Mesh Modification Under Local Domain Changes*

Narcís Coll, Marité Guerrieri and J. Antoni Sellarès

Departament d'Informàtica i Matemàtica Aplicada, Universitat de Girona,
{coll,mariteg,sellares}@ima.udg.es

Summary. We propose algorithms to incrementally modify a mesh of a planar domain by interactively inserting and removing elements (points, segments, polygonal lines, etc.) into or from the planar domain, keeping the quality of the mesh during the process. Our algorithms, that combine mesh improvement techniques, achieve quality by deleting, moving or inserting Steiner points from or into the mesh. The changes applied to the mesh are local and the number of Steiner points added during the process remains low. Moreover, our approach can also be applied to the directly generation of refined Delaunay quality meshes.

1 Introduction

In many two-dimensional geometric modelling problems it is desirable to obtain a triangular mesh that respects the domain of interest ensuring that the triangles of the triangulation satisfy some quality requirements. There exist many works on the generation of a quality mesh for a Planar Straight Line Graph (PSLG) domain. Delaunay refinement mesh generation algorithms have taken place in this frame of investigations [12, 13, 11, 7]. In keeping quality of a mesh two objectives are pursued. First, get skinny triangles, triangles without the required quality, out of the mesh. Second, force segments of the PSLG into the mesh. Both goals are achieved by the addition of Steiner points, points that do not belong to the original mesh. In current Delaunay refinement algorithms, two kinds of Steiner points deal with the former goal, namely, circumcenters and off-centers. The later objective is carried out by the addition of midpoints on constrained segments to insert. Local optimization techniques, like refinement, derefinement, topological changes and mesh smoothing, are employed usually as a postprocess to improve mesh quality [1, 5].

In this work we address the problem of adjusting a mesh to local changes of its domain. The initial motivation for this problem came from our interest

*Partially supported by the Spanish Ministerio de Educación y Ciencia under grant TIN2004-08065-C02-02.

on the simulation of cuts in triangulated objects. We modify a quality mesh of a PSLG under the insertion/removal of elements (points, segments, polygonal lines, etc.) to/from the PSLG, while keeping the quality of the mesh along the process. There exist some work related to the dynamic insertion and deletion of points and segments on Delaunay triangulations [6, 8], although they do not fit into the schema of Delaunay refinement algorithms.

Obviously, when a PSLG is changed, we can use a Delaunay refinement algorithm to modify the underlying mesh: the updated PSLG can be considered as input PSLG and a new mesh can be generated from scratch. However, when a PSLG is dynamically modified, apart of the quality requirement of its underlying mesh, we expect that additional features are going to be provided, namely:

Incrementality: The mesh of the updated PSLG is obtained without the regeneration of the whole mesh.

Locality: The changes applied to the mesh do not imply a propagation of these modifications to the whole mesh.

Optimality: The Steiner points added as a result of the modification of the mesh should be as few as possible.

A possible incremental solution to the insertion problem, alternative to generate a new mesh from scratch, is to apply a Delaunay refinement algorithm to the PSLG obtained joining: the current PSLG, the Steiner points of the current mesh and the elements to be inserted into the PSLG. In this way we can take advantage of the work done during the generation of the current mesh. However, considering Steiner points of the current mesh as part of the new PSLG contributes to the degradation in the distribution of Steiner points in successive updates of the mesh, generating a high number of small triangles, as we will show with some examples below. Suppose we have an initial PSLG composed of a square boundary and two points a and b (Figure 1(a)). After applying Ruppert's Delaunay refinement algorithm we obtain the mesh shown in Figure 1(b). The insertion of a new point, c, onto the PSLG close to an existing Steiner vertex, s, creates two new skinny triangles that have to be refined (Figure 1(c)). Then, an incremental Delaunay refinement algorithm generates the mesh in Figure 1(d).

The solution we propose to avoid the excessive insertion of vertices and the generation of small triangles produced by a degradation in the distribution of points in successive updates of the mesh is the deletion or the movement of Steiner vertices, followed if necessary by the addition of Steiner vertices according to Ruppert's algorithm. In Figure 1 we can compare the previous result of applying a Delaunay incremental refinement algorithm, Figure 1(d), with the movement of the Steiner vertex s to a suitable zone, in Figure 1(e), and with the deletion of the Steiner vertex, shown in Figure 1(f). From these examples one can observe that moving the vertex results in a much better mesh than in the case of using the incremental algorithm, but the optimal solution is obtained by the deletion process. This suggests that removal of

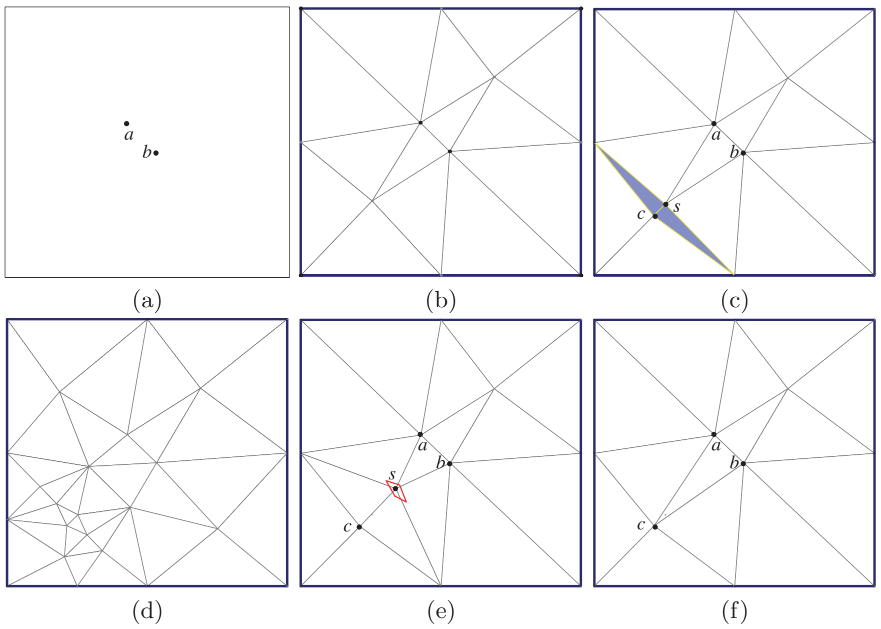


Fig. 1. (a) An initial PSLG composed of a square boundary and two points. (b) A Delaunay refined triangulation of the PSLG. (c) A new point to be added to the mesh with the skinny triangles associated to its insertion in grey. (d) The resulting mesh after applying an incremental Delaunay refinement algorithms to the mesh in Figure 1(c). (e) Result obtained if the Steiner vertex is moved to a suitable region. (f) Mesh achieved if the Steiner vertex is deleted.

Steiner vertices will therefore have priority in front of movement of Steiner vertices.

In this paper we introduce a framework, that combines mesh improvement techniques, for modifying incrementally a mesh under local changes of its PSLG domain. Starting from a quality mesh of the initial PSLG, we insert/remove elements to/from the PSLG in such a way that as the PSLG changes the underlying mesh is modified keeping its quality during the process. Our algorithms make only local modifications: deletion, movement or insertion of Steiner points from/into the mesh. Moreover, the number of Steiner points added during the process remains low.

One of the main ideas behind our proposed algorithms is that the bad vertex of some skinny triangles can be moved to a *quality zone* ensuring that a prefixed mesh quality is obtained. We give a numerical method for finding the optimal placement where the vertex has to be moved.

Our framework can also be applied to directly generate refined Delaunay quality meshes. We give initial experimental results showing that the number of Steiner points obtained with our approach is smaller compared to the number obtained when traditional circumcenter refinement methods are used.

2 Preliminaries

A *Planar Straight Line Graph* (PSLG) is a set of points and segments satisfying two constraints: all endpoint segments are points in the PSLG, segments may intersect each other only at their endpoints.

A triangulation $\mathcal{T}$ is a *conforming triangulation* of a PSLG, Ω, if each *point* in Ω corresponds to a *vertex* in $\mathcal{T}$, and a *segment* of Ω is represented by a linear contiguous sequence of *edges* of $\mathcal{T}$. New Steiner vertices, not points of Ω, may appear, and each segment of Ω may have been subdivided into shorter edges by these additional vertices. Flipping these edges is forbidden, then they are marked as *locked.* In a *conforming Delaunay triangulation* of a PSLG, the Steiner vertices are added so that the Delaunay property is maintained.

The *star* of a vertex q, $\mathcal{S}_q$, of a triangulation $\mathcal{T}$ consists of all the triangles of $\mathcal{T}$ that contain q. The *link* of q, $\mathcal{L}_q$, is the polygon determined by the set of edges of the triangles in $\mathcal{S}_q$ that are not incident to q. Since the average degree of a node in a planar graph is less than six [3], the average number of triangles of $\mathcal{S}_q$ or the average number of edges of $\mathcal{L}_q$, is at most six. The *kernel* of $\mathcal{L}_q$, denoted by $ker(\mathcal{L}_q)$, is the set of all points $p \in \mathcal{L}_q$, such that for every vertex v of $\mathcal{L}_q$, the segment vp is within $\mathcal{L}_q$.

Given an edge $e \in \mathcal{L}_q$, being e_i and e_f its endpoints, we take the following notation (see Figure 2):

- $H_{q,e}$ is the open half-plane determined by e and containing the vertex q.
- $t_{q,e}$ is the triangle with vertices e_i, e_f and q.
- $t'_{q,e}$ the adjacent triangle to $t_{q,e}$ by e.
- $c_{q,e}$ the circumcircle of $t'_{q,e}$.
- $a_{q,e}$ the arc $c_{q,e} \cap H_{q,e}$. We will say that $c_{q,e}$ is the supporting circle of $a_{q,e}$.

A triangle having an angle $\beta < \alpha$, for certain fixed α, is called *skinny.*

The *diametral circle* of a subsegment (portion of a segment from a PSLG) is the (unique) smallest circle that contains the subsegment.

A subsegment is said to be *encroached* if a vertex lies strictly inside its diametral circle, or if the subsegment does not appear in the triangulation.

2.1 Incremental Delaunay Algorithm

There exists three types of algorithms for constructing Delaunay triangulations, namely, divide-and-conquer, sweepline and incremental. Because of our goals we concentrate our attention in the latter ones.

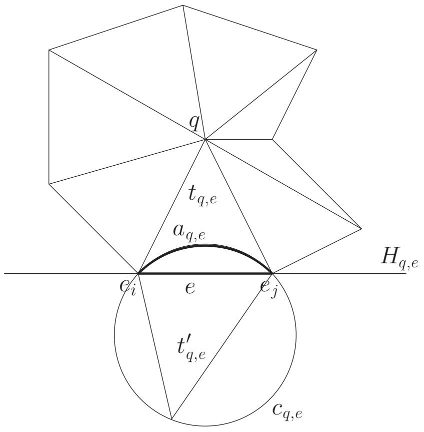


Fig. 2. Notation used in the definitions.

Incremental algorithms add vertices one by one and update the triangulation after each vertex is added maintaining the Delaunay property. The original algorithm, developed by Lawson [9], is based upon edge flips. There are incremental algorithms due to Bowyer [2] and Watson [14] that do not use edge flips. In Lawson's algorithm, when a vertex is inserted, the triangle that contains it is found, and three new edges are inserted to attach the new vertex to the vertices of the containing triangle. If the new vertex falls upon an edge of the triangulation, that edge is deleted, and four new edges are inserted to attach the new vertex to the vertices of the containing quadrilateral. Next, a recursive procedure tests whether the new vertex lies within the circumcircles of any neighboring triangles, each affirmative test triggering an edge flip that removes a locally non-Delaunay edge. Each edge flip reveals two additional edges that must be tested.

2.2 Ruppert's Algorithm

The Delaunay refinement algorithm, first described by Ruppert [12], refines the mesh by inserting additional *Steiner* vertices (using Lawson's algorithm to maintain the Delaunay property) until all triangles satisfy the quality constraint. Vertex insertion follows two rules:

- Any *encroached* subsegment is bisected by inserting a vertex at its midpoint.
- Each *skinny triangle* is normally split by inserting a vertex at its circumcenter. The Delaunay property guarantees that the triangle is eliminated. However, if the new vertex would encroach upon any subsegment, then it is not inserted; instead, all the subsegments it would encroach upon are split.

Encroached subsegments are given priority over skinny triangles.

Ruppert's algorithm guarantees the following properties:

- *Edges of the mesh well-graded.* New edges generated are greater than the smallest distance between two non-incident features of the input PLSG.
- *Optimal size of the mesh.* The number of Steiner points added is within a constant factor of the minimum number of points added by *any* meshing of the same input.
- *Termination.* In order to ensure Ruppert's algorithm termination the upper bound for α is $\arcsin \frac{1}{2\sqrt{2}} \approx 20.7°$, angles between incident segments in the input PSLG greater or equal than $90°$ are required, and co-circular points are not allowed. Shewchuk [13] relaxes this minimum angle requirement to $60°$. Pav, in [11], showed that the algorithm terminates for a wider class of input than previously suspected, establishing this bound in $45°$. In fact, other considerations have to be taken into account with respect to the input, but they rarely appear in practice.

3 Quality Zones

One of the main ideas behind our proposed algorithms is that some skinny triangles can be eliminated by moving the Steiner points corresponding to their bad vertex. The solution relies on the fact that, for a given Steiner vertex, it is possible to define a zone where it can be moved ensuring that a prefixed mesh quality can be obtained. To make sure that the result is a valid triangulation, a Steiner vertex q has to be moved to a point in $ker(\mathcal{L}_q)$ and then the Delaunay property among the new triangles has to be maintained by flipping locally non-Delaunay edges.

3.1 Definitions

Let $\mathcal{T}$ be a Delaunay refined triangulation of a PSLG and α be a given quality angle.

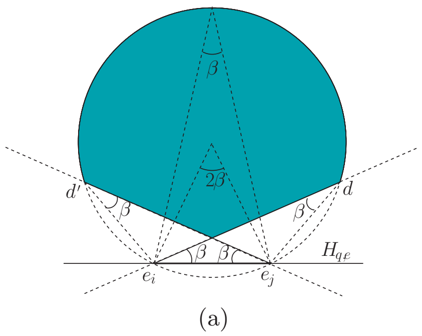
The *feasible zone of an edge* $e \in \mathcal{L}_q$ for an angle β is the set $\mathcal{F}_{e,\beta} = \{p \in H_{q,e} | \widehat{e_i p e_f} \geq \beta, \widehat{p e_f e_i} \geq \beta, \widehat{e_f e_i p} \geq \beta\}$ (see Figure 3(a)).

As can be observed in Figure 3(a), the feasible zone is a convex region obtained as intersection of two half-planes and a circle. Let d be the point in $H_{q,e}$ such that $e_i e_f d$ is an isosceles triangle with $\widehat{d e_i e_f} = \widehat{e_i e_f d} = \beta$. One half-plane is defined by the line through e_i and d that does not contain e_f, and the other half-plane is defined by the line through e_f and d' that does not contain e_i. The center f of the circle is the point of $H_{q,e}$ located on the bisector of $e_i e_f$, such that $\widehat{e_i f e_f} = 2\beta$. Then, from a well known geometric property, at any point p on the circle boundary the following expression is satisfied: $\widehat{e_i p e_j} = \beta$.

The *feasible zone of a vertex* q for an angle β is the set $\mathcal{F}_{q,\beta} = \bigcap_{e \in \mathcal{L}_q} \mathcal{F}_{e,\beta}$.

The *non-skinny zone of a vertex* q is the set $\mathcal{F}_{q,\alpha}$.

The *Delaunay zone of an edge* $e \in \mathcal{L}_q$, denoted $\mathcal{D}_{q,e}$, is the set of points of $H_{q,e}$ external to $c_{q,e}$ (see Figure 3(b)).



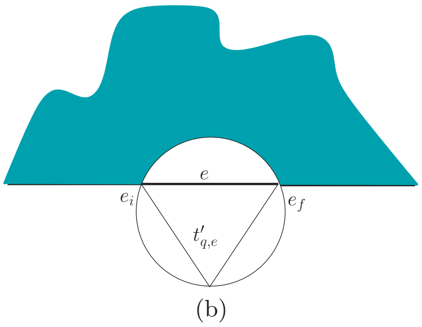


Fig. 3. (a) A feasible zone $\mathcal{F}_{q,e,\beta}$. (b) Delaunay zone $\mathcal{D}_{q,e}$.

The *Delaunay zone of a vertex* q is the set $\mathcal{D}_q = \bigcap_{e \in \mathcal{L}_q} \mathcal{D}_{q,e}$.

The Delaunay zone of a vertex is an open non-convex set and, as exhibited in Figure 4, may be constituted by several non-connected components. The boundary of $\mathcal{D}_q$ will be denoted by $\overline{\mathcal{D}_q}$.

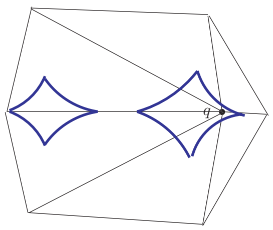


Fig. 4. Delaunay zone with two non-connected components.

The *quality zone of a vertex* q for the angle α is the set $\mathcal{Q}_{q,\alpha} = \mathcal{F}_{q,\alpha} \bigcap \mathcal{D}_q$ (see Figure 5).

From the last definition it is clear that if $p \in \mathcal{Q}_{q,\alpha}$ then the triangles of $\mathcal{S}_p$ are non-skinny and the exterior adjacent triangles to $\mathcal{L}_p$ edges are Delaunay.

It is not difficult to prove the following:

Lemma 1. *Let q be a vertex of a triangulation $\mathcal{T}$. Then, we have:*

- *If $\mathcal{F}_{q,\beta} \neq \emptyset$, $\mathcal{F}_{q,\beta}$ is a convex set included in $ker(\mathcal{L}_q)$.*
- *If $\mathcal{F}_{q,\beta} \neq \emptyset$ and $\beta' > \beta$, $\mathcal{F}_{q,e,\beta'} \subsetneq \mathcal{F}_{q,e,\beta}$ and $\mathcal{F}_{q,\beta'} \subsetneq \mathcal{F}_{q,\beta}$.*

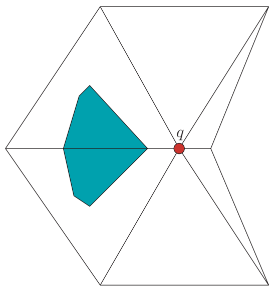


Fig. 5. Quality zone $\mathcal{Q}_{q,\alpha}$

3.2 Finding a Point That Maximizes the Minimum Angle

Let p be a point in $ker(\mathcal{L}_q)$. We denote by $T_q(p)$ the set of triangles determined by p and the edges of $\mathcal{L}_q$. If $\mathcal{L}_p$ is formed by k edges, we have a collection of $3k$ angular functions $\phi_j(p)$, $j = 1, \cdots, 3k$, each one representing an angle of a triangle of $T_q(p)$.

We are interested in finding a point $\widetilde{p} \in ker(\mathcal{L}_q) \cap \mathcal{D}_q$ maximizing the function $\Phi(p) = \min_j \phi_j(p)$. When instead of searching for the point $\widetilde{p} \in \mathcal{D}_q$ we seek a point $p^* \in ker(\mathcal{L}_q)$, we have a collection of quasiconvex functions and the problem can be solved in $O(k)$ time using Generalized Linear Programming [1].

Then, we first find the point p^*. If $p^* \in \mathcal{D}_q$ then p^* is the optimal solution $\widetilde{p}$. Otherwise we find the point $\overline{p}$ that maximizes the function $\Phi(p)$ restricted to $\overline{\mathcal{D}_q}$ and we take a point inside $\mathcal{D}_q$ close to $\overline{p}$ as the target point $\widetilde{p}$.

Instead of implementing the Generalized Linear Programming approach for finding p^*, we describe an alternative numerical technique for solving the problem directly that also allow us to find $\overline{p}$ if it is necessary. The technique is based on the following lemmas:

Lemma 2. *Let l be a line such that $l \cap \mathcal{F}_{q,\beta} \neq \emptyset$. Then, we have:*

1. *$l \cap \mathcal{F}_{q,\beta}$ is a point or a segment.*
2. *If $l \cap \mathcal{F}_{q,\beta}$ is a segment, the midpoint m of $l \cap \mathcal{F}_{q,\beta}$ satisfies $\Phi(m) \geq \beta$.*

Proof. Let $s = l \cap \mathcal{F}_{q,\beta}$. Since $l \cap \mathcal{F}_{q,\beta}$ is a convex set, s is a point or a segment. Let m be the midpoint of s. Suppose that $\Phi(m) < \beta$. By Lemma 1 we know that $\mathcal{F}_{q,\beta} \subset \mathcal{F}_{q,\Phi(m)}$ and consequently m is an interior point of $\mathcal{F}_{q,\Phi(m)}$, which is contradictory with the fact that m is on the boundary of $\mathcal{F}_{q,\Phi(m)}$.

Lemma 3. *Let $a_{q,e}$ be a Delaunay arc whose supporting circle $c_{q,e}$ contains p^*. Let p' be the point that maximizes $\phi(p)$ restricted to $a_{q,e}$, and let $\overline{p}$ be the point that maximizes $\phi(p)$ restricted to $\overline{\mathcal{D}_q}$. If $p' \in \overline{\mathcal{D}_q}$ then $\phi(p') = \phi(\overline{p})$.*

Proof. Since $p' \in \overline{\mathcal{D}_q}$, clearly $\phi(p') \leq \phi(\overline{p})$. Suppose $\phi(p') < \phi(\overline{p})$. Since p' maximizes $\phi(p)$ restricted to $a_{q,e}$, $\mathcal{F}_{q,\phi(p')}$ is tangent to $c_{q,e}$ at p'. Moreover, due to that $\mathcal{F}_{q,\phi(p')}$ is convex, $c_{q,e}$, $p^* \in \mathcal{F}_{q,\phi(p')}$ and $p^* \in c_{q,e}$, we have $\mathcal{F}_{q,\phi(p')} \subset c_{q,e}$. By Lemma 1 we have $\mathcal{F}_{q,\phi(\overline{p})} \subsetneq \mathcal{F}_{q,\phi(p')}$, consequently $\mathcal{F}_{q,\phi(\tilde{p})} \subsetneq c_{q,e}$, which is contradictory with the fact that $\overline{p} \in \overline{\mathcal{D}_q}$.

The maximizing algorithm consists of two main steps, first we find the point p^* that maximizes Φ. If p^* happens to lie inside the Delaunay Zone, $\mathcal{D}_q$, then it is the target point $\widetilde{p}$; on the contrary we have to find $\overline{p}$ on $\overline{\mathcal{D}_q}$. Both steps are based on the optimal gradient method that progressively approaches the point maximizing a function. This method needs the direction $v(p)$ in which the function increases more quickly. In our case we approximate $v(p)$ as follows (see Figure 6):

1 Determine the minimum angle β of $T_q(p)$.
2 If β is adjacent to p, let $v(p)$ be the direction of its angle bisector.
3 If β is not adjacent to p, let r be the vertex of β and let $v(p)$ be the perpendicular vector to rp satisfying that the half-plane determined by p and $v(p)$ does not contain the other vertex of the triangle.

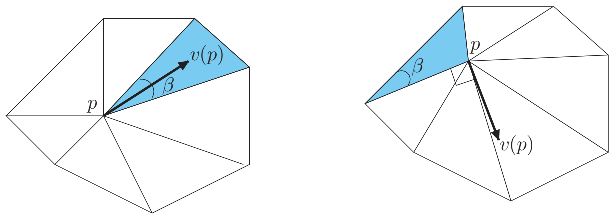


Fig. 6. On the left, β is adjacent to p and increases in the direction of its angle bisector. On the right, β is adjacent to an edge of $\mathcal{L}_q$ and increases in the direction perpendicular to the edge adjacent to p and β.

Figure 7 illustrates the first step of the maximizing algorithm that finds p^* based on Lemma 2. Let p_k be the point obtained after the k-th iteration of the algorithm. The point q is used to compute the initial iteration p_0. The point p_{k+1} is computed from p_k by applying the following steps:

1 Compute the minimum angle β_k of $T_q(p_k)$ and the vector $v(p_k)$.
2 Compute the feasible zone $\mathcal{F}_{q,\beta_k}$.
3 Compute the segment s as the intersection between $\mathcal{F}_{q,\beta_k}$ and the half-line determined by p_k and $v(p_k)$.
4 Take p_{k+1} as the midpoint of s.

The algorithm finishes when the distance between p_k and p_{k+1} is lesser than ε and the difference between β_k and β_{k+1} is lesser than δ, where ε and δ are parameters that permit controlling the accuracy of the solution.

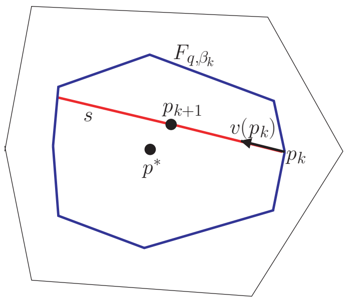
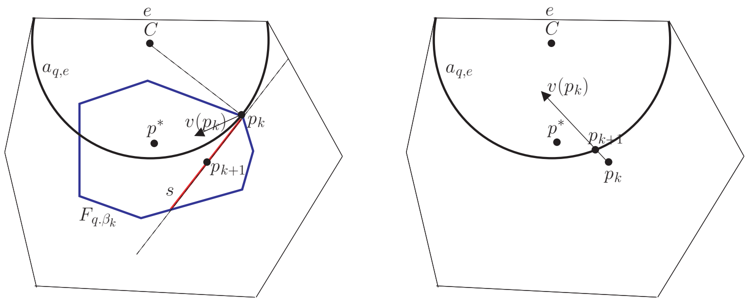


Fig. 7. First step of the maximizing algorithm.

Figure 8 illustrates the second step of the maximizing algorithm, based on Lemma 3, that finds a point $\overline{p}$ on $\overline{D_q}$ maximizing $\Phi(p)$. Let $a_{q,e}$ be a Delaunay arc whose supporting circle $c_{q,e}$ contains p^*. Any point of $a_{q,e}$ is qualified to compute the initial iteration p_0.



(a) $p_1 \in a_{q,e}$. (b) $p_1 \notin a_{q,e}$.

Fig. 8. Second step of the maximizing algorithm.

There are two cases for computing p_{k+1} from p_k. The first one is triggered when p_k lies on $a_{q,e}$ (see Figure 8(a)). In this case we apply the following steps:

1 Compute the minimum angle β_k of $T_q(p_k)$ and the vector $v(p_k)$.
2 Compute the orthogonal projection v of $v(p_k)$ onto the tangent line to $c_{q,e}$ at p_k.
3 Compute the feasible zone $\mathcal{F}_{q,\beta_k}$.
4 Compute the segment s intersection between $\mathcal{F}_{q,\beta_k}$ and the half-line determined by p_k and v.
5 Take p_{k+1} as the midpoint of s.

The second case occurs when p_k does not lie on $a_{q,e}$ (see Figure 8(b)). In this case the point p_{k+1} is taken as the intersection point between $a_{q,e}$ and the half-line determined by p_k and $v(p_k)$.

Only when the point $\overline{p}$ lies on $\overline{\mathcal{D}_q}$, the point $\widetilde{p}$ is taken as the last iteration point not lying on $a_{q,e}$. Then, the described process must be applied to Delaunay arcs whose supporting circle contains p^* until the point $\widetilde{p}$ is found. If the process fails for all these arcs, we compute the set P of the intersection points between the arcs, and we take $\widetilde{p}$ as an interior point of $\mathcal{D}_q$ very close to the point of P maximizing $\Phi(p)$.

4 Basic Operations

Movement and deletion of Steiner points are the *basic operations* used by our improvement process. Steiner points to be treated by the process can belong to two main groups. The first group is formed by the Steiner points located on any segment of the PSLG, and the second group is formed by the remaining Steiner points. We have named *restricted* vertices the points of the first group, since their movement will be restricted to the corresponding segment, and *free* vertices the points of the second group.

4.1 Moving Free Vertices

The key concept regarding the movement of a free vertex q is to substitute this vertex by the best point in the quality zone $\mathcal{Q}_{q,\alpha}$, being α the quality of the mesh. Then, we have to find a point $\widetilde{p} \in \mathcal{D}_q$ maximizing the function $\Phi(p)$ and satisfying $\Phi(\widetilde{p}) \geq \alpha$. In order to do that, we apply the maximizing algorithm explained in the previous section. Observe that the simple insertion of $\widetilde{p}$ into the triangulation guarantees that exterior triangles to $\mathcal{L}_q$ are Delaunay, but does not guarantee the Delaunay property among interior triangles. For this reason, the vertex q has to be deleted and the vertex $\widetilde{p}$ has to be inserted using an Incremental Delaunay algorithm. Then, it is possible that $\mathcal{L}_{\widetilde{p}} \neq \mathcal{L}_q$ and, consequently, $\Phi'(\widetilde{p}) \geq \Phi(\widetilde{p})$, where $\Phi'(p)$ is the function to be maximized in the interior of $\mathcal{L}_{\widetilde{p}}$. In this case, we initiate an iterative process that in each step updates q with the vertex $\widetilde{p}$ obtained in the previous step, applies the optimizing algorithm to q and inserts the new $\widetilde{p}$ using an Incremental Delaunay algorithm. The process finishes when $\mathcal{L}_{\widetilde{p}} = \mathcal{L}_q$. Only when the final $\widetilde{p}$ satisfies $\Phi(\widetilde{p}) \geq \alpha$, the point $\widetilde{p}$ is inserted into the mesh as a Steiner vertex.

4.2 Deleting Free Vertices

Devillers in [4] proposed an algorithm to delete a point from a Delaunay triangulation. Basically, his algorithm retriangulates $\mathcal{L}_q$ by determining *Delaunay* *ears* using the concept of *power* of q. In our case we need to obtain not just a Delaunay triangulation, but a Delaunay refined one. Consequently we verify whether the Delaunay ear is skinny or not, stopping the process when a skinny one is found.

4.3 Moving Restricted Vertices

The movement of restricted vertices is constrained over their correspondent subsegments. This kind of vertices can be present on a boundary subsegment or on a non-boundary subsegment of a PSLG. Since the optimizing algorithm can easily be adapted in order to guarantee that the vertex $\widetilde{p}$ lies on the subsegment, in both cases we apply the iterative process explained in Section 4.1.

4.4 Deleting Restricted Vertices

Deletion of a restricted vertex depends on whether it belongs or not to a boundary subsegment of the PSLG. The presence of a restricted vertex on a non-boundary subsegment implies the application of the deletion algorithm explained in Section 4.2 to the two sides of the subsegment independently. In this way the point is deleted only if each side independently fulfills the point deletion verification, and then the region in each side is retriangulated individually. The same steps from the algorithm of Section 4.2 can be carried out without changes, with the only extra consideration that a restricted vertex on a subsegment can not be deleted if a vertex of its link is encroached by the subsegment. In case of a boundary subsegment the process detailed above is applied only to the interior side.

4.5 Expanding Deletion of Vertices

Once a vertex has been deleted from the mesh other vertices are susceptible of deletion. Each time a vertex is deleted all its adjacent Steiner vertices are added to a queue to be deleted, causing in this way several iterations. The iteration process ends when none of the vertices in the queue can be deleted.

5 Modifying a Delaunay Refined Mesh

Modification of a Delaunay refined mesh means to insert new PSLG elements into the mesh or delete PSLG elements from the mesh. The elements can be points, segments, polygonal lines and polygonal holes.

An element is inserted in the mesh by inserting its points and then checking if each segment of the element is a sequence of edges of the mesh. If the check fails, the segment is inserted by a recursive process that adds its midpoint and checks if the two subsegments are edges of the mesh. The edges corresponding to segments are marked as locked. When the element is a polygonal hole, the triangles located in the interior of the hole are deleted.

An element is deleted by marking its edges as unlocked, considering its points Steiner vertices and trying to delete them by using the Devillers algorithm. When the element is a polygonal hole, we first triangulate its interior.

After the insertion or deletion of an element, a process of improvement is called that expands the quality through the mesh. We could follow two approaches: to apply our improvement process each time a part of the element is inserted or deleted, or apply the process right after the whole element is inserted or deleted. This situation has been illustrated with the insertion of a segment in Figure 9 (first approach) and in Figure 10 (second approach). As can be observed in the figures, the use of the improvement process after each point insertion increases the number of Steiner points. Consequently, in our algorithms we will follow the second approach.

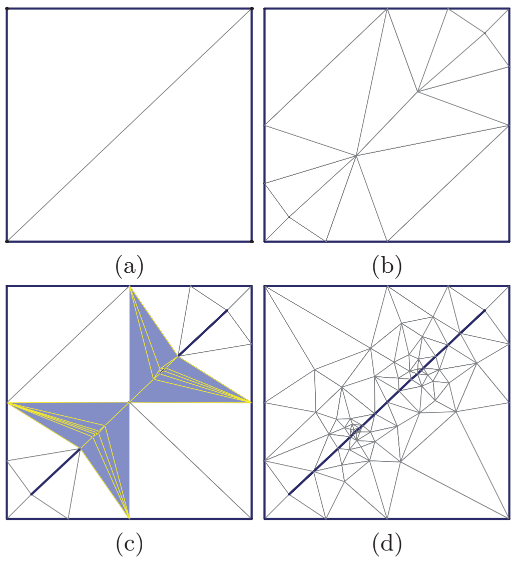


Fig. 9. (a) Initial mesh of a PSLG formed by a square boundary. (b) The mesh after the insertion of two points. (c) The resulting mesh with skinny triangles generated by the insertion of a segment whose endpoints are the points previously added. (d) Resulting mesh obtained applying a improvement process after each partial element addition.

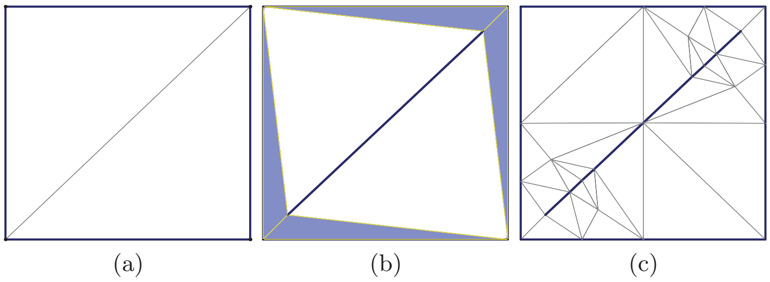


Fig. 10. (a) Initial mesh of a PSLG formed by a square boundary. (b) The resulting mesh with skinny triangles generated by the insertion of a segment. (c) Resulting mesh delaying improvement process at the end.

5.1 Improvement Process

The improvement process receives as input two lists: the list of points to be inserted, initially containing only midpoints of encroached subsegments, and the list of skinny triangles to be removed, originated by the modification of an element. The output of the process is a mesh with the desired quality. The process maintains the two lists and finishes when both are empty. Priority is established on midpoints. To remove a skinny triangle, we first check its Steiner vertices for deletion, then we check its Steiner vertices for movement, and finally we add circumcenters to the list of points. This order of treatment of skinny triangles is important to obtain a reduction in the number of vertices. Following the rule from Ruppert's algorithm encroached circumcenters are not inserted and the midpoint of the encroached subsegments are added to the list of points to be inserted.

6 Generating a Delaunay Refined Mesh

As stated in the introduction, our improvement process can also be applied to generate a refined Delaunay quality mesh of a PSLG. The complete process consists of the following steps. First, a conforming Delaunay triangulation of the PSLG is generated, then the list of skinny triangles to be removed is obtained, and finally our improvement method is applied to eliminate those skinny triangles. Observe that initially the list of points to be inserted is empty.

7 Experimental Results

We have implemented our algorithms in C++ language and using OpenGL libraries to build an interactive interface. Our application takes a triangulated

PSLG as input. This initial mesh is refined until the desired quality is achieved. Also, the mesh can be modified by adding or deleting elements while keeping the quality established.

We have run several simulations in order to test our implementation and we have compared these simulations with meshes generated using *TRIANGLE*, a freely available software produced by Jonathan R. Shewchuck [13] (http://www.cs.cmu.edu/~quake/triangle.html), and an incremental Ruppert algorithm based on the work of Miller et al. [10]. In table 1 we present the statistics and the reference to the correspondent set of outputs. The input PSLG is composed of a square boundary and a polygonal hole described by 20 points. The final PSLG consists of the initial PSLG plus four polygonal holes each one of them composed of 10 points. In incremental Ruppert and in our approach these last four holes are added to the mesh one at a time. Tests have been carried out varying the quality requirement. The first column of table 1 shows the quality measures considered. The following columns show the number of triangles of the initial mesh and those generated by *TRIANGLE*, the incremental Ruppert algorithm, finishing with our algorithm. It can be observed in the results obtained that the number of triangles generated by the incremental Ruppert, or from the scratch are higher than applying our algorithm. Also notice that the percentage of the increment increases as the quality angle gets higher.

Table 1. Number of triangles obtained after the insertion of four holes.

α	**Initial mesh**	**From scratch**	**Incremental Ruppert**	**Our approach**	**Figure**
20°	124	439	421	314	11
25°	170	514	547	413	12
30°	257	717	1275	565	13
32°	350	1745	2771	674	14

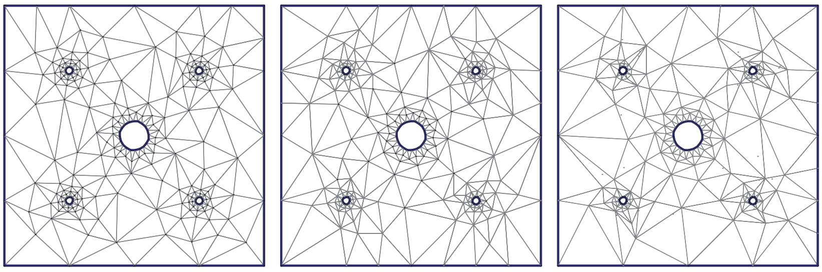

(a) From scratch. (b) Incremental Ruppert. (c) Our approach.

Fig. 11. Results obtained for an angle $\alpha = 20°$.

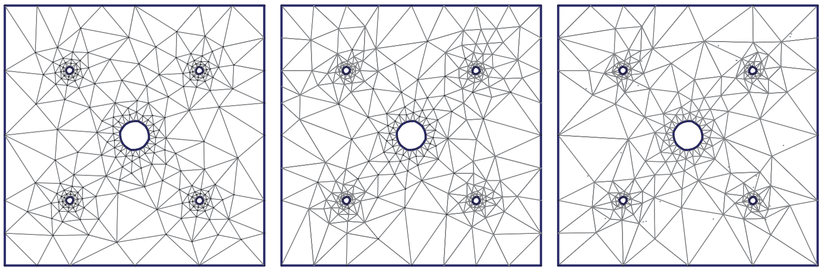

(a) From scratch. (b) Incremental Ruppert. (c) Our approach.

Fig. 12. Results obtained for an angle $\alpha = 25°$.

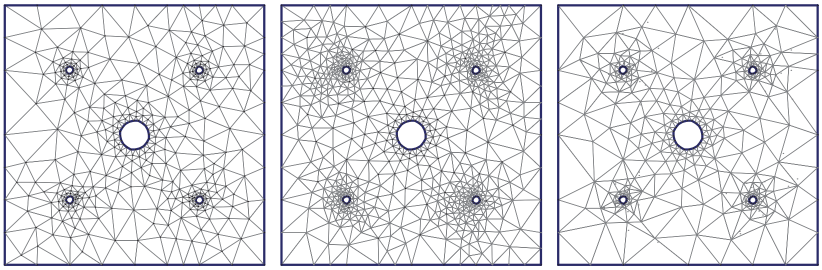

(a) From scratch. (b) Incremental Ruppert. (c) Our approach.

Fig. 13. Results obtained for an angle $\alpha = 30°$.

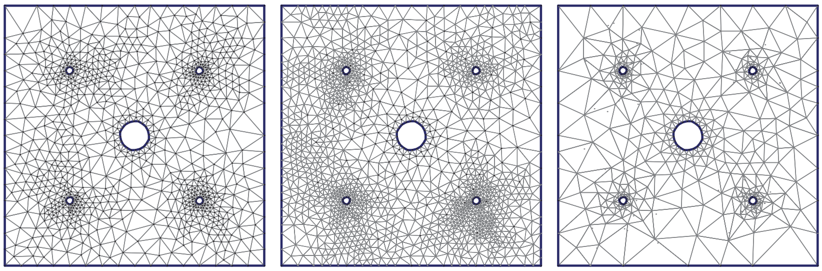

(a) From scratch. (b) Incremental Ruppert. (c) Our approach.

Fig. 14. Results obtained for an angle $\alpha = 32°$.

All previous examples deal with the addition of elements to the initial PSLG, but our algorithm allows us to delete elements from the PSLG. Figure 15 shows an example of segment deletion. Notice that the mesh obtained after the segment deletion is quite similar to the initial mesh.

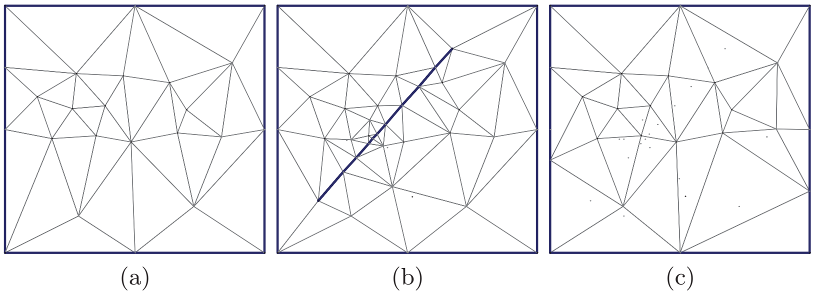


Fig. 15. (a) Initial mesh. (b) A mesh with a segment to be deleted. (c) Resulting mesh after the segment deletion.

Finally, we present some initial results obtained when we use our improvement method for Delaunay refined mesh generation. The PSLG used models the Lake Superior. Figure 16(a) shows the mesh generated taking 34° as quality angle by *TRIANGLE*, and Figure 16(b) the result after applying our method.

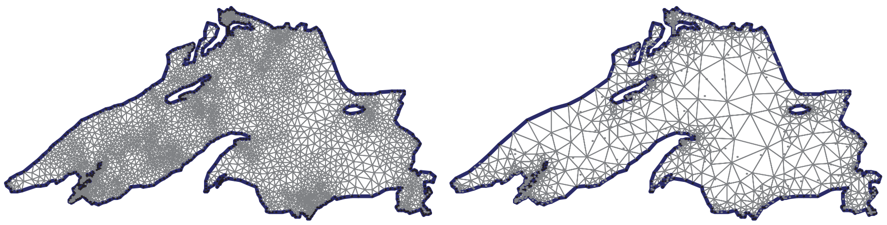

(a) Mesh produced by *TRIANGLE*.
(b) Mesh produced by our approach.

Fig. 16. Results obtained for an angle $\alpha = 34°$.

8 Future Work

Future work includes an exhaustive analysis of the conditions in which the algorithm terminates.

A large experimentation with other input PSLGs is necessary to confirm our initial results in Delaunay refined mesh generation.

Our ultimate goal is to extend our framework towards the modification and generation of three dimensional meshes.

Acknowledgments

The authors wish to thank Frederic Pérez for his help in preparing this paper, and to the reviewers for their comments and suggestions.

References

1. N. Amenta, M. Bern and D. Epstein. Optimal point placement for mesh smoothing. *In SODA:ACM-SIAM Symposium on Discrete Algorithms*, 1997.
2. A. Bowyer. Computing Dirichlet Tessellations. *Computer Journal*, 24:2:162–166, 1981.
3. K. Clarkson and K. Mehlhorn and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3:185–212, 1993.
4. O. Devillers. On deletion in Delaunay triangulation. *15th Annual ACM Symposium on Computational Geometry*, 181–188, 1999.
5. L. Freitag and M. Jones and P. Plassmann. An efficient parallel algorithm for mesh smoothing. *In Proceedings of the Fourth International Meshing Roundtable*, 47–58, 1995.
6. N. Han-Wen and A.-F. van der Stappen. A Delaunay approach to interactive cutting in triangulated surfaces. *In J.D. Boissonnat, J. Burdick, K. Goldbrg & S. Hutchinson (Eds.), Algorithmic Foundations of Robotics V, Springer-Verlag*, 113-129 , 2004.
7. S. Har-Peled and A. Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. *21st Annual ACM Symposium on Computational Geometry (SoCG)*, 228–229, 2005.
8. M. Kallmann and H. Bieri and D. Thalmann. Fully Dynamic Constrained Delaunay Triangulation. *Geometric Modelling For Scientific Visualization - Spring-Verlag*, 2003.
9. C.-L. Lawson. Software for $\mathcal{C}^1$ Surface Interpolation. *Mathematical Software III(John R.Rice, editor)*, 161–194, 1977.
10. G.-L. Miller and S.-E. Pav and N.-J. Walkington. Fully incremental 3d Delaunay mesh generation. *In Proceedings 11th International Meshing Roundtable*, 75–86, 2002.
11. S.-E. Pav. Delaunay Refinement Algorithms. *Department of Mathematical Sciences - Carnegie Mellon University - PhD thesis*, 2003.
12. J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18:3:548–585, 1995.
13. J.-R. Shewchuk. Delaunay Refinement Mesh Generation. *School of Computer Science - Carnegie Mellon University - PhD thesis*, 1997.
14. D.-F. Watson. Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *Computer Journal*, 24:2:167–172, 1981.