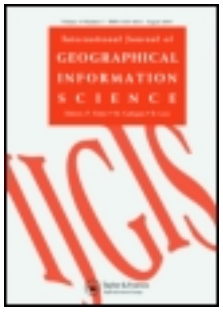

International Journal of Geographical Information Science

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tgis20

Sweep-line algorithm for constrained Delaunay triangulation

V. Domiter [a] & B. Žalik [a]

[a] Department of Computer Science, Faculty of Electrical Engineering and Computer Sciences, University of Maribor, Smetanova ul. 17, SI 2000 Maribor, Slovenia

Version of record first published: 19 Mar 2008.

To cite this article: V. Domiter & B. Žalik (2008): Sweep-line algorithm for constrained Delaunay triangulation, International Journal of Geographical Information Science, 22:4, 449-462

To link to this article: http://dx.doi.org/10.1080/13658810701492241

International Journal of Geographical Information Science
Vol. 22, No. 4, April 2008, 449–462

Research Article

Sweep-line algorithm for constrained Delaunay triangulation

V. DOMITER* and B. ŽALIK

Department of Computer Science, Faculty of Electrical Engineering and Computer Sciences, University of Maribor, Smetanova ul. 17, SI 2000 Maribor, Slovenia

(*Received 3 August 2006; in final form 25 May 2007*)

This paper introduces a new algorithm for constrained Delaunay triangulation, which is built upon sets of points and constraining edges. It has various applications in geographical information system (GIS), for example, iso-lines triangulation or the triangulation of polygons in land cadastre. The presented algorithm uses a sweep-line paradigm combined with Lawson's legalisation. An advancing front moves by following the sweep-line. It separates the triangulated and non-triangulated regions of interest. Our algorithm simultaneously triangulates points and constraining edges and thus avoids consuming location of those triangles containing constraining edges, as used by other approaches. The implementation of the algorithm is also considerably simplified by introducing two additional artificial points. Experiments show that the presented algorithm is among the fastest constrained Delaunay triangulation algorithms available at the moment.

Keywords: Algorithm; Constrained Delaunay triangulation; Sweep-line approach

1. Introduction

An old problem in engineering and science is to cover an area of interest with suitably-shaped triangles, what is considered as a triangulation. In geographical information system (GIS), the application of triangulation is varied as, for example, the representation of a digital terrain model using TIN (Saux *et al.* 2004), for the simulation of erosion (Sparovek and Schnug 2001), determining sun exposure, or optimal terrain guarding (Kaučič and Žalik 2004).

Delaunay triangulation (DT) is, without doubt, the most frequently applied when triangulation has to be established from scratch (i.e. from a set of points). Delaunay triangulation is an optimal triangulation, optimizing the minimal inner angles of all triangles guaranteeing well-shaped triangles. Optimality is assured by the so-called empty circle property, invented by Delaunay (1934).

In addition to points, sets of line segments are frequently prescribed as additional constraints for triangulation. A typical example is terrain generation from a set of iso-lines (Tse *et al.* 2004). Constrained Delaunay triangulation (CDT) is a generalization of DT, where the input consists of a planar straight-line graph (PSLG) described as $G=\{V, E\}$, where V stands for input points and E for input edges (*see* figure 1a).

*Corresponding author. Email: vid.domiter@uni-mb.si

International Journal of Geographical Information Science
ISSN 1365-8816 print/ISSN 1362-3087 online © 2008 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/13658810701492241

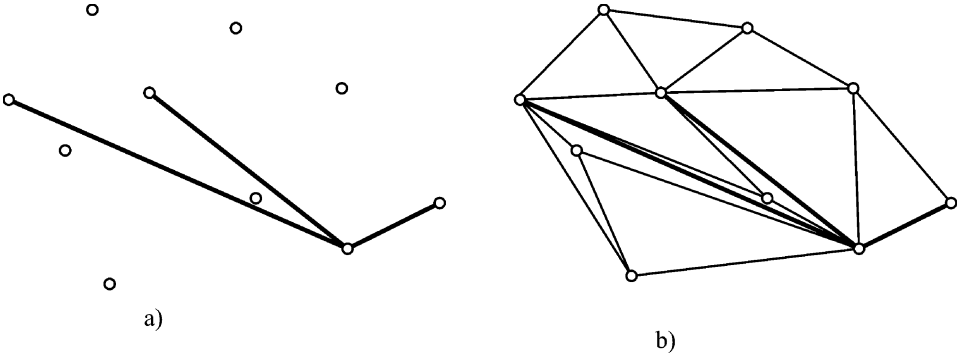

Figure 1. (a) Input PSLG graph and (b) corresponding CDT.

The constraining edges must become edges of the generated triangles, as shown in figure 1b. For this reason, a weaker Delaunay property has to be used. In figure 2a, the circumcircle of triangle t_2 contains vertices P_1 and P_3. P_1 violates Delaunay property, therefore, triangles t_1 and t_2 swap their common edge to form the new triangles t_1 and t_2 (*see* figure 2b). This process is realized as a recursive procedure called legalization and will be briefly considered later.

When legalizing t_2, we see that P_3 lies within its circumcircle, but behind constraining edge e. Because of that, P_3 is not visible from triangle t_2. Constrained Delaunay triangulation weakens the property, which is valid only for visible vertices.

Time-efficient algorithms are desirable because, usually, a considerable number of vertices and edges have to be triangulated in practice. The algorithms for constructing CDT can be classified as two groups:

(i) the approaches of the first group consider points and edges separately (Anglada 1997). Delaunay triangulation is constructed firstly on a given set of points V and the edges from set E are then inserted into DT. Those triangles which contain inserted edges are removed and replaced by new triangles, which fulfill the weaker Delaunay property. These solutions differ mainly in the first part, where different strategies for constructing DT can be used: divide-and-conquer (Chew 1989), sweep-line (Fortune 1987), incremental

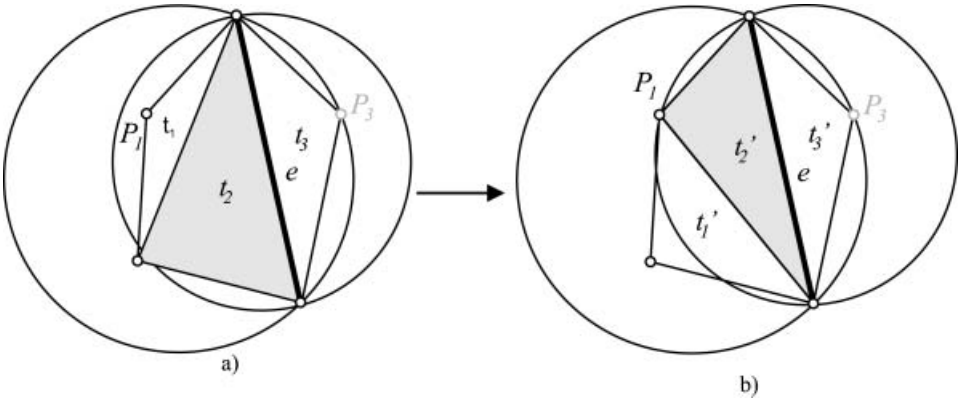

Figure 2. Weaker Delaunay property.

algorithms (Guibas *et al.* 1992), and high-dimensional embedding (Edelsbrunner and Seidel 1986);

Fast location of those triangles being pierced by the inserted edge has to be assured. Pierced triangles are located by a simple traversal using adjacency links of the triangles for considering their position against the constraining edge. To start the traversal, we first need to locate a starting triangle, which usually needs an efficient data structure for point-in-triangle location. Consequently, the memory requirement is considerable. However, when providing suitable data structure, a dynamic (incremental) manipulation of the inserted edges is possible, being desirable for many applications.

(ii) if data flow is not incremental, the algorithms that process points and edges simultaneously are more attractive. Such algorithms are faster, consume less memory and are less sensitive to data distribution. Unfortunately, their implementation is more complicated (Shewchuk 1999b).

This paper presents a new CDT algorithm based on the sweep-line paradigm. It processes the points and edges at the same time. However, insertion of any particular edge is postponed until it is swept entirely. In this way, determination of those triangles being pierced by the edge is efficient, simple and does not require any additional searching data structure. The presented CDT algorithm is based on a DT sweep-line algorithm presented by Žalik (2005). In addition to adapting it for CDT, this algorithm was also simplified. By introducing two additional artificial points, the number of possible algorithm states has been reduced, which makes its implementation simpler and easier.

2. Background

Different geometric problems have been efficiently solved by the sweep-line approach (de Berg *et al.* 1997). It consists of two steps. In the first step, the geometric elements (points and constraining edges in our case) are sorted. In the second step, the sweep-line starts to glide from $-\infty$ to $+\infty$ and stops at those event points where the required geometric tasks are performed and data structures are updated. Usually, the problem is completely solved in the half-plane already swept by the line and completely unsolved in the other half-plane. Unfortunately, DT is a case where the above statement is false. Namely, the circumcircle of a triangle whose upper vertex has just been hit by the sweep-line, exceeds the sweep-line and some of the points before the sweep-line could be located in the circle. For a long-time it was believed that the sweep-line approach could not be used for constructing DT. In 1987, Fortune (1987) negated this by proposing a very clever sweep-line algorithm. Actually, his algorithm constructs a Voronoi diagram – a dual of DT.

Fortune solved the problem by considering it in three-dimension (3D). At each point being triangulated, he placed a cone and observed the intersections between the cones. These intersections are in the form of parabolic arcs. If these arcs are projected back on the plane, the Voronoi diagram is obtained (and with this DT also). Fortune's algorithm is among the most efficient algorithms, but its implementation is far from simple. Recently, another sweep-line algorithm was published (Žalik 2005), combining the sweep-line paradigm with so-called Lawson's legalization (Lawson 1977). Lawson showed that any triangulation can be transformed into DT by applying the empty circle test on all pairs of the triangles.

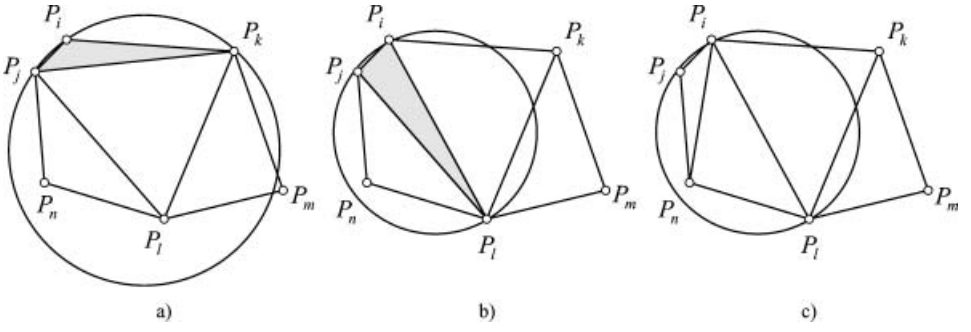

Figure 3. Empty circle property and the legalization.

If the empty circle property is violated, the common edge of the two triangles is swapped, as can be seen in figure 3. Delaunay triangulation is obtained when this procedure is recursively applied on all edges of the triangulation. Let us observe an example. In figure 3a the circumcircle of triangle $\Delta P_iP_jP_k$ contains vertex P_l of the neighboring triangle $\Delta P_jP_lP_k$. Two new triangles $\Delta P_lP_iP_j$ and $\Delta P_kP_iP_l$ are obtained after swapping the common edge. The swapped edge P_iP_l belongs to the DT (it is legal) while the remaining edges P_iP_j, P_jP_l, P_lP_k and P_kP_i (*see* figure 3b) determine the pairs of triangles, which have to be recursively checked for the empty circle property. When checking edge P_jP_l, Delaunay property is violated again. The circumcircle of triangle $\Delta P_lP_iP_j$ contains vertex P_n which results in the new triangles $\Delta P_nP_iP_j$ and $\Delta P_lP_iP_n$ (*see* figure 3c). If triangulation is already DT and a new point is added into the triangulation (this is also the case at the sweep-line approach), only a few edge swaps are needed for the legalization (de Berg *et al.* 1997). However, CDT adds another limitation. The legalisation process is also stopped in the case where two triangles share the constraining edge.

The sweep-line DT algorithm combined with Lawson's legalization appeared very efficient and relatively simple to implement. In order to cover all aspects of this paper, we will briefly discuss its main idea in section 3.1, using a small example. Thereafter, in continuation, our extension of this algorithm towards CDT is explained in detail.

3. The algorithm

3.1 *Sweep-line DT algorithm using an advancing front*

In the example in figure 4, the sweep-line has already passed the first 10 points. They are surrounded by two polylines:

(i) the lower border determines a part of the convex hull (plotted in dash-dotted style);
(ii) the upper border polyline (plotted in dashed style) is considered as an advancing front.

All points between the lower convex hull and the advancing front are triangulated according to the empty circle property, i.e. they form DT. When the sweep-line meets the next point (vertex P_{11} in our example), the algorithm performs the following actions:

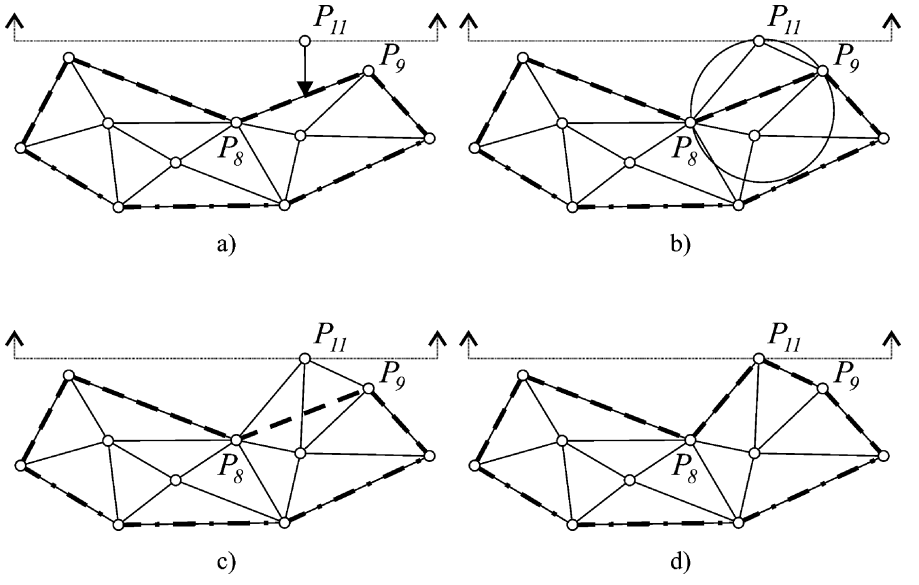

Figure 4. Sweep-line DT triangulation using an advancing front.

(i) a vertical projection of point P_{11} on the advancing front is determined (figure 4a);
(ii) the new triangle $\Delta P_9P_{11}P_8$ is formed (*see* figure 4b) and checked for empty circle property with its neighbors. If this property is violated, the process of legalization is executed (figure 4c);
(iii) the advancing front is updated (figure 4d).

Details of the algorithm, including special cases, are given in a study by Žalik (2005).

3.2 *CDT algorithm*

The proposed CDT algorithm consists of three parts:

(i) initialization;
(ii) sweeping;
(iii) finalization.

3.3 *Initialization*

First, all input points are sorted regarding y coordinate, regardless of whether they define an edge or not. Those points having the same y coordinates are also sorted in the x direction. Each point is associated with the information whether or not it is the upper ending point of one or more edges e_i. An example is shown in figure 5. Points P_5 and P_9 store the information about edges e_2 and e_5, correspondingly. Point P_7 is associated with the edges e_1, e_3 and e_4. If our edge is horizontal, the ending point with smaller x coordinate is considered as the upper point.

The first triangle is created after sorting and analyzing the vertices. Initialization, as proposed by Žalik (2005), distinguishes between six fundamentally different cases, and whether the cases when the first $m<n$ points, are vertically or horizontally collinear. This complicates the implementation and is the core of possible

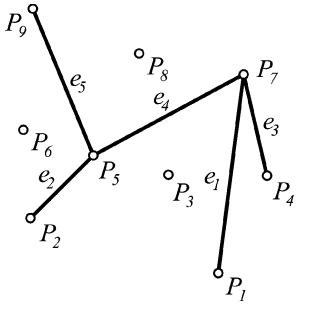

Figure 5. Associating edge information to the points.

programming errors. In the case of CDT, even more cases would need to be considered. For this reason, another approach is used, following the idea of so-called big or hyper triangles, as used by incremental insertion algorithms (Guibas *et al.* 1992). In our case, just two artificial points are needed, marked as P_{-1} and P_{-2}. They are located below all the points, and their x coordinates exceed the minimal and the maximal x values of the input points. Formally:

$$\mathrm{P}_{-1} = \left(x_{min} - \delta_x, y_{min} - \delta_y\right)$$

$$\mathrm{P}_{-2} = \left(x_{max} - \delta_x, y_{min} - \delta_y\right)$$

$$\delta_x = \alpha * \left(x_{max} - x_{min}\right)$$

$$\delta_y = \alpha * \left(y_{max} - y_{min}\right)$$

$$\alpha > 0$$

For x positions of the artificial points, it only matters that they are situated outside the input range. The constant α takes an arbitrary positive value (in our implementation $\alpha = 0.3$).

The remaining initialization tasks are now trivial. The first point $P_0 \in V$ forms the initial triangle with the artificial points P_{-1} and P_{-2}, as can be seen in figure 6. The

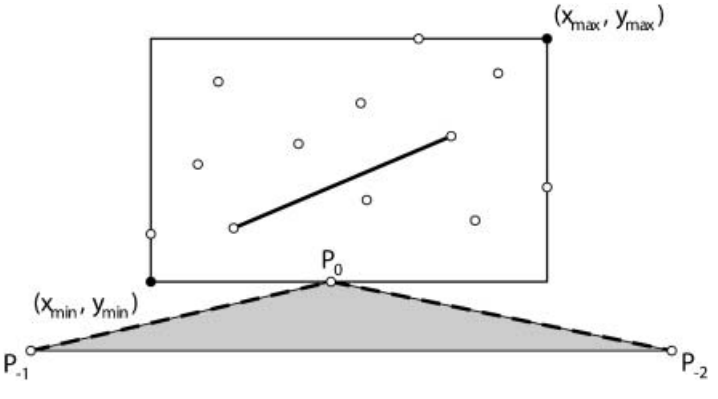

Figure 6. Initial triangle.

initial state of the advancing front is also initialized trivially. It consists of two line segments $P_{-1}P_0$ and P_0P_{-2}. The use of two artificial points also simplifies the sweeping phase, as will be mentioned later. However, during the finalization phase, all triangles, having at least one vertex among the artificial points, are erased.

3.4 *Sweeping*

Following the idea of the sweep-line paradigm, the algorithm stops at every point and checks its status. Two cases are possible:

(i) a point is isolated or it is the lower edge point (we name this case as a point event);
(ii) a point is the upper ending point of at least one line segment (this case is considered as an edge event).

Obviously, the point insertion is executed at each point regardless of its type, but the edge insertion is done only when the whole edge is swept.

3.4.1 Point event. When the sweep-line meets point P_i, its vertical projection P_Q to the advancing front is determined (figure 7a). The advancing front line segment P_LP_R containing P_Q is found (P_L and P_R refer to the left and right ending points of the advancing front line segment). The advancing front is implemented as a double-linked list, combined with a hash table to accelerate geometric search.

In the sweep-line DT algorithm sketched in section 3.1 (Žalik 2005), five possible cases appear for controlling the point event. However, because of the introduction of artificial points P_{-1} and P_{-2}, only two cases arise:

(i) middle case. Projection point P_Q lies strictly within an advancing front segment P_LP_R, $P_Q \neq \{P_L, P_R\}$, as shown in figure 7a. A new triangle $\Delta P_RP_iP_L$ is created (figure 7b), legalized and finally, the advancing front is updated (figure 7c);
(ii) left case. The projected point P_Q coincides with a point P_L (figure 8a). Two new triangles are generated, legalized and the advancing front is updated, as shown in figure 8b.

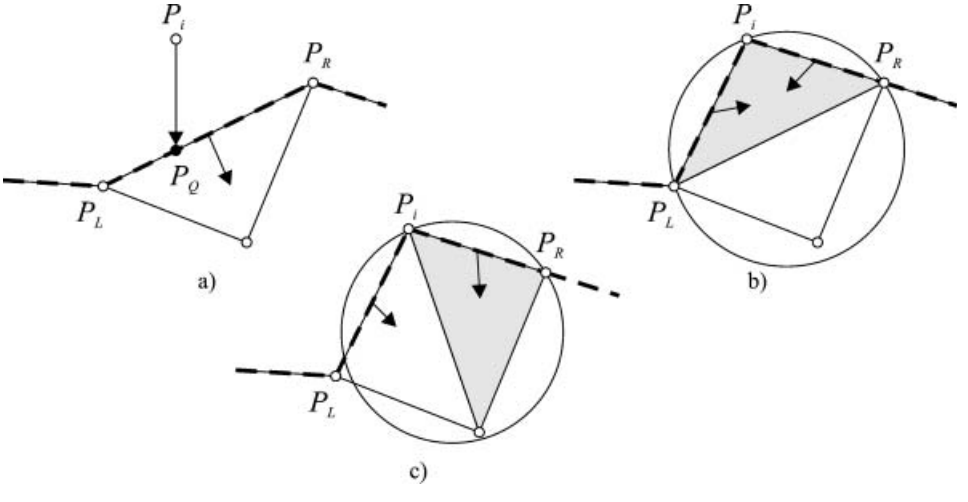

Figure 7. (a) Vertical projection of point P_i hits advancing front segment P_LP_R. (b) New triangle $\Delta P_RP_iP_L$ is created. (c) Advancing front is updated.

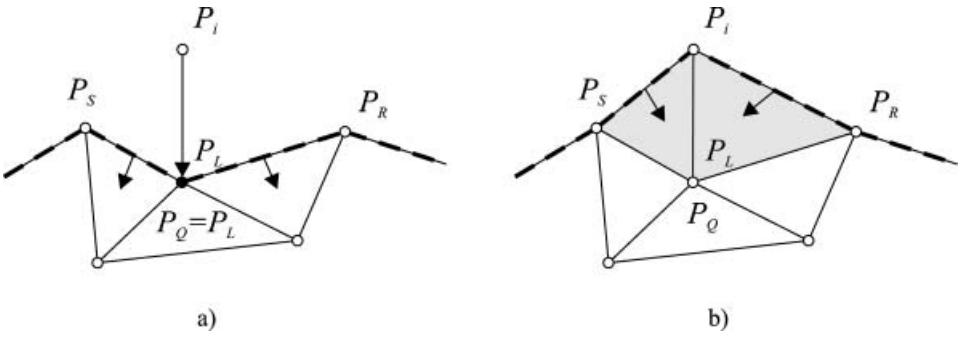

Figure 8. (a) Projected point P_Q coincides with P_L. (b) Two triangles $\Delta P_L P_i P_S$ and $\Delta P_L P_R P_i$ are created.

However, the process of forming triangles continues at this stage. Additional triangles are generated by connecting point P_i with the visible points on the advancing front. A direct application of this idea generates a lot of badly shaped triangles, which need to be legalized later by edge swapping. To avoid this, two heuristics are proposed:

(i) the angle between the new triangle's edge and the advancing front edge is checked. If the angle is smaller than $\pi/2$, a triangle is generated, otherwise the process of adding triangles is stopped (*see* figure 9);
(ii) the angles between the advancing front line segments are controlled to reduce the undulatation of the advancing front [in (Žalik 2005) such parts are named basins]. If the angle is larger than $3\pi/4$, the basin is filled by triangles (*see* figure 10).

Each new triangle is legalized instantly, so that Delaunay property is always assured.

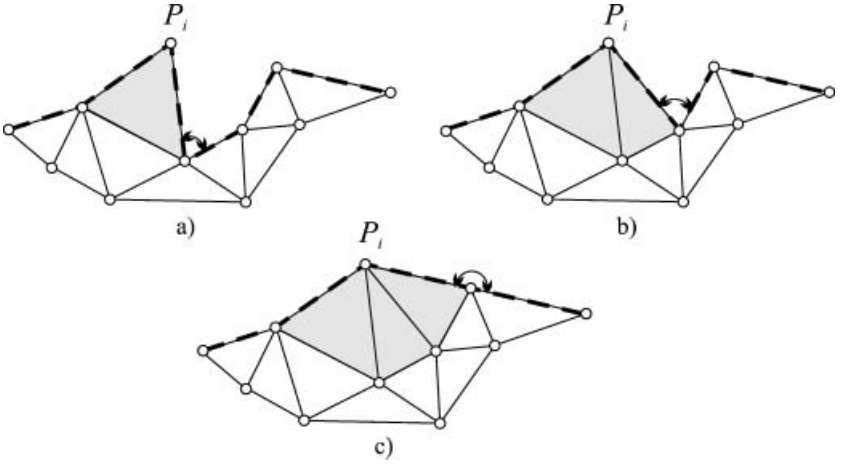

Figure 9. (a) The angle on the right side of the new triangle is smaller than $\pi/2$, (b) therefore, a new triangle is added. (c) Again, the angle to the right of the new triangle is checked, which is still larger than $\pi/2$. The process stops when no such angles exist.

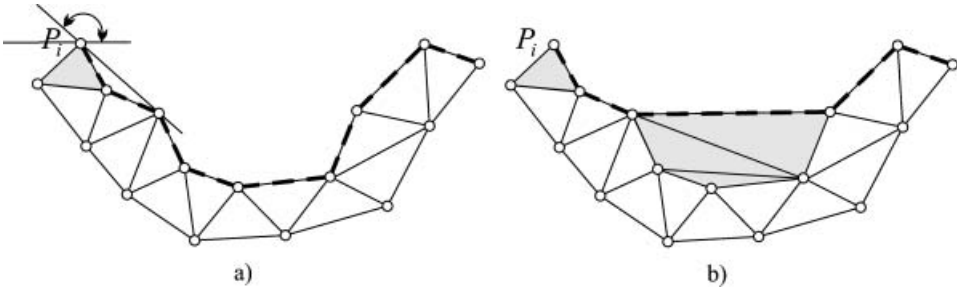

Figure 10. (a) After insertion of point P_i a basin is detected and (b) afterwards filled with triangles.

3.4.2 Edge event. The edge is inserted when the sweep-line meets its ending point (the coordinate with higher y coordinate). Actually, at first the ending point is included in CDT, as explained in the previous subsection. The edges associated with the upper ending point of the line segment are used one by one in arbitrary order. The algorithm works in three steps:

(i) locating the first intersected triangle;
(ii) removing intersected triangles;
(iii) triangulating empty areas. Location of the first triangle pierced by edge e is very simple. It is one of the triangles surrounding point P_i. This triangle is quickly found by using cross-product (*see* figure 11).

The second task, i.e. removing the remaining pierced triangles, is performed by traversing the triangulation through triangle adjacency links (referred to as triangle traversal). The idea is the same as explained in detail in a work by Anglada (1997). Roughly, in each visited triangle the algorithm selects the next triangle to proceed traversal, by determining its position regarding the constraining edge. Each traversed triangle is afterwards removed, and its vertices are stored in two separate lists above and below the edge e. In this way so-called upper Π_U and lower Π_L pseudo-polygons are obtained, which are finally triangulated (figure 12b).

In our algorithm, three cases have to be considered regarding the position of the edge e against the advancing front. The first case has already been discussed, where edge e is entirely below the advancing front. The easiest case appears when edge e does not intersect any triangles (figure 13). Here, only the lower pseudo-polygon Π_L exists and its vertices are determined by following the advancing front from point P_i to the ending point of the edge e (referred to as advancing front traversal).

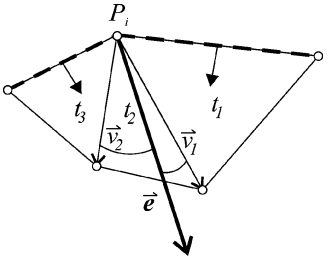

Figure 11. The search starts in triangle t_1, where the sign of vector product $v_1 \times \vec{e}$ is calculated. The sign changes with vector product $v_2 \times \vec{e}$ in triangle t_2, which is the first intersected triangle.

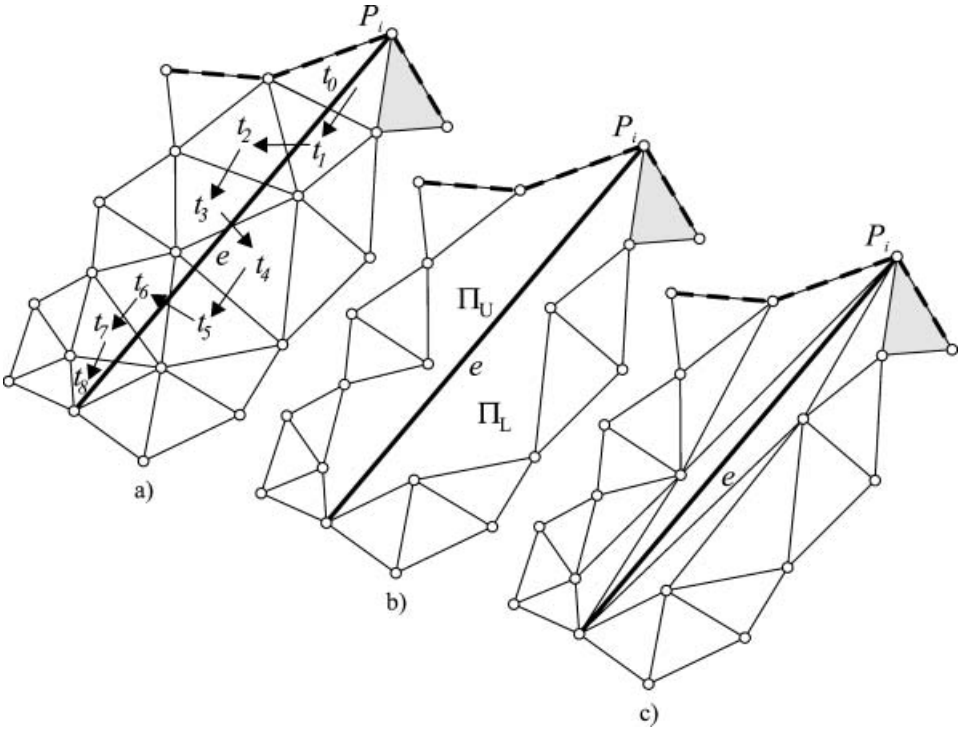

Figure 12. (a) Traversing through triangles t_0-t_8 (b) forms empty pseudo-polygons Π_U and Π_L (c) which are triangulated at the end.

The most difficult case occurs when edge *e* intersects the advancing front at one or more points (figure 14). However, even this case can be solved simply enough. The algorithm switches between the two types of traversal as follows: when traversing the advancing front, the side of the considered advancing front point is calculated regarding the constraining edge. If lying above the edge, traversal switches from the advancing front traversal to triangle traversal, and vice versa, and when the point is below the edge it is switched from triangle to advancing front traversal.

All three steps can be bypassed, if edge *e* completely coincides with one of the triangle's edges. Triangle edge is marked as fixed and cannot be swapped.

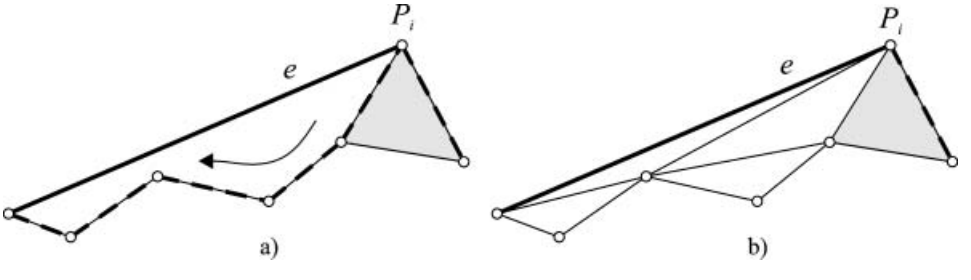

Figure 13. Edge insertion where no triangles are intersected.

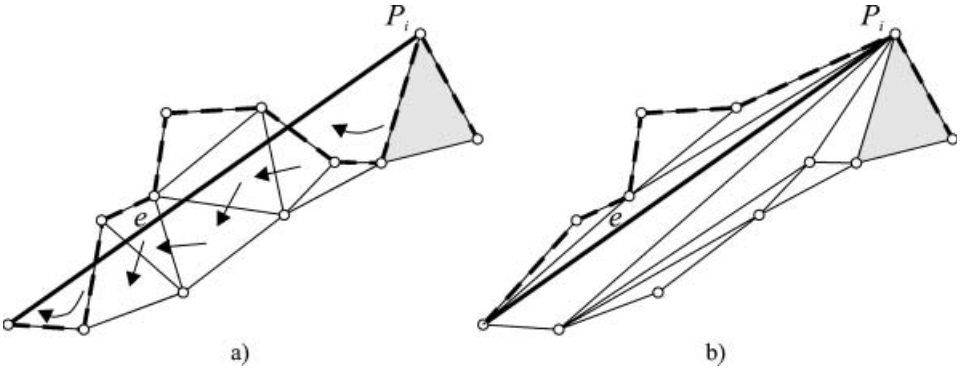

Figure 14. Edge insertion which combines AF and triangle traversal respectively.

3.5 *Finalization*

When all points and edges have been swept, the triangulation is still incomplete. Two further tasks remain:

(i) removing the triangles defined by at least one artificial point;
(ii) adding the bordering triangles (the edges of bordering triangles should form the convex hull of V).

Both tasks are solved simultaneously. The algorithm starts at the first advancing front point right of P_{-1}. Triples of consequent advancing front points are taken and the sign area of the obtained triangle is calculated. If positive, the points determine the missing triangle, which is generated and legalized (*see* figure 15a). When the second artificial point P_{-2} is reached, the algorithm is slightly modified, as we do not have an advancing front in this case (*see* figure 15b). The algorithm follows the artificial triangles from P_{-2} to P_{-1}. The considered artificial triangle is deleted and similar to before, the signed area for the consequent triple of points is calculated. In this case, if the signed area is negative, a new triangle is added and legalized (figure 15c).

4. Results

Algorithms are usually analyzed regarding time and space complexity. The time complexity depends on the characteristics of the data. The asymptotic worse case and the expected time analysis is, therefore, derived. However, the users are more

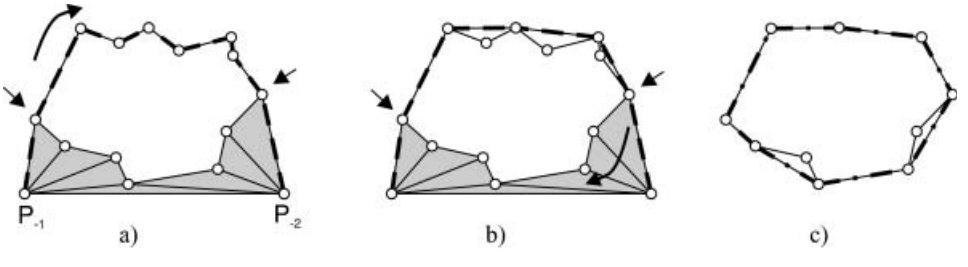

Figure 15. Finalization forms upper and lower convex hull.

interested in actual run-time on data from practice (two of our tested examples are shown in figure 16).

We are aware that this comparison also depends on many factors such as programming language, programming experience, style of programming, operating system, and the architecture of the computer used but, despite this, such experiments give valuable information to the reader about the efficiency of any algorithm.

Our algorithm (denoted as SL) has been compared with three known algorithms, implemented by Shewchuk in package Triangle (Shewchuk 1999a). His implementations are frequently used as reference implementations, owing to their stability and efficiency (Bilgili *et al.* 2006, Zhang and Smirnov 2005). In his package, three different algorithms can be found:

(i) divide-and-conquer algorithm by Lee and Schachter (1980) (denoted as LS-D&C);
(ii) sweep-line algorithm by Fortune (1987) (denoted as F-SL);
(iii) randomized incremental algorithm by Mücke *et al.*(1996) (denoted as M-INC).

All these three algorithms construct DT at first and then insert edges in triangulation, as explained in the introduction.

All measurements were carried out on an Intel Pentium M 2.0 GHz processor and 1 GB RAM running under Windows XP operating system. All algorithms were implemented in C and compiled by a VisualC compiler.

We provided the correct data input for all algorithms (the authors used parameters -pcBPNEIOX for Shewchuk's algorithms, which state that we triangulate a PSLG that must be enclosed with the convex hull. There are no holes in the input graph and no exact arithmetic is used), and the checking of the data was

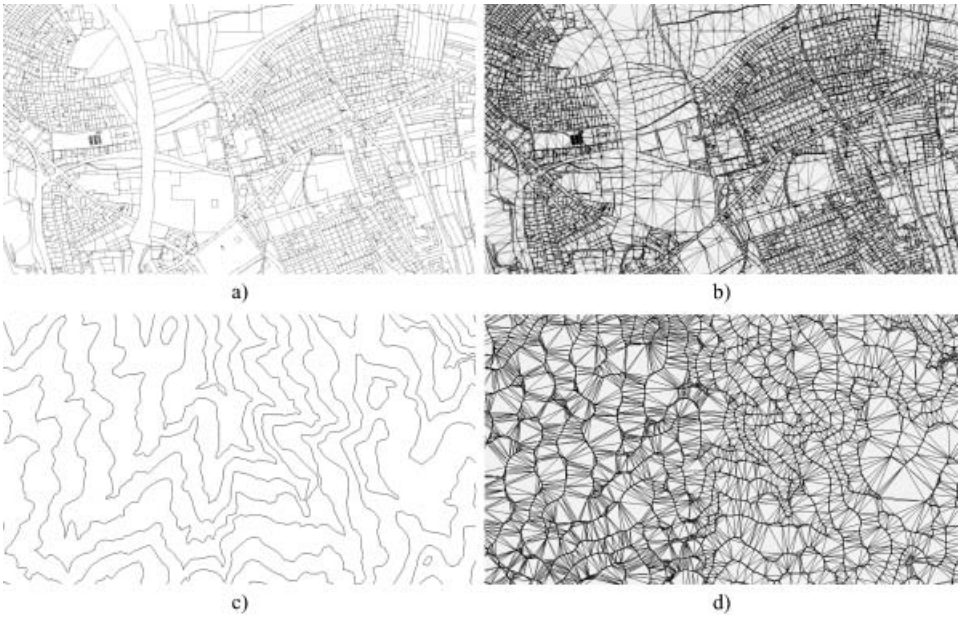

Figure 16. Examples of real input data from (a) land cadastre and (c) iso-lines with (b, d) resulting CDTs.

Table 1. Measured CPU times for tested data.

Example	Points	Edges	CPU time (s)			
			M-INC	F-S	LS-D&C	SL
a	23262	28155	0.1624	0.1030	0.0690	0.0658
b	93048	112620	0.7996	0.4656	0.3376	0.3227
c	162,244	162,312	1.2844	0.7848	0.5124	0.5313
d	372,192	450,480	4.5940	2.0312	1.6185	1.2397
e	1,297,360	1,296,020	21.1436	8.7032	4.3500	4.8907
f	1,488,768	1,801,920	26.5833	9.6303	7.7188	4.7813
g	2,444,700	2,933,200	56.5155	29.2453	22.8850	8.7660
h	5,189,440	5,184,080	/	64.9476	19.9814	20.4637
i	5,955,072	7,207,680	/	119.7030	45.0468	23.4323

disabled in the Shewchuk's code. Data checking is time consuming and unnecessary because the input is known to be correct. We have performed experiments on a number of real data sets varying in the number of input points and edges.

The results of the experiments are collected in table 1. As can be seen from the results, the M-INC algorithm is very slow. Our algorithm is considerably faster than the F-S algorithm in all cases, and is faster than the LS-D&C algorithm in the majority of cases.

Table 2 shows the processing times for individual parts of the algorithm. It can be seen that initialization takes more than half of the total time, while finalization is a negligible cost. During initialization, the input points have to be sorted and the constraining edges are assigned to the individual points. This task, however, takes the majority of CPU time.

5. Conclusion

This paper presents a new algorithm for constructing a constrained Delaunay triangulation. The proposed algorithm achieves good performance by an effective handling of constraining edges, using sweep-line paradigm. The algorithm exploits characteristics of the advancing front approach, not only for the quick addition of new triangles, but also for fast triangle location during edge insertions. For this reason, no other searching data structures are required, which results in a small memory requirement. The main credit for this algorithm's efficiency is down to heuristics for the considerable reduction in triangle swaps and to the routines for preliminary detection of simple edge insertion cases. This algorithm is easy to understand and relatively easy to implement and, as such, represents an interesting alternative in practice.

Table 2. Measured steps of SL algorithm.

Example	Points	Edges	CPU time (s)		
			Initialization	Sweeping	Finalization
b	93,048	112,620	0.1093	0.2033	0.0000
f	1,488,768	1,801,920	1.6406	3.1976	0.0000
g	2,444,700	2,933,200	3.7343	5.8853	0.0053

Acknowledgements

We would like to thank Dr Ivana Kolingerova and Dr Josef Kohout from West Bohemia University in the Czech Republic for their help and motivation when preparing this algorithm. We would also like to thank the Slovenian Research Agency for supporting this research under project P2-0041.

References

ANGLADA, M.V., 1997, An improved incremental algorithm for constructing restricted Delaunay triangulation. *Computers and Graphics*, **21**(2), pp. 215–223.

DE BERG, M., VAN KREVELD, M., OVERMARS, M. and SCHWARZKOPF, O., 1997, *Computational Geometry, Algorithms and Applications* (Berlin: Springer-Verlag).

BILGILI, A., SMITH, K.W. and LYNCH, D.R., 2006, BatTri: a two-dimensional bathymetry-based unstructured triangular grid generator for finite element circulation modeling. *Computers and GeoSciences*, **32**(5), pp. 632–642.

CHEW, L.P., 1989, Constrained Delaunay triangulations. *Algorithmica*, **4**(1), pp. 97–108.

DELAUNAY, B., 1934, Delaunay triangulation. Available online at: http://en.wikipedia.org/wiki/Delaunay_triangulation (accessed 25 May 2007).

EDELSBRUNNER, H. and SEIDEL, R., 1986, Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, **1**(1), pp. 25–44.

FORTUNE, S.J., 1987, A sweepline algorithm for Voronoi diagrams. *Algorithmica*, **2**, pp. 153–174.

GUIBAS, L., KNUTH, D. and SHARIR, M., 1992, Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, **7**, pp. 381–413.

KAUČIČ, B. and ŽALIK, B., 2004, K-guarding of polyhedral terrain. *International Journal of Geographical Information Science*, **18**(7), pp. 709–718.

LAWSON, C.L., 1977, Software for C 1 surface interpolation. In *Mathematical Software III*, J.R. Rice (Ed.), pp. 161–194 (New York, NY: Academic Press).

LEE, D.T. and SCHACHTER, B.J., 1980, Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, **9**(3), pp. 219–242.

MÜCKE, E.P., SIAS, I. and ZHU, B., 1996, Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulatons. In *Proceedings of the 12th annual symposium on computational geometry*, pp. 274–283 (Philadelphia, PA: ACM Press).

SAUX, E., THIBAUD, R., KI-JOUNE, L. and MIN-HWAN, K., 2004, A new approach for a topographic feature-based characterization of digital elevation data. In *Proceedings of 12th International Symposium of ACM GIS*, pp. 73–81 (Washington, DC: ACM Press).

SHEWCHUK, J.R., 1999a, Triangle: a two-dimensional quality mesh generator. Available online at: http://www.cs.cmu.edu/quake/triangle.html (accessed 22 July 2007).

SHEWCHUK, J.R., 1999b, Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 16th annual symposium on Computational geometry*, pp. 350–359 (Hong Kong: ACM Press).

SPAROVEK, G. and SCHNUG, E., 2001, Temporal erosion-induced soil degradation and yield loss. *Soil Science Society of America Journal*, **65**(5), pp. 1479–1486.

TSE, R.O.C., GOLD, C.M. and KIDNER, D., 2004, An original way of building a TIN with complex urban structures. In *Proceedings of ISPRS 2004 – XXth Congress* (Istanbul, Turkey).

ZHANG, H.Z. and SMIRNOV, A.V., 2005, Node placement for triangular mesh generation by Monte Carlo simulation. *International Journal for Numerical Methods in Engineering*, **2**(7), pp. 973–989.

ŽALIK, B., 2005, An efficient sweep-line Delaunay triangulation algorithm. *Computer-Aided Design*, **37**(10), pp. 1027–1038.