the constrained edge as its vertex and at the same time is intersected by the constrained edge).
2. By walking from the starting triangle over its neighbouring triangles, find all edges intersecting the constrained edge and store them in a queue.
3. Loop until the queue is empty: pull one edge out of the queue, if the edge is a diagonal of a convex quadrangle, flip the edge. If the quadrangle is not convex, return the non-flipped edge into the queue to be solved later.

In case of CDT:

4. Insert all newly made edges except the constrained one into another queue.
5. Loop until the queue is empty: pull out an edge and check it on empty circumcircle criterion. If the criterion does not hold, flip the edge and insert the newly created edge into the queue.

After retriangulation, the contour lines for $<z_1;z_2>$ has to be recomputed where $z_1$ and $z_2$ are the minimum and the maximum heights on the triangles changed in retriangulation ($z_{min} \leq z_1 \leq z_2 \leq z_{max}$).

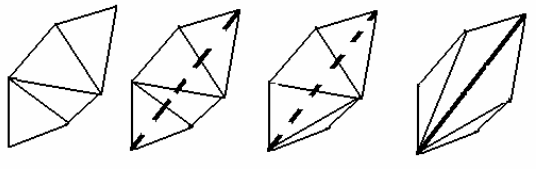



**Fig. 8.** Retriangulation, a) The original triangulation, b) The inserted constrained edge is dashed, c) After one edge flip, d) The resulting triangulation

## 5 Experiments and Results

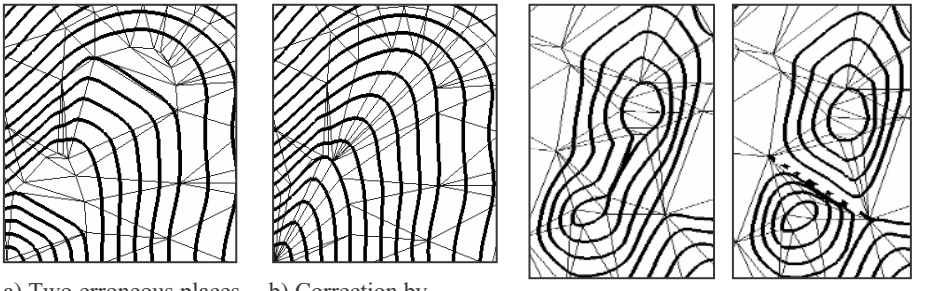
The described methods – DT, GT, CDT, contour lines computation and triangulation editor - were implemented as a set of programs in Delphi under Windows NT. The triangulation editor, besides others, allows inserting constraints on user request, as already described. Our DT and CDT implementation is of $O(N)$ expected time and memory complexity and allows to compute triangulations up to about 1mil. of points. As our GT implementation uses only brute force algorithm with $O(N^3)$ time and $O(N^2)$ memory complexity, we are able to test data sets up to 2000 points in GT. However, such size of data sets was enough to detect potential differences against DT results.

The use of constrained edges in DT as well as in GT brings a substantial improvement of contour lines shapes in critical places. See examples: Fig. 9a) shows two problems in the contour lines. Fig. 9b) shows correction of these situations thanks to the constrained edges. Fig. 9c) shows two wrongly connected hills, Fig. 9d) indicates correction of the situation by a constrained edge.

In the future, we would like to detect the problems in a triangulation automatically, so that the contour lines were computed on a mesh already optimized by constraints, without the need of manual corrections and of recomputation.

## 6 Conclusion

While inspecting the DT, GT and contour lines computed on them, we came to conclusion that GT brings more problems than advantages (time and memory complexity, skinny triangles) and, therefore, we recommend to use DT as a triangulation for contour lines computation. However, constrained edges are necessary to obtain satisfactory results. So far, we have to find problematic places manually, then insert constrained edges, recompute the attached part of the triangulation and of the contour lines. Automation of the contour lines improvements is our next research goal.



a) Two erroneous places in contour lines

b) Correction by constrained edges

c) Incorrect connection of two hills

d) Correction by a constrained edge insertion

**Fig. 9.** Two errors in contour lines and their correction with the use of constrained edges

## References


1. Anglada, M.V. : An Improved Incremental Algorithm for Constructing Restricted Delaunay Triangulations, Computers & Graphics 21 (1997) 215-223
2. Aurenhammer, F.: Voronoi Diagrams - a Survey of a Fundamental Geometric Data Structure, ACM Computing Survey, Vol.23, No.3 (1991) 345-405
3. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry. Algorithms and Applications, Springer-Verlag (1997)
4. Dickerson, M.T., Drysdale, R.L.S., McElfresh, S.A, Welzl, E.: Fast Greedy Triangulation Algorithms, Proc.10th Annual Symp. on Comp.Geom., ACM (1994) 211-220
5. Dyn, N., Levin, D., Rippa, S.: Data Dependent Triangulations for Piecewisse Linear Interpolation, IMA Journal of Numerical Analysis 10 (1990) 137-154
6. Kolingerová, I., Žalik, B.: Improvements to Randomized Incremental Delaunay Insertion, Computers & Graphics 26 (2002) 477-490
7. Krcho, J.: Modelling of Georelief and Its Geometrical Structure Using DTM: Positional and Numerical Accuracy (In Slovak), Georeliéf a geologické procesy, Q111, Bratislava (2001) 269-574
8. Levcopoulos, C., Lingas, A.: Fast Algorithms for Greedy Triangulation, BIT 32 (1992) 280-296
9. Okabe A., Boots B., Sugihara K.: Spatial Tesselations: Concepts and Applications of Voronoi Diagrams, John Wiley & Sons, Chichester New York Brisbane Toronto Singapore (1992)
10. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction, Springer Verlag New York Berlin Heidelberg Tokyo (1985)
11. Sloan, S.W.: A Fast Algorithm for Generating Constrained Delaunay Triangulations, Computers & Structures 47 (1993) 441-450
12. Su P., Drysdale R.L.S.: A Comparison of Sequential Delaunay Triangulation Algorithms, In: Proc. 11-th Annual Symp. on Comp.Geom., ACM (1995), 61-70
13. Žalik, B., Kolingerová, I.: An Incremental Construction Algorithm for Delaunay Triangulation Using the Nearest-point Paradigm, Int. J. Geographical Information Science 17 (2003) 119-138