An improved incremental algorithm for constructing restricted Delaunay triangulations

Marc Vigo Anglada

Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Abstract

This work presents an algorithm that given a generalized planar graph obtains its constrained Delaunay triangulation (CDT). The proposed method, which follows the general approach of [3], works incrementally based on two improved procedures. The first improvement gives a procedure that inserts a new point in a CDT; the second one is an algorithm that enforces a new constraining edge in a CDT. In particular, an algorithm that generates the CDT of a given polygon (possibly with holes) is also obtained. These algorithms have been implemented and included in several applications, showing their robustness and eficiency, even when the original graph has many vertices or edges.

1 Introduction

Many authors have studied the problem consisting of, given a set of points, obtain its Delaunay triangulation (and its dual, the Voronoi graph), and many different algorithms to solve it have been proposed [4], [1], [7]. Other works deal with the restricted version of this problem: given a set of points and a set of non-crossing edges between these points, obtain a Delaunay triangulation that includes these edges [2], [3]. This problem is known in computational geometry as the restricted Delaunay triangulation, in brief, the CDT. It has practical applications in several fields: the triangulation of surfaces, finite element methods, terrain modelling and object reconstruction, among others.

Most of the algorithms proposed for obtaining a Constrained Delaunay Triangulation (with the exception of [3]) require all the points and edges that form the entry to the algorithm to be known beforehand. This condition is not always possible, and often the points and edges are obtained in an incremental manner. Some applications for example, require calculation of an initial triangulation and its re-definition in certain zones, with the addition of new points and edges.

Our goal is to obtain an incremental algorithm that, given a planar graph $\mathcal{G} = (\mathcal{A}, \mathcal{V})$, constructs a Delaunay triangulation of $\mathcal{V}$ restricted by the edges of $\mathcal{A}$.

2 Terminology and definition of the problem

In this section we will give a series of definitions for the notation used and so be able to delimit the problem to be solved.

Definition 1 (Planar graph). Let $\mathcal{V}$ be a finite set of n vertices in $\mathbb{R}^2$, and let $\mathcal{A}$ be a set of edges determined by the vertices of $\mathcal{V}$. A planar graph is the pair $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ that fulfils:

1. for each edge $ab \in \mathcal{A}$, $ab \cap \mathcal{V} = \emptyset$, and
2. for each edge pair $ab \neq cd$ in $\mathcal{A}$, $ab \cap cd = \emptyset$.

Definition 2 (Triangulation). A triangulation is a planar graph $\Delta = (\mathcal{V}, \mathcal{A})$ such that $\mathcal{A}$ is maximal.

Due to its maximality, the edges of $\mathcal{A}$ include the convex hull in $\mathcal{V}$ and divide the interior into non-overlaping triangular faces, the triangles of Δ.

Any planar graph $\mathcal{G} = (\mathcal{A}, \mathcal{V})$ can be augmented with a set of edges $\mathcal{A}'$ until they become a triangulation, $\Delta = (\mathcal{V}, \mathcal{A} \cup \mathcal{A}')$. If $\mathcal{A} \neq \emptyset$, we call Δ a *restricted triangulation* of $\mathcal{G}$ by the set $\mathcal{A}$.

Definition 3 (Delaunay triangulation). A triangulation $\Delta = (\mathcal{V}, \mathcal{A})$ is considered Delaunay if all edges $ab \in \mathcal{A}$ fulfil the so-called *empty circumcircle property* (with respect to the set of points $\mathcal{V}$) that is to say, there is a circle that passes through a and b such that the other points of $\mathcal{V}$ are exterior to the circle.

In the non-degenerate case, excluding the case in which four or more points of the original set are co-circular, the Delaunay Triangulation for a given set of points is unique, while in the degenerate cases alternatives are accepted. Note that the condition of the empty circumcircle for the edges can also be stated in the same way for the triangles: for each triangle of a Delaunay Triangulation, the interior of the circumcircle of its vertices contains no point of the original set $\mathcal{V}$.

Given three vertices a, b and c, the triangle will be designated $T(a, b, c)$, and the circle that passes through the three points (or circumcircle of the triangle) will be designated $CC(a, b, c)$.

Definition 4 (Visibility). Two vertices a and b of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ are visible (in $\mathcal{G}$) if the segment ab does not cut any of the edges of $\mathcal{A}$.

Definition 5 (Constrained Delaunay Triangulation). Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a planar graph with $\mathcal{A} \neq \emptyset$. $\Delta = (\mathcal{S}, \mathcal{A} \cup \mathcal{A}')$ is a Constrained Delaunay Triangulation of $\mathcal{G}$ if the edges $ab \in \mathcal{A}'$ are such that a and b are visible in $\mathcal{G}$, and ab fulfils the property of the empty circumcircle with respect to the vertices only visible from a and b [6].

A Delaunay Triangulation will be designated by the initials DT, and a Constrained Delaunay Triangulation by CDT.

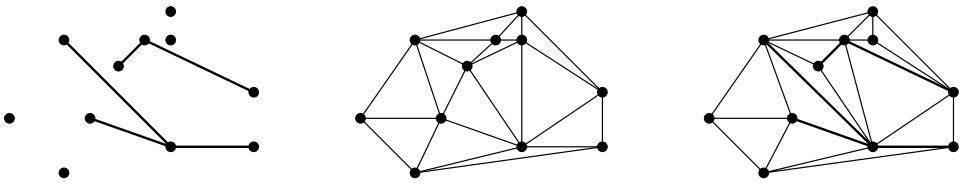

Figure 1: Examples of graphs and triangulations.

The algorithm to be obtained will be incremental, that is to say, it will generate the CDT from a graph by adding each of the points and edges of the graph one at a time.

Consequently, two actions will take place, the first, 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 $(\mathcal{V} \cup \{p\}, \mathcal{A})$; and the second, 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 $(\mathcal{V}, \mathcal{A} \cup \{ab\})$.

The edges of the set $\mathcal{A}$, which constrain the triangulation, are called fixed edges, and as they are incremental, the algorithm will not eliminate them from the CDT.

This definition coincides with that of [3]. However, it differs in the solution found for the method of insertion of vertices as well as in the method of edge insertion. The procedures developed in this paper have the advantage of being simple to implement, and resulting in an efficient code. Moreover, the proposed algorithm to triangulate a face takes advantage of the geometric properties of the original data, accelerating the program.

3 Algorithm for insertion of vertices

The method of point insertion is very well known. It has been used, among others, by Lawson [5] and Sloan [8] in order to obtain an incremental algorithm that calculates the DT of a set of points. This method is based on the following proposition:

Proposition 1. *Let Δ be a DT of a given set of points, S. Therefore, the insertion of one point $p \notin S$ inside the triangulated region, forming the new DT $S \cup \{p\}$, only modifies the triangles of Δ whose circumcircle contains the point p.*

For a proof of this proposition see [5]. The derived algorithm begins by generating a triangle that contains all the points from which the DT can be obtained in order to guarantee that all the points will lie within the triangulation. The points are inserted within the triangulation one at a time; each inserted point implies making a series of changes in edges shared by two triangles, until all the triangles that contain the inserted point have been updated. Lawson demonstrated that this iterative process converges, after a finite number of steps, towards the new DT.

The above proposition can be generalized for constructing a CDT instead of a DT. In this case, it is only necessary to consider the condition of visibility between points, thus providing a new proposition:

Proposition 2. *Let Δ be a CDT of a given set of points, S. The insertion of a point $p \notin S$ inside the triangulated region, forming a new CDT of $S \cup \{p\}$, only modifies the triangles of Δ so that:*

(i) their circumcircle contains the point p

(ii) the three vertices of the triangle are visible from p.

The proof of this proposition is found in [3]. This proposition led to give a new method for the insertion of a point within a CDT, wich is similar to the

one for inserting a point inside DT proposed in [8], but without allowing the edges that constrain the CDT to be altered. Consequently, it is sufficient to maintain a flag in the data structure that identifies the fixed edges, that is to say, those that constrain the triangulation. During the process of edge swap, the fixed edges are not allowed to be modified. The following is the routine, written in pseudo-code, to insert a point in a CDT, under the condition that the point must be inside the region of the plane covered by the CDT.

```
Procedure AddPointCDT(T:CDT, p:Vertex)
    { Precondition:  p ∉ T ∧ p ∈ Interior(T) }
    stack := EmptyStack
    Find the triangle t ∈ T that contains p
    Divide t in three triangles, t1, t2 and t3, depending on p
    Push(stack,t1)
    Push(stack,t2)
    Push(stack,t3)
    While ¬ IsEmpty(stack) do
        t := Pop(stack)
        topo := OpposedTriangle(t,p)
        If the edge shared by t and topo is not fixed and
               p ∈ Circumcircle(topo) then
            Swap the edge shared by t and topo
            Push(stack,t)
            Push(stack,topo)
        EndIf
    EndWhile
EndProc
```

The step that divides a triangle t into three according to an interior point p forms three new triangles. If the vertices of the triangle t were v_1, v_2 and v_3, the new triangles would be $t_1 = (v_1, v_2, p)$, $t_2 = (v_2, v_3, p)$ and $t_3 = (v_3, v_1, p)$. This division is shown in Figure 2a. This action and all those that change the triangulation $\mathcal{T}$ adequately update the information so that new triangles are introduced as necessary, the old ones are erased and the adjacency links are updated.

The function `OpposedTriangle` given a triangle t and one of its vertices, p, returns the adjacent triangle of t such that it is opposed to p, that is to say, the adjacent triangle of t that does not share the vertex p.

The operation of edge exchange, known as *swap*, changes the edge shared by two adjacent triangles that form a convex quadrilateral, for one which joins their un-shared vertices. This operation is shown in Figure 2b.

The time complexity to locate a triangle from a triangulation that contains a given point is $O(n)$ in the worst case, where n is the number of triangles of the triangulation. The iteration has a complexity time that is

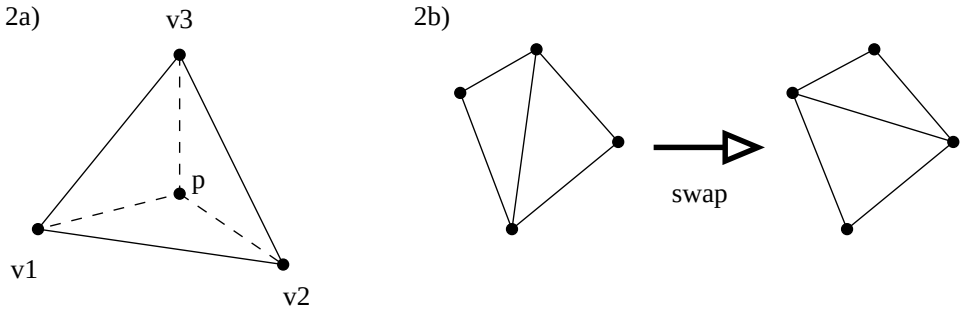

Figure 2: Operation of dividing triangles according to an interior point (2a) and the operation of edge swap (2b).

linear to the number of triangles from the CDT that fulfil the conditions of proposition 2. As this number cannot be greater than n, the worst case time complexity of procedure `AddPointCDT` is $O(n)$.

Lemma 1. *Let $t_1 = T(a, b, c)$ and $t_2 = T(q, b, a)$ be two adjacent triangles of a CDT, where ab is not a fixed edge. Then, any point interior to $CC(t_1)$ opposed to c by the edge ab is also interior to $CC(t_2)$.*

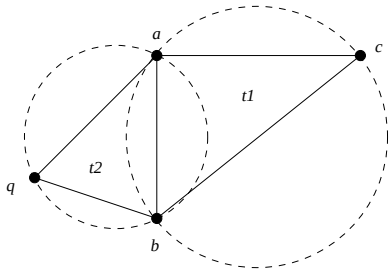

Figure 3: Representation of lemma 1

Proof. As t_1 and t_2 are triangles of a CDT, they fulfill the empty circumcircle property. In particular, the point q must be exterior to $CC(t_1)$, as ab is not fixed. Consequently, the two circumcircles intersect in the points a and b (see Figure 3) and the line through these points splits the circle $CC(t_1)$ in two, one included inside $CC(t_2)$ and another, where the vertex c lies. Therefore, any point inside $CC(t_1)$ opposed to c by the edge ab will be interior to $CC(t_2)$ as well. □

Proposition 3. *The previous algorithm, based on edge swapping, given a CDT of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ and a vertex $p \notin \mathcal{A}$ produces the CDT of the augmented graph $\mathcal{G} = (\mathcal{V} \cup \{p\}, \mathcal{A})$.*

Proof. Proposition 2 indicates that it is only necessary to consider the triangles $t = T(a, b, c)$ that contain the vertex p and whose vertices a, b and c are visible from p. These triangles form a region bounded by a star-shaped polygon Q_p such that the CDT of $\mathcal{G}'$ is obtained by linking all its vertices with p, [3]. Let t be a triangle of Q_p, we will prove that the algorithm will finish by making a swap of its vertices so that external boundaries of Q_p will remain untouched.

In the event that p is inside the triangle t, the initial part of the algorithm will divide the triangle into three, so the vertices of t will connect to p.

Otherwise, as p is outside t, note that there exist only one vertex of t such that the edge of t opposed to this vertex separates the point p and the vertex. This vertex will be called c, and the other two vertices of the triangle will be called a and b (see Figure 4).

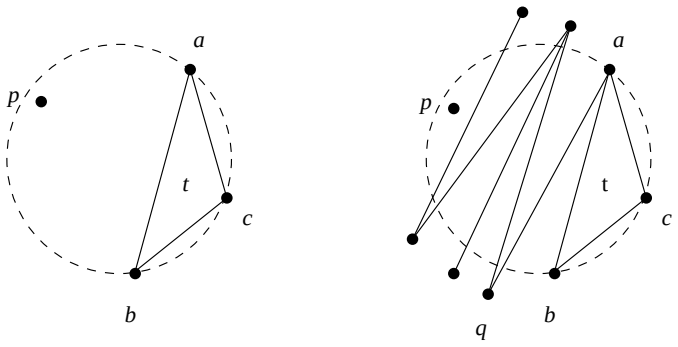

Figure 4: The point p is contained within the circumcircle of t but not within the triangle t.

For t to be a triangle of the CDT of the original graph, its circumcircle, $CC(a, b, c)$ cannot contain any point of $\mathcal{V}$. Nevertheless, there will be a series of edges of the original CDT that separate vertex p from vertex c, with their ends not interior to $CC(a, b, c)$, that can be classified as more or less proximal to the point p (see Figure 4). Note that none of these edges can be fixed; otherwise the vertex c would not be visible from p. We will do induction on the number of edges with these properties.

There will be at least one edge with these characteristics, the edge ab of the triangle t. Therefore, the base case of the induction is that there is only one edge, ab. In this case, the algorithm will have introduced p within the adjacent triangle of t according to edge ab (dividing the triangle into three), and it will therefore have formed the triangle $T(a, b, p)$ and pushed the triangle t to the stack. The algorithm will pop this triangle at some time and consider (and perform if valid) swapping edge ab for cp.

Consider the case where there exist n separating edges, that is, let $t = T(a, b, c)$ be a triangle of the initial CDT such that its circumcircle contains the vertex p, with its three vertices visible from p and that it is

separated from p by n edges (including the edge aq). Without losing generality, suppose that the following edge of those separating the triangle from p is edge aq (see Figure 4; the other possible case is that the following edge is of the type br and will be solved in a symmetrical way). Under these conditions, lemma 1 guarantees that p is inside $CC(a, b, q)$, because t and $T(q, b, a)$ are triangles adjacent from the original CDT, the point p is inside $CC(a, b, c)$ and the line through a and b leaves the points p and q on different half-planes. Furthermore, the vertex q is visible from p since $T(a, b, q)$ was a triangle of the original CDT and, therefore, $CC(a, b, q)$ does not contain any other point of the set $\mathcal{V}$.

By the induction hypothesis, as the triangle $T(a, b, q)$ is separated from p by $n - 1$ edges, with all its vertices visible from p and such that its circumcircle contains p, the algorithm will, at some time, have made the edge swaps to produce the triangles $T(p, b, q)$ and $T(a, b, p)$. Therefore, triangle t (as it is adjacent to $T(a, b, p)$) will have been pushed in the stack, and the algorithm will consider swapping edge ab for pc. □

4 Algorithm for insertion of edges

The algorithm for inserting an edge ab in a CDT of a graph follows the following steps:

1. Remove the triangles $t_1, \cdots, t_k$ cut by ab from the CDT so that a region without triangulation is left.
2. Add the edge ab to the result.
3. Re-triangulate the upper and lower regions of the edge ab that were not triangulated in the first step.

Figure 5 shows the process for insertion of an edge within a CDT. Each of these steps will be explained in detail and the new triangulation obtained shown to be the CDT required.

The first step implies finding the triangle of the CDT that contains the vertex a which is cut by the segment ab. The location of the triangle of a triangulation that contains a point is already well known; the most effective way consists in starting from any triangle in the triangulation and moving to adjacent triangles, according to the classification of the point in question with respect to the edges of the triangle, until the triangle that contains the point is reached. This same procedure is valid for the location of any triangle, t, of the triangulation that has point a as one of its vertices.

From triangle t, we can use the adjacency relationship between the triangles to move through the triangles that converge on a, moving in a clockwise direction until we find the triangle that is cut by the segment ab. This triangle is called t_1 (see Figure 6).

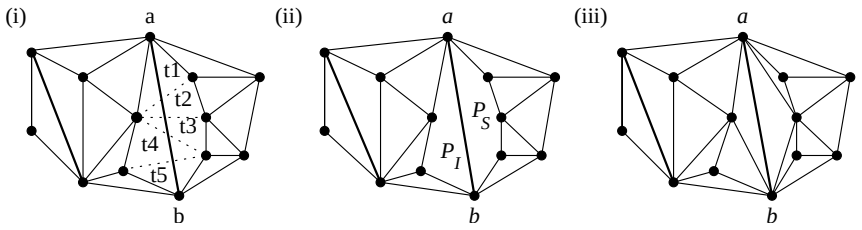

Figure 5: Insertion of an edge within a CDT: (i) location and elimination of the triangles cut by the edge; (ii) insertion of the edge; (iii) re-triangulation of the regions adjacent to the edge.

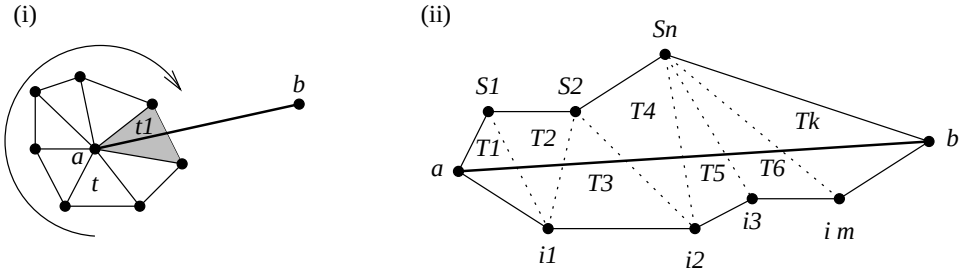

Figure 6: Insertion of an edge: (i) Location of the triangle t_1 that contains ab and is cut by ab (ii) Elimination of the triangles $t_1, \cdots, t_k$ cut by ab.

Once the triangle t_1 has been located, the remaining triangles $t_2, \cdots, t_k$ cut by ab are easily found by using the same adjacency relationship and the classification test of a point with respect to a line. This process also allows simultaneous classification of the vertices of the triangles $\{t_j\}_{j=1}^{k}$ in two sets, that of the vertices above the edge ab, $\{s_j\}_{j=1}^{n}$, and that of those below, $\{i_j\}_{j=1}^{m}$. The series of vertices $\{a, s_1, \cdots, s_n, b\}$ close a region of space located above the edge ab, which is called the *upper pseudo-polygon* and designated P_U; in the same way, the vertices $\{b, i_1, \cdots, i_m, a\}$ limit a region of the space that is called the *lower pseudo-polygon* designated P_L. These two regions, P_U and P_L are those which have to be re-triangulated.

Note that, strictly speaking, the regions P_U and P_L may not be a conventional polygon, since they can contain vertices that are repeated within the series of vertices that form their contour [†]. Figure 7 shows an example where the upper pseudo-polygon contains repeated vertices. This does not complicate the method proposed for the re-triangulation of the pseudo-polygon. It only has to work carefully when establishing the links between the new triangles generated by the re-triangulation and the already existent triangulation.

[†]This circumstance is not specified explicitly in the algorithm proposed in [3]

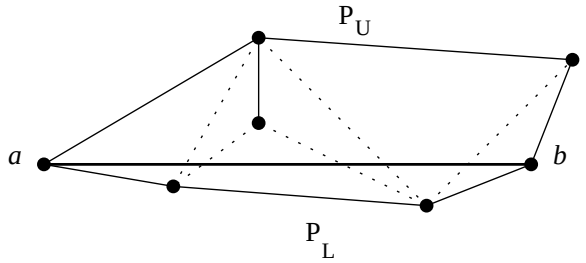

Figure 7: Example in which the upper pseudo-polygon contains repeated vertices

The interior of the pseudo-polygons P_U and P_L is re-triangulated to obtain the new CDT using a recursive algorithm. This is the fundamental point where our proposal differs from [3]. Each of the two pseudo-polygons is triangulated separately, since the new CDT cannot contain edges with one end in P_U and the other in P_L that cut ab, since this edge prevents their visibility. The following is a description of the algorithm for the case of P_U; for P_L the same reasoning is valid.

In the first place, note that all triangulation of the pseudo-polygon P_U must forcibly contain a single triangle with ab as one of its edges, that is to say, a certain triangle with vertices a, b and c, where $c = s_l, 1 \leq l \leq n$. As the triangulation required has to be Delaunay, the triangle $T(a, b, c)$ must fulfil the condition of the empty circumcircle, that is to say that, its circumcircle cannot contain any vertex of P_U (except for its three points a, b and c). The recursive algorithm for triangulating the interior of P_U is derived from this property (Figure 8):

- Find the vertex $c = s_l$ such that the circumcircle of $T(a, b, c)$ does not contain any other point of P_U;
- Form the triangle $T(a, b, c)$, that divides P_U into two sub-regions, $P_E = \{a, s_1, \cdots, s_l\}$ and $P_D = \{s_l, \cdots, s_n, b\}$;
- Recursively apply the algorithm to the regions P_E and P_D, with respect to the edges ac and cb respectively in each case.

The basic case of recursion is that the region to be triangulated is delimited by three or less points, easy cases to prove.

The following shows that the triangulation resulting from the application of the described algorithm is the CDT of the original graph to which the edge ab has been added.

Proposition 4. *Let Δ be a CDT triangulation of a given planar graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ and let ab be an edge such that $ab \notin \mathcal{A}$, $a, b \in \mathcal{V}$. Then, the triangulation obtained by removing the edges from Δ cut by ab, inserting the*

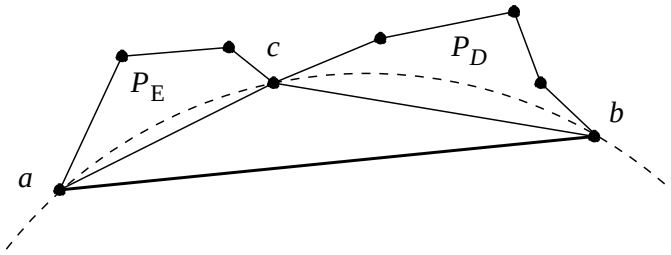

Figure 8: Representation of the recursive algorithm to triangulate a pseudo-polygon.

edge ab and re-triangulating the pseudo-polygons P_U and P_L (resulting from the removal of edges) is the CDT of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{A} \cup \{ab\})$.

Proof. The segment ab must be an edge of the new CDT; therefore, as it is the triangulation a planar graph, the edges cut by ab cannot form part of the new CDT and thus they can be eliminated. Since the points of the graph are the same as those of the original CDT, the edges that remain will continue to satisfy the condition of the empty circumcircle and therefore continue being valid edges of the new CDT. Consequently, the new CDT can be obtained by merging the remains of the CDT after elimination of the edges with the Delaunay Triangulation of the pseudo-polygons P_U and P_L, constrained by the edges that form their contour. □

The following is the routine that inserts an edge into a CDT and the improvement of the action that calculates the Delaunay Triangulation of a pseudo-polygon.

```
Procedure AddEdgeCDT(𝒯:CDT, ab:Edge)
    { Precondition:  a,b ∈ 𝒯 ∧ ab ∉ 𝒯 }
    Find the triangle t ∈ 𝒯 that contains a
       and is cut by ab
    P_U := EmptyList
    P_L := EmptyList
    v := a
    While b not in t do
        t_seg := OpposedTriangle(t, v)
        v_seg := OpposedVertex(t_seg, t)
        If v_seg 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 T
        t := t_seg
    EndWhile
    TriangulatePseudopolygonDelaunay(P_U, ab, T)
    TriangulatePseudopolygonDelaunay(P_L, ab, T)
    Reconstitute the triangle adjacencies of T
    Add edge ab to T
    Mark the edge ab from T as fixed
EndProc
```

The function `Opposedtriangle` is the same as that of the algorithm for insertion of points. The function `OpposedVertex` given two adjacent triangles, t and t_{opo}, returns the vertex of t_{opo} that does not belong to t.

The step that reconstitutes the internal adjacencies is required to reconstruct the adjacency lists between the new triangles resulting from the triangulation of the interior of the pseudo-polygons and the triangles of the initial CDT. One must be careful in the case when a pseudo-polygon contains repeated points, since in this case it is possible that two adjacent triangles of the interior of the pseudo-polygon are linked through an edge that already existed in the initial CDT.

```
Procedure TriangulatePseudopolygonDelaunay(P:VertexList,
   ab:Edge,T:CDT)
    If P has more than one element then
        c := First vertex of P
        For each vertex v ∈ P do
            If v ∈ Circumcircle(a,b,c) then
                c := v
            EndIf
        EndFor
        Divide P into P_E and P_D, giving P = P_E + {c} + P_D
        TriangulatePseudopolygonDelaunay(P_E,ac,T)
        TriangulatePseudopolygonDelaunay(P_D,cb,T)
    EndIf
    If P is not empty then
        Add triangle with vertices a, b, c into T
    EndIf
EndProc
```

Let n be the number of triangles of the initial CDT, and let e be the number of triangles of the CDT cut by edge ab. Then the worst case time complexity of the step that finds the triangle from the CDT that contains

point a and is cut by edge ab is $O(n)$, because locating the point inside the CDT is $O(n)$ and finding the triangle that has a as vertex and whose opposite edge is cut by ab is also $O(n)$. Constructing the upper and lower pseudo-polygons has a worst case time complexity of $O(e)$. The number of vertices of these pseudo-polygons will be $O(e)$ in the worst case. Procedure `TriangulatePseudopolygonDelaunay` has a worst case time complexity of $O(e^2)$, since in each recursive call the total number of points that bound the left and right pseudo-polygons decreases in one unit. Then, as the number of edges from a triangulation is linear to the number of vertices, procedure `AddEdgeCDT` has a worst case time complexity of $O(n^2)$.

5 Algorithm for Delaunay triangulation of a face

The above algorithms allow us to obtain a CDT of any generic graph. Specifically, they can be used to obtain the Delaunay Triangulation of a face. In the first place, we define a face as a particular case of a graph.

Definition 6 (Polygon). A polygon is a planar graph $\mathcal{P} = (\mathcal{V}, \mathcal{A})$ in which the edges of $\mathcal{A}$ form a single cycle. This cycle is the *boundary* of the polygon. It divides the plane in a bounded region, its interior, $\overset{\circ}{\mathcal{P}}$, and an unbounded region, its exterior.

Definition 7 (Face). A face is a planar graph formed by a set of polygons $\mathcal{P}_1, \ldots, \mathcal{P}_N$ in which

1. $\overset{\circ}{\mathcal{P}}_i \subset \overset{\circ}{\mathcal{P}}_1 \ \wedge \mathcal{P}_i \cap \mathcal{P}_1 = \emptyset, 1 \leq i \leq N \ \wedge$
2. $\overset{\circ}{\mathcal{P}}_i \cap \overset{\circ}{\mathcal{P}}_j = \emptyset, \ 1 < i, j \leq N, \ i \neq j.$

$\mathcal{P}_1$ is denominated the *exterior polygon* and the other polygons are *holes* in the face.

All faces divide the plane in two regions, its interior, which can be expressed as $\overset{\circ}{\mathcal{P}}_1 - \cup_{j=2}^{N} \{\overset{\circ}{P}_j\}$, and an unbounded one, its exterior.

The Delaunay Triangulation of a face is no more than the CDT of its graph, that is to say, the DT of its vertices constrained by the set of its edges. Consequently, the above algorithms can be used to obtain the CDT of the face, adding the vertices and edges to the face in an incremental way. We must, however, begin with an initial CDT that contains all the points of the face, since this is required as a precondition by the algorithm for point insertion. As in proposal [8], the initial CDT is formed by a single triangle (the super-triangle) that contains the face. This can be easily calculated from the box containing the vertices of the face.

Normally, when speaking of the triangulation of a face it is understood that we are speaking of covering the inside of the face with triangles (disjointed). In our case, the CDT of the face that we obtain not only covers its interior, but also part of its exterior (it includes all the regions included within the super-triangle). Covering the polygon only requires removal of the triangles that are exterior to the face of the resulting CDT. This can be done simply through a seed algorithm that, starting from an exterior triangle (any of those with one of the vertices of the super-triangle is valid), moves through all the triangles of the CDT following the adjacency links. At each step, each triangle is marked as exterior or interior, depending on the parity of the number of times a fixed edge within the CDT has been passed. Once this marking process has finished the only triangles required are those identified as interior.

Since the algorithm is incremental, the order in which the points and the edges of the graph are introduced is free. The difficulty of introducing an edge within a CDT depends on the number of triangles that are cut by the edge; it is better to add the edges to the CDT as soon as possible (in this way, they cut less triangles). Another reason for introducing the edges as soon as possible is that we thus avoid swap actions that otherwise would be carried out when inserting new points to the CDT (recall that fixed edges limit the visibility between points and, therefore, cut the process of edge swapping). For the same reason, it is better to insert the edges of the interior of the graph as soon as possible as they will probably limit the visibility from the outer-most points.

These considerations have led to a proposal for the triangulation of a face according to the following premises:

- Insert the points and the edges in the order defined by the polygons that limit the face.
- Insert the vertices and edges of the holes of the face first and then the vertices and edges of the exterior polygon.

The first condition is based on the observation that two vertices that usually form an edge of the contour of a face are close together, and therefore when inserted within the CDT they will not cut too many triangles. The following is our proposal for an algorithm for triangulating a face based on the incremental introduction of the vertices and edges of the face.

```
Function CDTFace(C:Face) returns Triangulation
    Compute the super-triangle that contains C
    T := triangulation with only the super-triangle
    For each polygon P hole of C do
        For each vertex v of P do
            AddPointCDT(T,v)
```

```
            x := NextVertex(v,P)
            AddEdgeCDT(T,vx)
        EndFor
    EndFor
    P := Exterior polygon of C
    For each vertex v of P do
        AddPointCDT(T,v)
        x := NextVertex(v,P)
        AddEdgeCDT(T,vx)
    EndFor
    Remove exterior triangles
    return T
EndFunction
```

6 Testing

Several tests were made to test the robustness and efficiency of the algorithm. Figure 9 shows five polygons that have been triangulated using the procedures that have been exposed: case a is a simple polygon; case b is a polygon with vertices in its interior; case c is also a polygon with interior vertices that has been specially designed in order to test the timing of the algorithms; case d is a face with holes; and case e is a face with holes and interior vertices. The CPU times for the proposed algorithms are shown in table 1, together with the number of entities for each test case. Test case labelled in the table as "square" is a rectangular polygon with 400 edges bounding its exterior and 10.000 random points in its interior. The procedures have been implemented in C on a SUN/Sparc Station 20 and times were measured using internal clock. The table shows the number of vertices (including the interior ones), the number of restricting edges (edges of the polygon) and the number of edges in the resulting triangulation for each case. CPU times are shown for the procedure that inserts a new vertex inside a CDT (`AddPointCDT`) and for the procedure that enforces an edge between two vertices of a CDT (`AddEdgeCDT`).

From the table, it can be seen that including a new vertex in a CDT is more expensive than enforcing an edge between two vertices. Only in case c, specially designed, timing for the procedure `AddEdgeCDT` seems to be significative compared with the time spent by `AddVertexCDT`. The polygon from figure 9c has many long boundary edges that separate points leaving some on one side and some on the other.

The observed run times for test cases show a linear behaviour on the number of geometrical entities, even in the case of the square, which contains a large amount of vertices, although this was not expected after theoretical

Polygon	Vertices	Restricting edges	Resulting edges	CPU time (in sec.)		
				AddVertex	AddEdge	Total
figure 9a	33	33	63	0.02	0.00	0.02
figure 9b	76	31	194	0.04	0.00	0.04
figure 9c	98	47	244	0.05	0.02	0.07
figure 9d	204	204	432	0.11	0.02	0.13
figure 9e	346	249	756	0.23	0.04	0.27
square	10400	400	30298	10.33	0.51	10.84

Table 1: Timing statistics for polygons of figure 9.

results. This discrepancy is caused by the nature of the data, as stated in section 5: each vertex tends to be close to the previous one because they form an edge of the polygon (this fact is also discussed in [8]). This behaviour has also been observed in the applications where the proposed algorithms have been included: an application for discretizing trimmed surfaces, the triangulation of polyhedra in a CAD system for modelling solids, as part of an algorithm that calculates the MAT (medial axis transform) of 2-D polygons and within a system for rendering using radiosity that required configuration of a triangular mesh constrained by internal edges.

7 Conclusions

We have presented an algorithm for obtaining the Constrained Delaunay Triangulation of a general graph. The proposal works incrementally, and so allows the addition of a point to the CDT as well as the addition of an edge that constrains it. Finally, we have shown that this approach allows us to obtain the Delaunay Triangulation of the interior of any face.

This algorithm has been implemented and tested in several applications, showing to be a robust algorithm that is highly efficient even when the original graphs contains a large number of points and edges.

The implementation of this algorithm program is in the public domain. To obtain the source program, written in programming language C, contact the computer graphics section of the Software Department or contact the author directly through the e-mail address

`marc@lsi.upc.es`

8 Acknowledgements

The author wishes to thank Núria Pla for the suggestions made and mistakes found during the provisional draft of this document.

During the draft of this document the author was supported by an FI Grant from the Generalitat of Catalonia.

References

[1] A. Bowyer Computing Dirichelet tessellations. *The Computer Journal* 24, 162, 1981.

[2] L. Paul Chew Constrained Delaunay triangulations. *Algorithmica* 4:97–108, 1989.

[3] L. de Floriani and A. Puppo. An on-line algorithm for constrained delaunay triangulation. *Computer Vision, Graphics and Image Processing*, 54(3):290–300, July 1992.

[4] P. J. Green and R. Sibson Computing Dirichelet tessellations in the plane. *The Computer Journal* 21, 168, 1978.

[5] C.L. Lawson. Software for C1 surface interpolation. In J.R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.

[6] D. T. Lee and H. K. Lin. Generalized delaunay triangulation for planar graphs. *Discrete Computational Geometry*, 1:201–217, 1986.

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

[8] S.W. Sloan. A fast algorithm for constructing Delaunay triangulations in the plane. *Adv. Eng. Software*, 9(1):34–55, 1987.

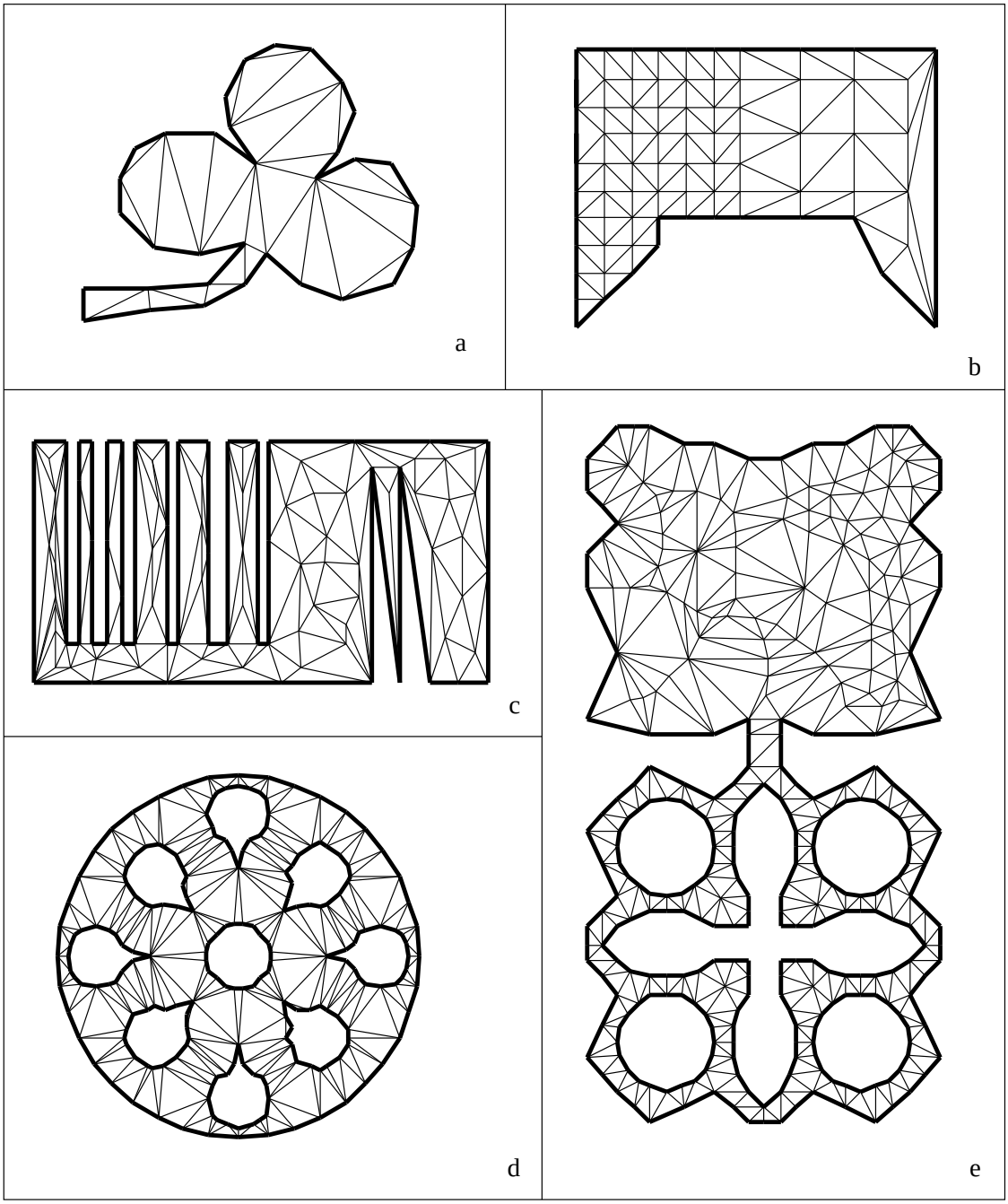

Figure 9: Examples of application of the algorithms.