Computing Directional Constrained Delaunay Triangulations

Marc Vigo Anglada

Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Abstract

This work presents two generalizations of the algorithm for obtaining a constrained Delaunay triangulation of a general planar graph presented in [Vig97] and [Vig95]. While the first generalization works with elliptical distances, the second one can deal with a set of deforming ellipses associated to each point of the plane. The pseudo-code of the procedures involved in the algorithms is included, the suitability of the algorithms is analyzed, and several examples are presented,

Keywords: Constrained Delaunay triangulations, elliptical distances

Computing Directional Constrained Delaunay Triangulations

Marc Vigo Anglada

Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

1 Introduction

The aim of this paper is the study and development of simple algorithms that can obtain a modified Delaunay triangulation of a general graph. The modification consists on replacing the usual distance criterion (the Euclidean distance in the plane) by a new criterion which include directionality. This problem emerged while developing a triangulation algorithm for surfaces that is adaptive to the curvature directions of the surface; in other words, the algorithm must create triangles whose shape depends on the curvature of the surface in each direction. The ideal case would be a triangulation such that its edges are aligned according to curvature directions, and have length proportional to curvature (long edges in plane regions, short edges in curved regions). That is the reason why the new algorithm is called the directional CDT algorithm.

Other authors have used the notion of a Delaunay triangulation (DT) generalized for non usual (Euclidean) distances: an algorithm for obtaining a DT of a set of points using a convex distance is presented in [CD86]; a generalization of the Voronoi graph and its dual, the Delaunay triangulation, for affine invariant norms is introduced in [NF89]; and [OBS92] describes several generalizations of the Voronoi graph for weighted distances. Two approaches that share some similarities with our are presented in [Mav90] and [Pos93].

In [Mav90], a method is developed for obtaining an adaptive mesh of a viscous flow taking into account a deformation directional function which assigns a vector (direction and norm) to each point of the plane. This proposal is similar to the one presented here, because it is based on the modification of a DT algorithm in such a way that it includes directional information, locally deforming the original space. The author affirms that the resulting method can be viewed as a process for constructing a DT where the circumcircles in deformed space correspond to ellipses in the original space. The proposal of [Pos93] consists on generalizing the idea of circumcircle by any convex "circum-shape'. This results in a DT algorithm that could be used by fluid dynamics FEM applications. The input of the algorithms includes a function that maps a convex circum-shape to each point of the plane. However, the author does not describe the algorithm used for obtaining the triangulation. Notice that these two proposals do not accept constraining edges (*i.e.* the Delaunay triangulation is not constrained).

The solution proposed here is a generalization of the incremental algorithm presented in [Vig97] and [Vig95]), which obtains a constrained Delaunay triangulation (CDT) of a general graph. This algorithm has the advantage that it admits constraining edges, and that it is incremental, therefore new points and vertices can be included in a triangulation making only local rearrangements of the structure, instead of having to recompute the whole triangulation. Furthermore, the resulting algorithm is very simple to implement, robust and efficient, even when the original graph has many vertices or edges [Vig97].

The rest of the paper is organized as follows. Next section is a review of the procedures involved in the incremental algorithm of [Vig97] and [Vig95]. Section 3 defines the problem that is going to be solved and presents two generalizations that can deal with elliptical distances for obtaining the triangulation of a general graph. Some examples are presented in Section 4, analyzing the results and showing the behavior of the new algorithms. Finally, conclusions are presented in the last section.

2 The incremental CDT algorithm

This section is dedicated to the review of the incremental algorithm presented in [Vig97]. As explained, the algorithm works incrementally, and given a planar graph $\mathcal{G} = (\mathcal{A}, \mathcal{V})$, constructs a Delaunay triangulation of $\mathcal{V}$ constrained by the edges of $\mathcal{A}$. It is based on two procedures, one for adding a new point in the interior of a CDT and another one that enforces a constraining edge between two points of the triangulation. These two procedures are represented in Figure 1.

Procedure `AddPointCDT` given the CDT of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ and a vertex $p \notin \mathcal{V}$, results in the CDT of the augmented graph $(\mathcal{V} \cup \{p\}, \mathcal{A})$. The procedure is the following:

Procedure *AddPointCDT*($\mathcal{T}: CDT,\ p: Vertex$)
  {Prec: $p \notin \mathcal{T} \wedge p \in Interior(\mathcal{T})$}
  stack := *EmptyStack*()
  Find the triangle $t \in \mathcal{T}$ *that contains* p
  Divide t *in three triangles,* t_1, t_2 *and* t_3*, depending on* p

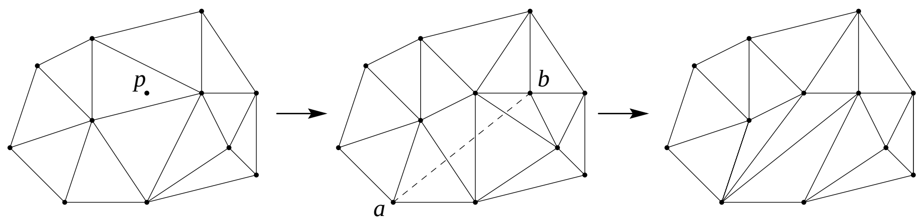

Figure 1: Representation of procedures that insert a point and an edge in a CDT. *Left*, point p is added to the original triangulation; *right*, edge ab is enforced, obtaining the final CDT.

    Push$(stack, t_1)$
    Push$(stack, t_2)$
    Push$(stack, t_3)$
    While *IsNotEmpty*$(stack)$ **do**
        $t :=$ *Pop*$(stack)$
        $t_{opo} :=$ *OpposedTriangle*(t, p)
        If *edge shared by* t *and* t_{opo} *is not fixed*
                and *PointInCircumcircle*(p, t_{opo}) **then**
            Swap the edge shared by t *and* t_{opo}
            Push$(stack, t)$
            Push$(stack, t_{opo})$
        endIf
    endWhile
endProcedure

The second procedure, `AddEdgeCDT`, given a CDT of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ and an edge $ab \notin \mathcal{A}$, $a, b \in \mathcal{V}$, results in the CDT of the augmented graph $(\mathcal{V}, \mathcal{A} \cup \{ab\})$:

Procedure *AddEdgeCDT*$(\mathcal{T} : CDT,\ ab : Edge)$
    {Prec: $a\}, b \in \mathcal{T} \wedge ab \notin \mathcal{T}$
    Find the triangle $t \in \mathcal{T}$ *that contains* a *and is cut by* ab
    $P_U :=$ *EmptyList*()
    $P_L :=$ *EmptyList*()
    $v := a$
    While $b \notin t$ **do**
        $t_{seg} :=$ *OpposedTriangle*(t, v)
        $v_{seg} :=$ *OpposedVertex*(t_{seg}, t)
        If v_{seg} *is above the edge* ab **then**
            AddList(P_U, v_{seg})
            $v :=$ *Vertex shared by* t *and* t_{seg} *above* ab
        else

      AddList(P_L, v_{seg})
      $v :=$ *Vertex shared by* t *and* t_{seg} *below* ab
    endIf
    Remove t *from* $\mathcal{T}$
    $t := t_{seg}$
  endWhile
  CDTPseudopolygon$(P_U, ab, \mathcal{T})$
  CDTPseudopolygon$(P_L, ab, \mathcal{T})$
  Reconstitute the triangle adjacencies of $\mathcal{T}$
  Add edge ab *to* $\mathcal{T}$
  Mark the edge ab *from* $\mathcal{T}$ *as fixed*
endProcedure

This procedure finds and removes triangles of the CDT cut by the edge ab that is being inserted; after that, edge ab is added in the CDT, leading two un-triangulated regions on both sides of the edge, the upper and lower pseudo-polygons. Procedure `CDTPseudopolygon` is a recursive method that retriangulates the interior of these two regions:

Procedure *CDTPseudopolygon*$(P : VertexList,\ ab : Edge,\ \mathcal{T} : CDT)$
  If P *has more than one element* **then**
    $c :=$ *First vertex of* P
    For *each vertex* $v \in P$ **do**
      If *PointInCircumcircle*$(v,$ *Triangle*$(a, b, c))$ **then**
        $c := v$
      endIf
    endFor
    Split P *into* P_E *and* P_D, *giving* $P = P_E + \{c\} + P_D$
    $CDTPseudopolygon(P_E, ac, \mathcal{T})$
    $CDTPseudopolygon(P_D, cb, \mathcal{T})$
  endIf
  If P *is not empty* **then**
    Add triangle with vertices a, b, c *into* $\mathcal{T}$
  endIf
endProcedure

The data structure representing a triangulation that these algorithms manage must provide the following methods:

- Obtaining all the triangles (sequential access is enough).
- The three vertices that form a given triangle.
- The three neighboring triangles to a one given.

- Knowing if an edge shared between two triangles is a constraining one.

Provided with these functions, all the required topological relationships are trivial to implement, such as `OpposedTriangle`, which returns the neighboring triangle opposing to one of its vertices; or `Opposedvertex`, which given two neighboring triangles, returns the vertex of the first one opposed to the second.

Notice that the function that guarantees the resulting triangulation is a CDT are those related with the empty circumcircle criterion, that is, the `PointInCircumcircle` test. The generalizations that we are gong to obtain will only have to adapt this test for elliptical distances. Remark that there are some conditions for the algorithms to work correctly: first, the resulting triangulation must remain a planar graph (crossing edges are not allowed), and second, the algorithm must respect the visibility conditions.

3 The directional CDT algorithm

The incremental algorithm for obtaining a CDT of a general graph is based on two procedures, one for adding a new vertex on a the triangulation and a second one that forces a constraining edge between two existing vertices of the triangulation (see [Vig97]). The geometric checks realized by both procedures to assure the Delaunay empty circumcircle criterion are included in a procedure (`PointInCircumcircle`) that checks if a point lies inside the circumcircle of a triangle. Therefore, only this procedure has to be modified in order that the generalized algorithm works with elliptical distances and circumellipses.

The easiest way to adapt this test is to transform the points received as input parameters according to the transformation map induced by the ellipse, and then to apply the usual test that checks if a point lies inside the circumcircle of a triangle. Therefore, we have changed the calls to procedure `PointInCircumcircle` by calls to `PointInCircumellipse`, which receives an additional parameter, the transforming ellipse. The question is which ellipse has to be chosen in each call. Notice that, as the transformation induced by an ellipse will be a linear map, the topological changes made by the algorithm (edge swaps, etc) will always be correct, because a linear map does not affect convexity of entities and visibility conditions between points. The method that checks if a point lies inside the circumcircle of a triangle is the following:

Function *PointInCircumellipse*($p : Vertex,\ t : Triangle,\ e : Ellipse$)

returns *Boolean*

$x_1 := \mathit{GetVertexTriangle}(t, 1)$
$x_1 := \mathit{GetVertexTriangle}(t, 2)$
$x_1 := \mathit{GetVertexTriangle}(t, 3)$
$xt_1 := \mathit{TransformPointEllipse}(x_1, e)$
$xt_2 := \mathit{TransformPointEllipse}(x_2, e)$
$xt_3 := \mathit{TransformPointEllipse}(x_3, e)$
$pt := \mathit{TransformPointEllipse}(p, e)$
return $\mathit{PointInCircumcircle}(pt, Triangle(xt_1, xt_2, xt_3))$
endFunction

The call to `TransformPointEllipse` maps a point from the original plane to the deformed plane, according to the transformation induced by an ellipse. All through this paper the original plane will be refered as $\mathbb{D}$, and $\mathbb{D}_E$ will be the deformed plane, since the ellipse that deforms the plane will be E.

The first step of the algorithm for obtaining a CDT of a general graph consists on scaling the vertices of the graph in order they are included inside a triangle (called the super-triangle, see [Vig97]); a precondition of the procedure that adds a new point in a CDT is that the point has to be interior to the current triangulation. Therefore, a good practice is to store the original coordinates of the vertices. Furthermore, one must work with special care with the vertices of the super-triangle, where the transformation function may not be defined.

Two generalizations of the algorithm are presented: in the first case, the deforming ellipse is unique, and in the second case, E will depend on the point of the plane. The resulting triangulations have the property that the alignment and length of output edges depends on the deforming ellipses.

Before, let us define the elliptical distance and the transformation induced by an ellipse.

Definition 1 (Elliptical distance). Given an ellipse E with associated matrix M_E, the elliptical distance between two points $p, q \in \mathbb{D}$ is

$$\|pq\|_E = [(p-q)^T M_E (p-q)]^{\frac{1}{2}}$$

Transformation induced by an ellipse

Let M_E be the matrix associated to a given ellipse E, let λ_1 and λ_2 be the eigenvalues of the matrix, and let $v_1 = (e_1, e_2)$ and $v_2 = (e_2, -e_1)$ be its the eigenvectors. Then, the boundary points of the ellipse are $x \in \mathbb{D}$ such that

$$x^T M_E x = 1$$

and

$$M_E = \begin{pmatrix} e_1 & e_2 \\ e_2 & -e_1 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} e_1 & e_2 \\ e_2 & -e_1 \end{pmatrix} = \begin{pmatrix} \lambda_1 e_1^2 + \lambda_2 e_2^2 & (\lambda_1 - \lambda_2) e_1 e_2 \\ (\lambda_1 - \lambda_2) e_1 e_2 & \lambda_1 e_2^2 + \lambda_2 e_1^2 \end{pmatrix}$$

We are looking for a deforming function, $f_E : \mathbb{D} \longrightarrow \mathbb{D}_E$, such that the image of E is a circle in $\mathbb{D}_E$. Consequently, f_E is a linear map defined by

$$f_E(x) = Mx$$

with

$$M = \begin{pmatrix} e_1 & e_2 \\ e_2 & -e_1 \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{pmatrix} \begin{pmatrix} e_1 & e_2 \\ e_2 & -e_1 \end{pmatrix} =$$

$$= \begin{pmatrix} \sqrt{\lambda_1} e_1^2 + \sqrt{\lambda_2} e_2^2 & (\sqrt{\lambda_1} - \sqrt{\lambda_2}) e_1 e_2 \\ (\sqrt{\lambda_1} - \sqrt{\lambda_2}) e_1 e_2 & \sqrt{\lambda_1} e_2^2 + \sqrt{\lambda_2} e_1^2 \end{pmatrix}$$

The linear map f_E will be called the **transformation function induced by ellipse** E.

Notice that an alternative could have been to define f_E as a simple rotation and an scaling, *i.e.*, matrix

$$M' = \begin{pmatrix} e_1 & e_2 \\ e_2 & -e_1 \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{pmatrix},$$

also fulfills our requirements.

3.1 Case I: The deforming ellipse is unique

In this case, the incremental method for obtaining a CDT works in deformed space, $\mathbb{D}_E$, that is, it measures elliptical distances. Therefore, the new algorithm can map the graph vertex coordinates applying the transformation function f_E, apply the incremental method for obtaining the CDT of the transformed graph, and finally map back the vertices in $\mathbb{D}$.

The triangles of the output triangulation will fulfill the so-called **empty circumellipse criterion**.

Definition 2 (Circumellipse of a triangle). Let E be an ellipse with associated matrix M_E, and let T be a triangle, the *circumellipse* of T is an ellipse E' sharing the same axes (eigenvectors) than E and that has the tree vertices of the triangle in its boundary. That is, E' is the ellipse with associated matrix $M_E{}'$

$$M_E{}' = r M_E \quad \text{with} \quad r \in \mathbb{R}^+$$

with the vertices x_1, x_2, x_3 of T in its boundary:

$$x_i^T M_E{}' x_i = 1, \quad i \in \{1, 2, 3\}$$

In other words, th elliptica distance form the vertices x_i to the center of the ellipse E is the same for all $i \in \{1, 2, 3\}$.

Proposition 1. *The circumellipse of any triangle of a CDT obtained applying the incremental method adapted for elliptical distances does not contain any vertex of the CDT visible from the triangle.*

The proof of this proposition is trivial, taking into account that function f_E maps the ellipse of $\mathbb{D}$ into a circle of $\mathbb{D}_E$.

3.2 Case II: There exists a deforming ellipse for every point on the plane

In this case, each point of the plane has an ellipse associated, that is, function f_E depends on the point of the plane. The generalization that we propose takes advantage of the fact that the algorithm works incrementally: the ellipse in each call to the procedures for inserting a point or an edge to the triangulation will be different.

When a new point is added in the triangulation, the ellipse that determines the deformation of the space is the one associated to the point. This is one of the differences with respect to the method proposed in [Mav90], where the deformation applied is an average of the deformations associated to the neighboring points to the one inserted. Our choice allows ensuring Proposition 2 on the resulting CDT (see below). Remark that, thanks to the fact that the algorithm works incrementally, the deformation determined by the ellipse must only be applied to a reduced set of points (the neighboring points to the one inserted). This results in the following directional algorithms for inclusion of a new point in a CDT:

Procedure *AddPointCDTE*($\mathcal{T} : CDT,\ p : Vertex$)
  {Prec: $p \notin \mathcal{T} \wedge p \in Interior(\mathcal{T})$}
  $stack :=$ *EmptyStack*()
  Find the triangle $t \in \mathcal{T}$ *that contains* p
  Divide t *in three triangles,* t_1, t_2 *and* t_3*, depending on* p
  $ell :=$ *DeformingEllipse*(p)
  Push($stack, t_1$)
  Push($stack, t_2$)
  Push($stack, t_3$)

```
  While IsNotEmpty(stack) do
    t := Pop(stack)
    t_opo := OpposedTriangle(t, p)
    If edge shared by t and t_opo is not fixed
        and PointInCircumcellipse(p, t_opo, ell) then
      Swap the edge shared by t and t_opo
      Push(stack, t)
      Push(stack, t_opo)
    endIf
  endWhile
endProcedure
```

Function `DeformingEllipse` is a routine that given a point of $\mathbb{D}$ returns the ellipse associated to it. This method should be supplied by the user of the directional CDT algorithm.

During the insertion of a new constraining edge into the triangulation, the deforming ellipse is not unique. Notice that the empty circumcircle criterion in this case is only used during the triangulation of the upper and lower pseudo-polygons (see [Vig97]). Therefore, to adapt the algorithm it is only necessary to have a different deforming ellipse in each step of the recursion for choosing the point that splits the pseudo-polygon (point c of the algorithm, see below); the chosen ellipse is the one associated to one of the endpoints of the pseudo-polygon, *i.e.* the endpoints of the edge for which the polygon is edge-visible. This choice allows us to assure Proposition 2 (see below): notice that in this way the circumellipse of the triangles formed in each recursive step is empty. Remark that the method proposed in [Mav90] does not allow constrained edges; the author states that the resulting triangulation will always contain the boundary edges, thanks to the configuration of the input points.

Since the procedure that enforces an edge between two existing points of a CDT does not perform any test related to the empty circumcircle criterion, it remains the same. Only the subroutine that retriangulates the upper and lower pseudo-polygons has to be modified:

```
Procedure CDTEPseudopolygon(P : VertexList, ab : Edge, T : CDT)
  If P has more than one element then
    c := First vertex of P
    ell := DeformingEllipse(a)
    For each vertex v ∈ P do
      If PointInCircumEllipse(v, Triangle(a, b, c), ell) then
        c := v
      endIf
```

endFor
Split P *into* P_E *and* P_D, *giving* $P = P_E + \{c\} + P_D$
$CDTEPseudopolygon(P_E, ac, \mathcal{T})$
$CDTEPseudopolygon(P_D, cb, \mathcal{T})$
endIf
If P *is not empty* **then**
Add triangle with vertices a, b, c *into*$\mathcal{T}$
endIf
endProcedure

Notice that in this case the deforming ellipse could have been the one corresponding to the other endpoint of the edge, *i.e.*, the ellipse associated to point b.

It is easy to show that this generalized algorithm correctly: the procedures that add a point or an edge into the triangulation converge in a finite number of steps, because they work as in the previous case, when the ellipse was the same for all the points. However, notice that in this case the notion of elliptical distance is no longer valid, and the empty circumellipse criterion does not provide a unique triangulation [CD86]. Therefore, the output triangulation when applying the proposed method depends on the order the points are added.

The resulting triangulation will fulfill the following property, called the **almost empty circumellipse criterion**:

Proposition 2. *Given a triangle T of a triangulation obtained applying the incremental method adapted for the case where the ellipse depends on the plane point, there exists a vertex v of T such that the circumellipse of T with respect to v does not contain any other point of the triangulation visible from the triangle that has been inserted before v.*

This proposition follows straightforward from the way the given algorithm works: when a new point or edge is added, the space is deformed with respect to the ellipse associated to the point being added or the endpoints of the pseudo-polygon formed when the constraining edge is forced.

4 Examples

This section shows some example of application of the modified algorithms to a general graph, showing the announced properties about the resulting triangulations.

First example consists on a graph composed of a concave polygon with a hole and a set of interior points. Figure 2 shows an example of the application of the modified algorithm to this graph. It is clear that the output triangles have their edges according to the shape of the deforming ellipse E, long edges are produced in the direction of the largest axe of the ellipse. Compare the results of the usual CDT with the triangulation obtained when applying the directional algorithm.

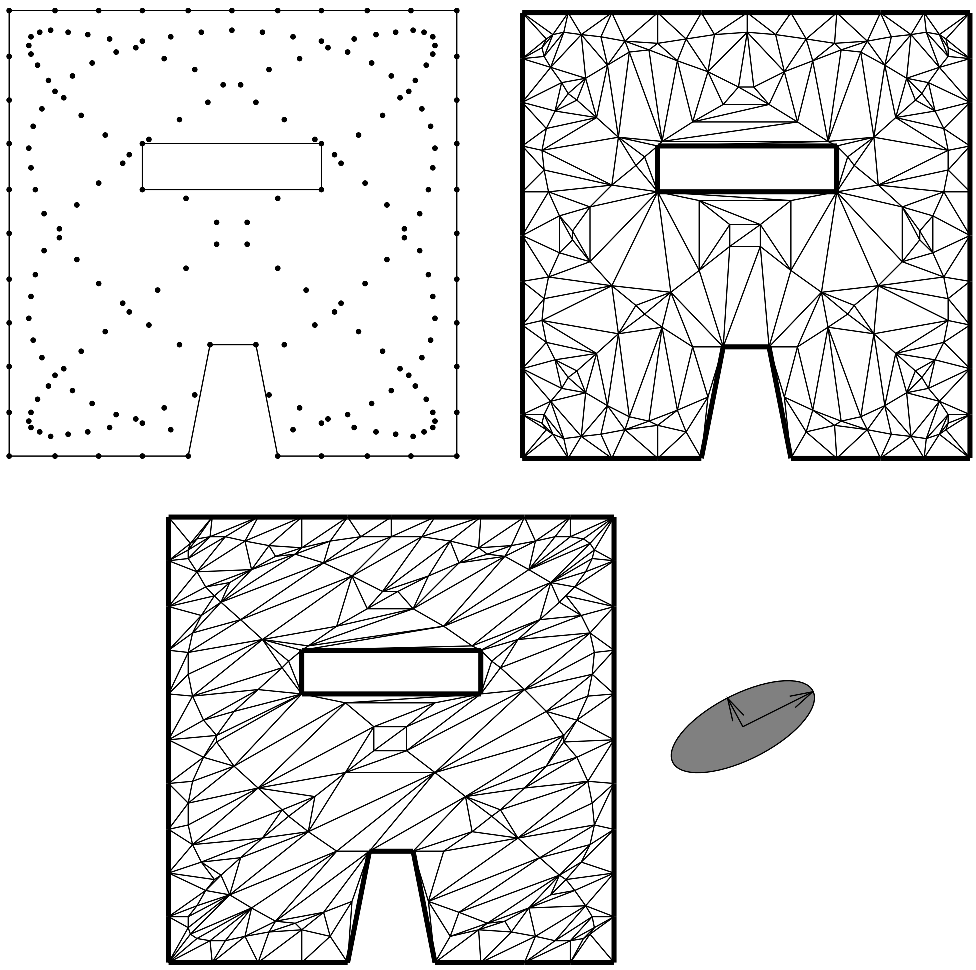

Figure 2: Example of application of the algorithm for the case that the ellipse E is unique. *Up*, original graph and usual CDT; *bottom*, output triangulation of the modified algorithm and deforming ellipse used.

Figure 3 shows an example where the modified algorithm has been applied to the same graph than in Figure 2, but working with a function f_E that provides a different ellipse for each point on the plane $u \times v$. As it can bee seen, while the aspect ratio of the ellipses depends on the horizontal direction (u), and the direction of their axes depends on the vertical direction (v). As expected, the resulting triangulation shows clearly that the shape of output edges is directly related to the direction of the deforming ellipses.

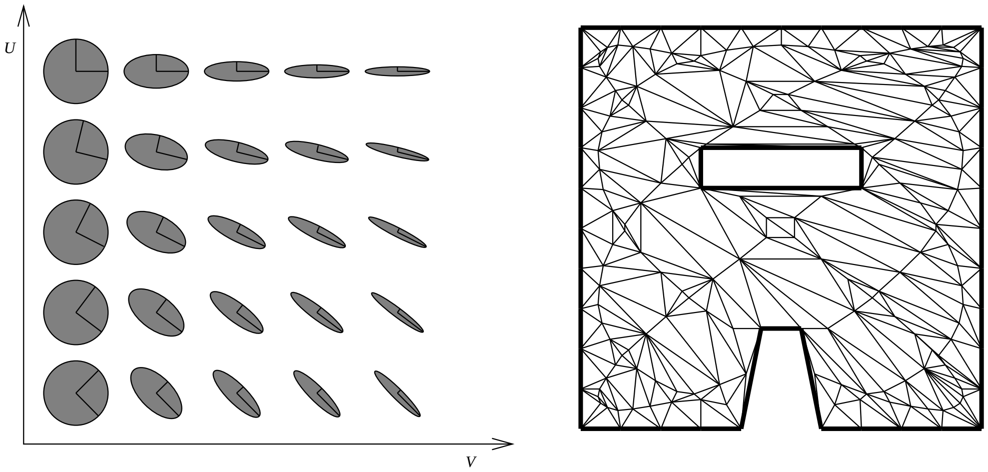

Figure 3: Example of application of the algorithm in the case the ellipse depends on the point of the plane. The figure on the left shows the ellipses evaluated at several points of the plane; the resulting triangulation is drawn on the right.

Second example (Figure 4) is a simple test case that shows one of the announced properties of the modified CDT algorithm in case II, when the deforming ellipse is not unique for each point of the plane: the resulting triangulation depends on the order of the points are inserted. The input graph in this case is a set of fifteen points, fourteen on the vertices of a regular polygon and the last one placed on its center. In this case, as there are not constraining edges, the output of the program is the convex hull of the point set. The deforming ellipses of this examples have the same configuration that in next example, that is, the ellipses point to the center of the graph, and their aspect ratio increases as they are farther from the center (see Figure 5, *bottom,left*). Points in the first case were inserted in the reverse order than in the second. While in first case (Figure 4a), last inserted point was the up-most one, in the second (Figure 4b) the last inserted point was the one in the center of the graph. Since the test that checks if a point is inside a circumellipse depends on the ellipse associated to the current inserted point, the algorithm has performed different swaps in each case.

A remark done by both authors of papers [Mav90] and [Pos93] is that the original set of points must be dense enough with respect to the deforming function, that is, the function must not present sharp changes between neighboring points; otherwise, the output triangulation could not reflect these

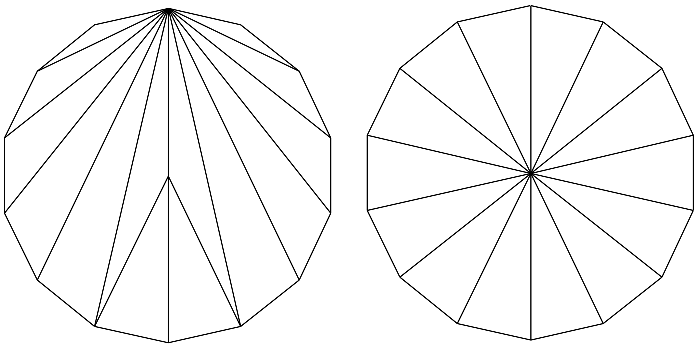

Figure 4: Simple example that shows that the output triangulation produced by the modified algorithm in case II depends on the order of the points are inserted. Points in the left triangulation have been inserted in the reverse order than in the right triangulation.

changes. This effect is exactly what happens in this test case: the ellipses associated to the central point (a circle) is very different from the ones associated to the external points. Consequently, the output triangulation cannot agree with these changes, and triangles cannot show the desired behavior (edges aligned according to direction of the elliptical axes).

Figure 5 shows the results of applying the directional algorithm to a quite large graph, composed of a clover-shaped polygon and a set of internal points. The deforming ellipses are the same that in last example, and internal points in each leaf of the clover are distributed in a different way: points in northwest leaf are placed forming a quadrilateral mesh, northeast leaf contains a lot of points nearly random distributed, southwest leaf has few points, and southeast leaf encloses many points, in concentric subsets. This causes the directional CDT algorithm to behave in a different way on each leaf. In northwest leaf the algorithm creates the same type of triangles than in the non-modified CDT version, the only difference is that edges point to the center of the clover. Since there are a lot of points in the northeast leaf, the algorithm can create the triangles as needed, and the results are good in this region. By the contrary, the lack of internal points in southeast leaf causes that the algorithm cannot create the appropriate triangles, as in the last example. Finally, in the southwest leaf some triangles are quite nice and some other are not, because of the irregular distribution of points in this region.

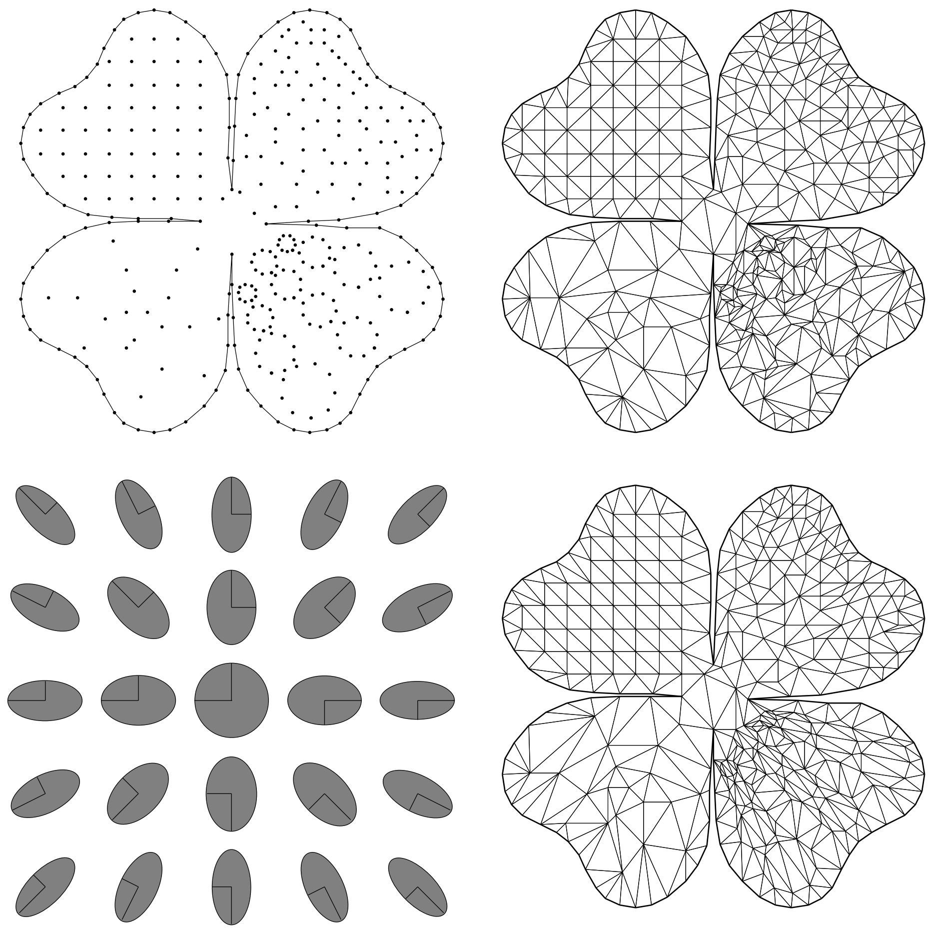

Figure 5: Second example. *Up*, original graph and usual CDT; *bottom*, deforming ellipses evaluated at several points of the plane and output triangulation.

5 Conclusions

Two generalizations of the incremental algorithm for obtaining a CDT of a general graph have been presented. First generalization allows to obtain a CDT that works with an elliptical distance. The second algorithm obtain a triangulation in the case there exists a different deforming ellipse associated to each point of the plane. The problems that can present the directional CDT algorithm have been analyzed, due to the multiplicity of solutions in some cases. Since the algorithms are based on the incremental CDT algorithm of [Vig97], they are easy to generalize, although they are also robust and efficient. Examples show that the results agree with the main aim, that is, that the shape of the output triangles depends on the directionality of the deforming ellipses, whenever possible.

The directional CDT algorithms were developed for a meshing application that triangulates parametric surfaces that are adaptive to the curvature — in terms of length and direction — of the patches. The procedures have been integrted there, improving the resulting triangulations. Furthermore, the algorithm can be applied in other environements, wherever the usual CDT is performed and some directional information could be of usefulness. The only requirement is to supply a simple function that maps an ellipse to each point of the plane, being the ellipse an indicator of the directional information. Other fields of applications include mesh simplification, digital terrain modeling, surface reconstruction and surface approximation.

References

[CD86] L.P. Chew and R.L. Drysdale. Voronoi diagrams based on convex distance functions. Technical Report Tech Report PCS-TR86-132, Darmouth College, Dept. of Computer Science, nov 1986.

[Mav90] D. Mavriplis. Adaptative mesh generation for viscous flows using Delaunay triangulation. *Journal of Computational Physics*, pages 271–291, 1990.

[NF89] G.M. Nielson and T.A. Foley. A survey of applications of an affine invariant norm. In Tom Lyche and Larry L. Schumaker, editors, *Mathematical Methods in CAGD*, pages 445–467, 1989.

[OBS92] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellatons. Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons Ltd, 1992.

[Pos93] M.K. Posenau. Approaches to high aspect ratio triangulations. In *Proceeds of 5th canadian Conference on Comp. Geom.*, Waterloo, Ontario, 1993.

[Vig95] M. Vigo. An incremental algorithm based on edge swapping for constructing constrained Delaunay triangulations. Technical report, Universitat Politecnica de Catalunya, 1995.

[Vig97] M. Vigo. An improved incremental algorithm for constructing restricted Delaunay triangulations. *Comput. & Graphics*, 21(2):215–223, 1997.