# 1B.1

# Interleaving Delaunay Refinement and Optimization for 2D Triangle Mesh Generation

Jane Tournois, Pierre Alliez, and Olivier Devillers

INRIA Sophia-Antipolis, France
firstname.lastname@sophia.inria.fr

**Summary.** We address the problem of generating 2D quality triangle meshes from a set of constraints provided as a planar straight line graph. The algorithm first computes a constrained Delaunay triangulation of the input set of constraints, then interleaves Delaunay refinement and optimization. The refinement stage inserts a subset of the Voronoi vertices and midpoints of constrained edges as Steiner points. The optimization stage optimizes the shape of the triangles through the Lloyd iteration applied to Steiner points both in 1D along constrained edges and in 2D after computing the bounded Voronoi diagram. Our experiments show that the proposed algorithm inserts fewer Steiner points than Delaunay refinement alone, and improves over the mesh quality.

**Key words:** 2D triangle mesh generation, Delaunay refinement, optimization, mesh smoothing, bounded Voronoi diagram.

## 1 Introduction

We consider the problem of generating 2D triangle meshes from a bounded domain and a set of geometric and sizing constraints provided respectively as a planar straight line graph (PSLG) and a sizing function. This problem is mainly motivated by several practical applications, such as numerical simulation of physical phenomena. The latter applications require *quality* meshes, ideally of smallest size. Quality herein refers to the shape and size of the elements: A triangle is said to be good if its angles are bounded from below, and if the length of its longest edge does not exceed the sizing function. The smallest size triangulation, conforming the sizing function, is sought after for efficiency reasons as the number of elements governs the computation time.

Quality triangulation is a well-studied problem, and many meshing strategies have been proposed and studied. Existing algorithms could be roughly classified as being *greedy*, *variational*, or *pliant*. Greedy algorithms commonly perform one local change at a time, such as vertex insertion, until the initial stated goal is satisfied. Variational techniques cast the initial problem into the one of minimizing an energy functional such that low levels of this energy correspond to good solutions for this

problem (reaching a global optimum is in general elusive). A minimizer for this energy may perform global relaxation, i.e., vertex relocations and re-triangulation until convergence. Finally, an algorithm is pliant when it combines both refinement and decimation, possibly interleaved with a relaxation procedure [1, 2].

Although satisfactory bounds have been obtained for the triangle angles using Delaunay refinement algorithms [3], one recurrent question is to try lowering the number of points (so-called *Steiner points*) added by the meshing algorithm. Recall that the Steiner points are inserted either on the constrained edges or inside the domain, to satisfy sizing and quality constraints, as well as to preserve the input constrained edges. At the intuitive level, inserting "just enough" Steiner points requires generating triangles which are everywhere as big as possible, while preserving the sizing constraints. Luckily enough, the triangle which covers the biggest area for a given maximum edge length is the equilateral triangle. In other words, and by using a simple domain covering argument, trying to generate *large and well-shaped triangles* simultaneously serves the goal of lowering the number of Steiner points. Our approach builds upon this observation.

### 1.1 Contributions

This paper proposes to mesh a 2D domain into large and well-shaped triangles by alternating Delaunay refinement and optimization. We pursue the goal of generating large triangles by performing refinement in a multilevel manner, i.e., by inserting a subset of the Voronoi vertices batch-wise at each refinement stage. Each refinement stage is parameterized with a decreasing size, which is computed from the current mesh. Each mesh optimization stage is performed by applying the Lloyd iteration both in 1D along constrained edges, and in 2D inside the domain. To ensure that the constraints are preserved during mesh optimization, the bounded Voronoi diagram (Voronoi diagram with constraints) is computed using a novel robust and effective algorithm. For the sake of efficiency, the optimization stages are applied with increasing accuracy. Our experiments show that, for the required sizing, our algorithm generates in general meshes with fewer Steiner points and better quality than Delaunay refinement alone. We now briefly review Delaunay refinement and mesh optimization techniques.

### 1.2 Delaunay Refinement

A mesh refinement algorithm iteratively inserts well chosen points, so-called *Steiner points* in a given coarse mesh until all constraints are satisfied. One trend in refinement algorithms is to insert as few Steiner points as possible. One popular approach is to take as initial coarse mesh the constrained Delaunay triangulation of the input PSLG, and to refine it. Refinement algorithms of this kind are called *Delaunay Refinement algorithms*. They were pioneered by Chew [4], and later extended by many authors [5, 6]. Delaunay refinement algorithms proceed as follows: During refinement, an edge is said to be encroached if a point of the triangulation (not its endpoints) is on or inside its diametral circle. As long as the current mesh contains

encroached edges, the algorithm inserts their midpoints. It then iteratively inserts the circumcenter of the "worst" triangle of the triangulation (according to size and shape criteria), unless it encroaches an edge. These steps are performed until all triangles are good, i.e., until each triangle has the size and shape criteria satisfied. Shewchuk [6] shows that this algorithm terminates with a finite number of Steiner points and with bounds on the triangles angles (provided the input PSLG does not contain any small angle). Several studies have been made on the insertion order of the Steiner points, and a satisfactory choice is to insert the circumcenter of the worst triangle first.

Another way to grasp the problem of minimizing the number of Steiner points is to define another type of Steiner points than circumcircle centers. The main idea is that if a triangle contains a large angle, Ruppert's algorithm inserts a Steiner point far away from it, in a place which may be irrelevant. Üngör [7] defines a so-called *off-center* as follows. The off-center $off_c$ of a triangle $\triangle(p, q, r)$ of shortest edge $pq$ is defined as the circumcenter of $\triangle(p, q, r)$ if the radius-edge ratio of $\triangle(p, q, r)$ is lower or equal to $\beta$, the angle quality bound. Otherwise, $off_c$ is the point $v$ on the bisector of $pq$, and inside its circumcircle, such that the radius-edge ratio of $\triangle(p, q, off_c)$ equals $\beta$. This technique is shown to insert fewer Steiner points than Ruppert's algorithm.

Chernikov and Chrisochoides generalize and improve over Üngör's off-centers solution [8]. They show that, more generally, any point in the *selection disk* of a bad triangle $\triangle$ can be chosen as a Steiner point. It is shown that, choosing any point inside the *selection disk* as Steiner point eliminates $\triangle$, and that the algorithm terminates. They propose an example of new Steiner point inside the *selection disk* and show that the corresponding refinement algorithm in general inserts fewer points than when using off-centers.

Some other methods combine both insertion and removal of mesh elements (including Steiner points) to obtain higher quality meshes. Such combination was used by Bossen and Heckbert [1] as well as by Coll et al. [2], to cite a few. Finally, Erten and Üngör [9] have recently introduced a new method which first tries relocating the vertices of bad triangles. If the relocations would not improve the mesh, they are canceled and a Steiner point is inserted inside $\triangle$.

### 1.3 Optimization

Many different triangulations covering a given domain and satisfying a set of constraints may exist, each of them of different quality. When high quality meshes are sought after, it is therefore desirable to resort to an optimization procedure so as to optimize a specific quality measure (see [10] for a comprehensive study of quality measures). Two questions now arise: Which criterion should we optimized? By exploiting which degrees of freedom? The optimized criterion can be directly related to the shape and size of the triangles [11], but other criteria have been proposed as well. We refer the reader to [12] for a comprehensive survey of mesh optimization techniques. As the number of degrees of freedom are both continuous and discrete (vertex positions and mesh connectivity), there is often a need for narrowing the space

of possible triangulations. For example, Chen proposes to cast the isotropic meshing problem as an (isotropic) function approximation problem, optimizing within the space of Delaunay triangulations [13].

Eppstein [12] highlights the fact that, in 2D, evenly distributed points lead to well-shaped triangles, assuming an isotropic triangulation such as the Delaunay triangulation. Isotropic meshing can therefore be casted into the problem of isotropic point sampling, which amounts to distribute a set of points on the input domain in as even a manner as possible. One way to distribute a set of points isotropically and in accordance with a sizing function is to apply the Lloyd iteration procedure (described in Section 2.2) over an initial Voronoi diagram. Du et al. [14] have shown how the Lloyd iteration transforms an initial ordinary Voronoi diagram into a centroidal Voronoi diagram, where each generator happens to coincide with the center of mass of its Voronoi cell. Another interesting feature of the Lloyd iteration in our context is that it can be applied to any dimension, i.e., in 1D for the Steiner points inserted on the constrained edges, and in 2D for the Steiner points inserted inside the domain. One drawback of the Lloyd iteration is its slow convergence. Moreover, it only converges to a local minimum of a certain energy functional [15]. Convergence accelerations are possible either by using Newton-Lloyd iterations or by using specific types of multilevel refinements [16, 17].

## 2 Algorithm

We introduce an algorithm which interleaves Delaunay refinement and optimization, in order to generate a triangle mesh satisfying both shape and size properties for each triangle. We sketch our algorithm in pseudo-code as follows:

**Algorithm 1** Meshing algorithm

**Input:** A PSLG domain $\Omega \subset \mathbb{R}^2$.
  Let $T$ be the constrained Delaunay triangulation of $\Omega$.
  **repeat**
    Batch refinement of the triangulation $T$ (Section 2.1).
    **repeat**
      Optimization of $T$ by the Lloyd algorithm (Section 2.2),
    **until** Stopping criterion $S$.
  **until** Refinement does not insert any new Steiner point in $T$.
  **repeat**
    Optimization of $T$ by the Lloyd algorithm,
  **until** Stopping criterion $S'$ stronger than $S$.
**Output:** The final triangle mesh.

The complete algorithm sequence is illustrated by Figure 1. Figure 2 compares the output of our algorithm with a mesh obtained by Delaunay refinement alone. Our method improves over the number of Steiner points as well as on the shape of the elements.

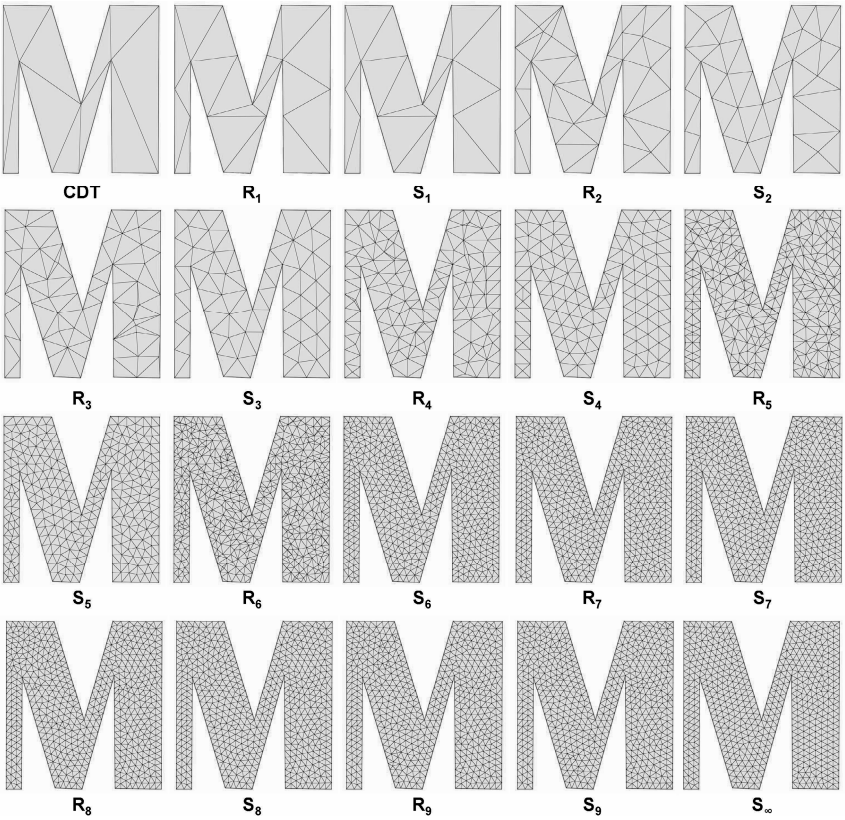


**Fig. 1.** Interleaved Delaunay refinement and optimization with a uniform sizing parameter. In the reading order: The constrained Delaunay triangulation (CDT) of the initial PSLG (13 vertices), then refinement ($R_i$) and smoothing ($S_i$) in alternation. The last step ($S_\infty$) is the final optimization until the stronger stopping criterion. The final mesh contains 645 vertices.

## 2.1 Refinement

The refinement steps are specifically designed to insert "just enough" Steiner points to respect the sizing function, and to provide the optimization steps with good initial solutions. We achieve this goal by inserting batches of Steiner points chosen as Voronoi vertices. The roles are separated: Each refinement step acts on the size of the elements, while each optimization step acts on the shape of the elements (see Section 1.3).

Let us call $\mu(p)$ the desired mesh sizing specified at point $p$. We assume that this sizing function is provided by the user. During meshing we call *sizing ratio* the ratio between the current size of an element and the local desired sizing. An element can be an edge or a triangle, i.e., the sizing ratio is computed as $sr(f) = \frac{\text{longest edge(f)}}{\mu(centroid(f))}$ for a triangle and $sr(e) = \frac{\text{length(e)}}{\mu(centroid(e))}$ for an edge. After meshing, the sizing ratios of all elements must be lower or equal to 1 in order to satisfy the sizing constraints,

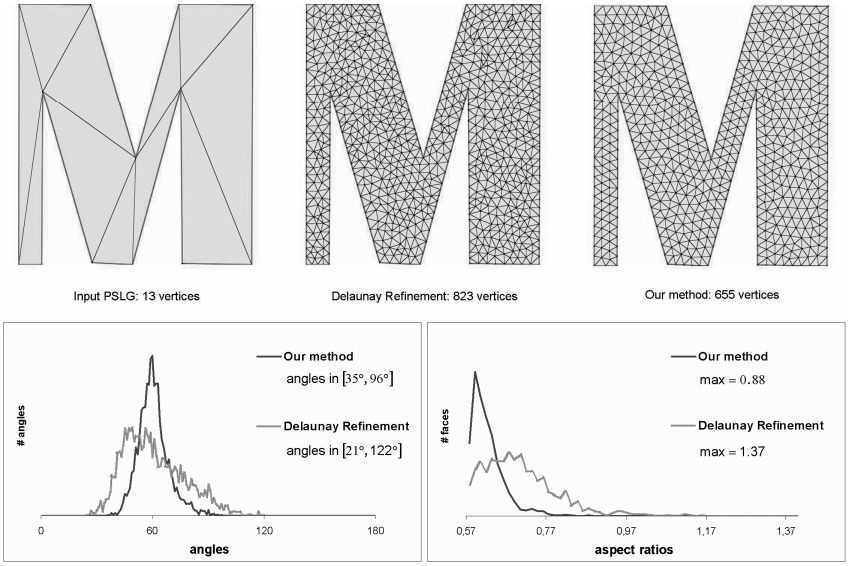


**Fig. 2.** Comparison between Delaunay refinement alone and our algorithm. The sizing function is uniform and equals 0.05. Our method inserts 20% fewer Steiner points than Delaunay refinement does. The distribution of angles is narrower at the end of our algorithm. The angles are within $[35°, 96°]$ with our method, vs $[21°, 122°]$ with Delaunay refinement. The practical upper bound on aspect ratios is lower with our method: $0.88$ vs $1.37$.

and a low number of Steiner points is obtained by keeping the sizing ratios as close to one as possible under these constraints.

Before each refinement step, the algorithm iterates over each triangle and constrained edge of the mesh in order to compute the maximum sizing ratio $sr_{max}$. From this value we compute the current *target sizing ratio* as $sr_{target} = \max(\frac{sr_{max}}{\sqrt{3}}, 1)$, which is then used as a slowly decreasing sizing criterion for refinement. The refinement factor $1/\sqrt{3}$ is justified by the fact that inserting all circumcenters in a triangulation made of equilateral triangles operates a $\sqrt{3} - section$ of the edges (compared to a bisection on the constrained edges). In 2D, the refinement corresponding to the target sizing ratio can therefore not be faster than this value. As we aim at refining the mesh with sizing ratios as uniform as possible, we parameterize the 1D refinement with the same target sizing ratio. The target sizing ratio is thresholded to 1 in order not to insert points that would introduce new triangles or edges with a sizing ratio lower than 1, i.e. smaller than the desired sizing. Hence, at each refinement iteration, the target sizing ratio slowly decreases, until it reaches the value of 1, which corresponds to satisfying the input sizing requirement, via the sizing function $\mu$, for each simplex of the mesh.

Refinement is first performed in 1D along the constraint edges, then in 2D. Call $\mathcal{L}$ the list of Steiner points to be inserted. The algorithm iterates over all triangles and constrained edges and measure their sizing ratio. If it is larger than the current target

sizing ratio $sr_{target}$, we add its midpoint or circumcenter to $\mathcal{L}$, and alter the sizing ratio of its (unprocessed) incident elements in order to avoid over-refinement.

More specifically, and relying on the fact that a local point insertion involves neighboring triangles, we inform the neighboring triangles of a split triangle that they should take upon themselves a part of this refinement. To do so, when a triangle $f$ is processed and eligible for splitting, the quantity $1/3 * (sr_f/\sqrt{3} - sr_{target})$ is added to the sizing ratios of each of its unprocessed neighboring triangles. In this way, we divide between $f$'s neighboring triangles the difference between what we expect to be the sizing ratio of $f$ after splitting ($sr_f/\sqrt{3}$) and the target sizing ratio. Then, these triangle sizing ratios may be increased or decreased, and respectively get more or less likely to be split during the current refinement step. A similar method is used in 1D over the edges, with $1/2$ as refinement factor. When all elements have been visited, all points in $\mathcal{L}$ are inserted to the constrained Delaunay triangulation. The next step is mesh optimization.

## 2.2 Optimization

The mesh optimization step involves the Lloyd iteration applied to the cells of a bounded Voronoi diagram (the pseudo-dual of the constrained Delaunay triangulation).

*Lloyd iteration.*

The Lloyd iteration [18] is a minimizer for the following energy functional:

$$\mathcal{E}(\{z_i\}_{i=1}^N) = \sum_{i=1}^{N} \int_{y \in V_i} \rho(y)||y - z_i||^2 dy,$$

where $\{z_i\}_{i=1}^N$ are a set of generators and $\{V_i\}_{i=1}^N$ the corresponding Voronoi cells. The Lloyd iteration minimizes this energy by alternately moving the generators to the centroid of their Voronoi cells, and recomputing the Voronoi diagram. The centroid $z^*$ of the cell $V$ is defined as:

$$z^* = \frac{\int_V y\rho(y)dy}{\int_V \rho(y)dy},$$

where $\rho$ is a density function defined on the domain $\Omega$. After convergence, the space subdivision obtained is a *centroidal Voronoi tessellation* [14, 15]. As it corresponds to a critical point of the energy $\mathcal{E}$, it is a necessary condition for optimality, in the sense of minimizing $\mathcal{E}$.

We choose to stop the Lloyd iteration when all generators move less than a user-defined distance threshold. We first define the notion of move ratio of a generator $z$ in its Voronoi cell as $m_r(z) = \frac{||z-z^*||}{size(cell(z))}$, where the size of the cell is defined as the longest distance between any pair of points of this cell. The Lloyd iteration is stopped when $\max_{z \in \{z_i\}_{i=1}^N} m_r(z) < p$ ($p$ is typically set to 3% during refinement,

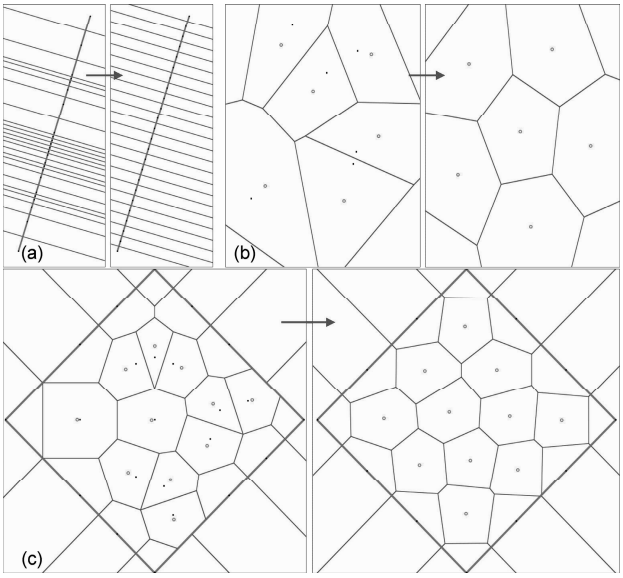


**Fig. 3.** Lloyd iteration with a uniform density. (a) Steiner points randomly inserted on a constrained edge, and after convergence of the Lloyd iteration. (b) Steiner points randomly inserted on a 2D domain, and after convergence of the Lloyd iteration. Sites are filled dots, and centroids are outlined dots (they coincide after convergence). (c) Lloyd iterations applied both in 1D and 2D.

and to 1% for the final relaxation step). In practice, the 1D Lloyd iteration, which allows moving the Steiner points to the centroids of their 1D cells (along the constraints), plays an important role at minimizing the number of Steiner points inserted. We could exhibit an extreme example where combining refinement with 1D Lloyd iteration inserts half of the number of 1D Steiner points inserted by the common recursive edge bisection of the constraints.

Our algorithm applies the Lloyd iteration both in 1D by moving Steiner points along the input constrained edges (Voronoi cells are line segments), and in 2D by moving Steiner points at the centroid of their bounded Voronoi cell. Figure 3 illustrates both 1D and 2D iterations.

*Bounded Voronoi Diagram.*

While the ordinary Voronoi diagram and Delaunay triangulation do not take constraints into account, we wish here to prevent the Voronoi regions to cross over the constraints. To this aim we use a *constrained Delaunay triangulation* [4,19,20] and a variant of its dual, the *bounded Voronoi diagram* (BVD), defined by Seidel [21]. The common duality between Delaunay triangulation and Voronoi diagram links each triangle to its circumcenter. In our context, each triangle may have its circumcenter on the other side of a constrained edge, therefore its corresponding incident Voronoi edge must be clipped by this constraint. The BVD is defined as follows: each cell $V_i$ of a generator $x_i$ is composed by the points of the domain $\Omega$ which are closer to $x_i$

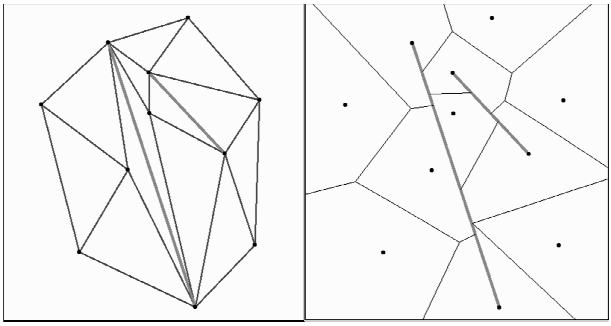

**Fig. 4.** Constrained Delaunay triangulation of a set of points (left) and its pseudo-dual bounded Voronoi diagram (right).

than to any other generator. As for the constrained Delaunay triangulation, the distance incorporates visibility constraints. The distance $d_S(x, y)$ between two points $x$ and $y$ of $\mathbb{R}^2$ is defined as:

$$d_S(x, y) = \begin{cases} ||x - y||_{\mathbb{R}^2} & \text{if } x \text{ “sees” } y, \\ +\infty & \text{otherwise.} \end{cases}$$

In this definition, $x$ “sees” $y$ when no constrained edge intersects the segment $[x, y]$. This visibility notion can be extended to triangles. We will see later how the notion of triangle sight, or symmetrically triangle “blindness”, is important to construct the bounded Voronoi diagram. Figure 4 illustrates a constrained Delaunay triangulation and its pseudo-dual bounded Voronoi diagram. Notice that constructing the standard Voronoi diagram by joining the circumcenters of all pairs of incident triangles would not even form a partition. The notion of triangle blindness is pivotal for constructing the bounded Voronoi diagram.

**Definition 2.1 (Blind triangle)** *A triangle $\triangle$ is said to be* blind *if the triangle and its circumcenter $c$ lie on the two different sides of a constrained edge $E$. Formally, $\triangle$ is* blind *if and only if there exists a constrained edge $E$ such that one can find a point $p$ in $\triangle$ (not an endpoint of $E$), such that the intersection $[p, c] \cap E$ is non empty.*

The BVD construction algorithm initially tags all triangles of the triangulation as being blind or not blind (Algorithm 2). It then constructs each cell of the diagram independently using these tags (Algorithm 3). Finally, all cells are assembled to build the complete bounded Voronoi diagram of a given set of points and constrained edges.

Algorithm 2 tags all triangles of the triangulation as being either *blind* or *non-blind*. In addition, each *blind* triangle stores which constrained edge in the triangulation acts as a visibility obstacle, i.e., which closest edge prevents it to *see* its circumcenter. Notice how the algorithm only needs to iterate over the constrained edges of the triangulation, as all sets of blinded triangles form connected components incident to constrained edges (the corresponding parts of the Voronoi diagram “hidden” by constrained edges are trees rooted from dual rays of these constraints).

**Algorithm 2** Tag blind triangles

```
Input: Constrained Delaunay triangulation cdt.
  Tag all triangles non-blind by default.
  for each Constrained edge e of cdt do
    Create a stack: faces
    for each Adjacent triangle f_e to e tagged non-blind do
      Push f_e into faces
      while triangles is non-empty do
        Pop f from stack triangles
        if f is blinded by e (use P) then
          Tag f as blinded by e
          for each Adjacent triangle f' to f do
            if f' is finite and non-blind
            and the common edge between f and f' is unconstrained then
              Push f' into triangles.
            end if
          end for
        end if
      end while
    end for
  end for
```

We define a robust predicate, called $\mathcal{P}$ in the sequel, to test if a triangle is blinded by a constrained edge. More specifically, $\mathcal{P}$ takes as input a triangle and a segment, and returns a Boolean indicating whether or not the circumcenter of the triangle lies on the same side of the segment than the triangle. The circumcenter is never constructed explicitly in order to obtain a robust tagging of the blind triangles. Each cell of the bounded Voronoi diagram can be constructed by circulating around vertices of the triangulation, and by choosing as cell vertex either circumcenters or intersections of the standard Voronoi edges with the constrained edges. Note that we do not need to construct bounded Voronoi cells incident to input constrained vertices as the latter are constrained and therefore not relocated by the Lloyd iteration. Algorithm 3 describes this construction, and Figure 5 illustrates the construction of a single bounded Voronoi cell. Figure 6 illustrates a bounded Voronoi diagram.

*Quadratures*

The Lloyd iteration requires computing centroids of line segments in 1D and of (possibly bounded) Voronoi cells in 2D. Such computations require quadrature formulas when a variable density function is specified, i.e., when the input sizing function is not uniform. The density function $\rho$ and the sizing function $\mu$ are linked by the following formula [22]: $\mu(x) = \frac{1}{\rho(x)^{d+2}}$, where $d$ is the dimension of the domain. In 2D, we have

$$\mu(x) = \frac{1}{\rho(x)^4}.$$

**Algorithm 3** Construct a BVD cell

**Input:** Unconstrained vertex $z$ of the constrained Delaunay triangulation $cdt$.
Call $P$ the polygon (cell) in construction,
Call $f$ the current triangle and $f_{next}$ the next triangle in the circulation,
Call $\mathcal{L}_{f,f_{next}}$ the line going through the circumcenters of $f$ and $f_{next}$.
**for each** Incident triangle $f$ to $z$ in $cdt$ **do**
  **if** $f$ is tagged non-blind **then**
    Insert the circumcenter of $f$ into $P$.
    **if** $f_{next}$ is blind **then**
      Call $S_{f_{next}}$ the constrained edge blinding $f_{next}$,
      Insert point $\mathcal{L}_{f,f_{next}} \cap S_{f_{next}}$ into $P$.
    **end if**
  **else**
    Call $S_f$ the constrained edge blinding $f$.
    **if** $f_{next}$ is tagged non-blind **then**
      Insert $\mathcal{L}_{f,f_{next}} \cap S_f$ into $P$.
    **else**
      Call $S_{f_{next}}$ the constrained edge blinding $f_{next}$,
      **if** $S_f \neq S_{f_{next}}$ **then**
        Insert $\mathcal{L}_{f,f_{next}} \cap S_f$ and $\mathcal{L}_{f,f_{next}} \cap S_{f_{next}}$ into $P$.
      **end if**
    **end if**
  **end if**
**end for**
**Output:** Bounded Voronoi cell of $z$.

The key idea behind a quadrature is to decompose a simple domain (be it an edge or a triangle in our case) into smaller sub-domains (so-called quadrature primitives) where simple interpolation schemes are devised. The number $N$ of quadrature primitives used for each element allows the user to tune the computation accuracy of the centroids. In practice this number is increased during refinement so that a low precision is used at coarse levels (typically $N = 10$), and a high precision is used at fine levels ($N = 100$ for the final level).

We use the trapezium rule in 1D, with $N$ sub-segments, and the midpoint approximation rule in 2D, with a decomposition of each bounded Voronoi cell into $N$ sub-triangles. More precisely, an initial step triangulates the cell by joining each of its vertices to its generator. The next step recursively bisects the longest edge of these triangles until the number of quadrature triangles reaches $N$. On each quadrature triangle, the midpoint approximation formula is applied:

$$\int_{\triangle} f(x)dx \approx \frac{|\triangle|}{3}(f(x_{12}) + f(x_{23}) + f(x_{13})),$$

where $x_{12}$, $x_{23}$ and $x_{13}$ are the midpoints of a quadrature triangle edges. Finally, we sum the integrals on each quadrature triangle in order to obtain an approximate centroid of the whole bounded Voronoi cell.

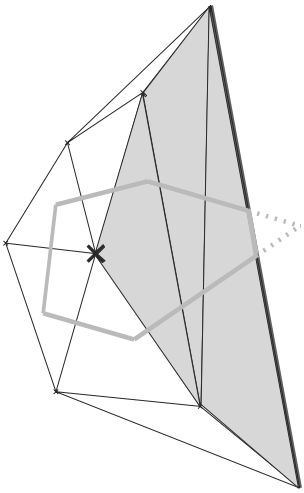

**Fig. 5.** Construction of a cell of the bounded Voronoi diagram. The standard Voronoi diagram is truncated on the constrained edge (colored triangles are blind).

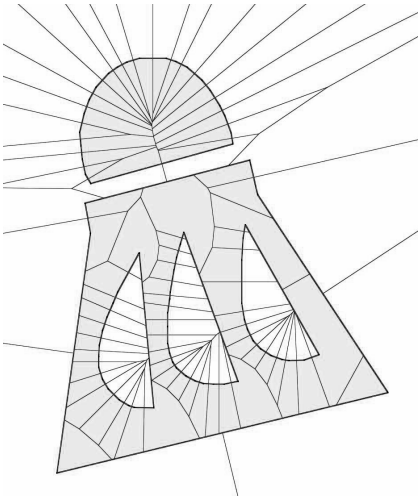

**Fig. 6.** Bounded Voronoi diagram of a PSLG.

## 3 Results

Our algorithm is implemented in C++, using the *Computational Geometry Algorithms Library* CGAL [23]. The Delaunay refinement algorithm that we have used to compare our meshes to is the terminator algorithm defined by Shewchuk [24], implemented as a CGAL package [25].

Figures 7 and 8 show uniform meshes, and compare our algorithm with the standard Delaunay refinement algorithm. The number of Steiner points inserted are respectively 22% and 23% fewer with our method, and the angle distributions are better centered around 60 degrees, as expected. The intervals in which the angles lie are also tighter: $[33°, 99°]$ vs $[20°, 124°]$ for Figure 7, and $[29°, 103°]$ vs $[22°, 127°]$ for Figure 8. The distributions of aspect ratios (circumradii to shortest edge ratios) show that our algorithm produces many more triangles with aspect ratios close to the optimal value, which is $1/\sqrt{3} \approx 0.57$ for an equilateral triangle. This shows that, although our algorithm does not provide theoretical bounds on the triangle angles or aspect ratios, the practical bounds are improved.

Obviously, any mesh generation algorithm which incorporates smoothing or optimization is expected to obtain meshes of higher quality than a greedy algorithm such as Delaunay refinement, at the price of higher computation times (10 times slower for our algorithm in the uniform cases, and close to 100 times slower in the non-uniform cases which involve costly quadratures). In our experiments, interleaving refinement and optimization, similar in spirit to multilevel algorithms, consistently leads to higher quality meshes than refining, then optimizing at the finest level. In addition to the mesh quality obtained, the choice of the Lloyd iteration instead of the

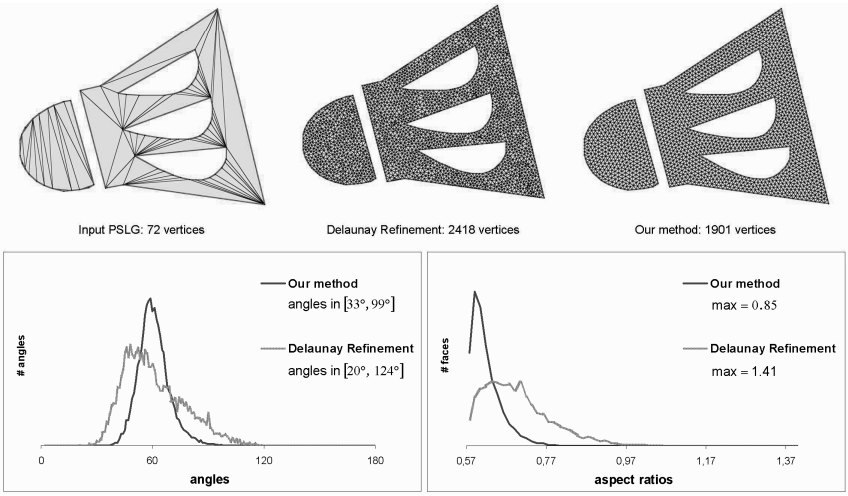


**Fig. 7.** The sizing function is uniform and equals 0.02. Our method inserts 22% fewer Steiner points than Delaunay refinement does. The angles are in $[33°, 99°]$ with our method, vs $[20°, 124°]$ with Delaunay refinement. The maximum aspect ratio is lower: $0.85$ vs $1.41$.

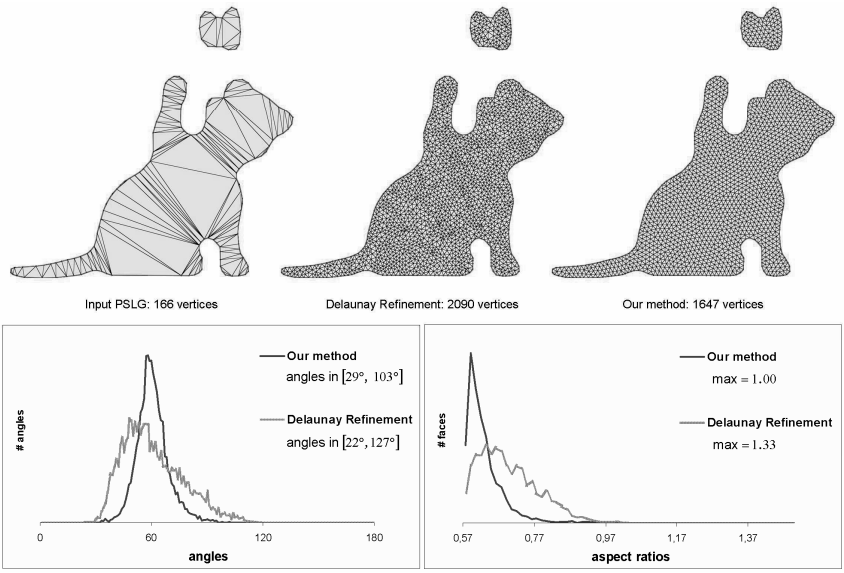


**Fig. 8.** The sizing function is uniform and equals 0.02. Our method inserts 23% fewer Steiner points than Delaunay refinement does. The angles are in $[29°, 103°]$ with our method, vs $[22°, 127°]$ with Delaunay refinement. The maximum aspect ratio is lower: $1.00$ vs $1.33$.

many other mesh optimization techniques is justified by the possibilities to apply it in 1D and 2D, and to take a sizing function as input with variable precision quadratures.

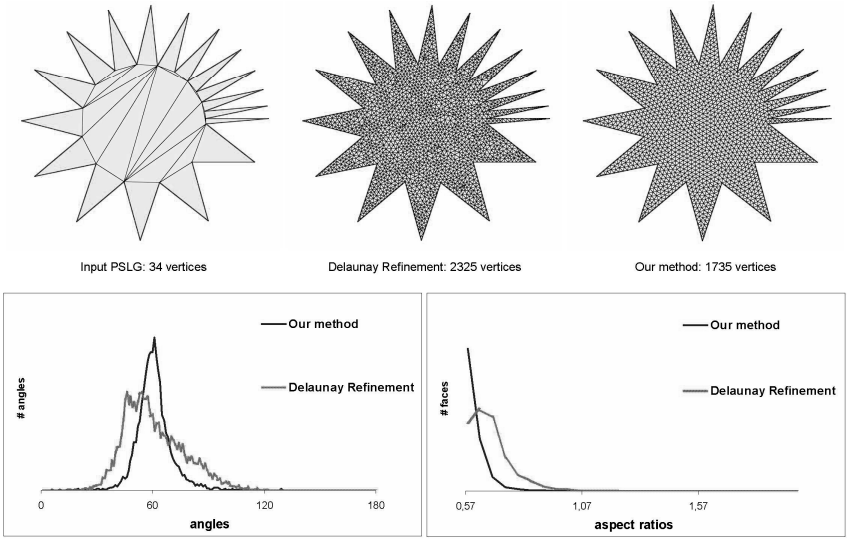


**Fig. 9.** The input PSLG constraints form acute angles from $5°$ to $37°$. The sizing function is uniform and equals 0.02. Our method inserts 26% fewer Steiner points than Delaunay refinement does. The angle distribution shows small angles due to the input. The angles range in $[5°, 128°]$ with our method, vs $[5°, 131°]$ with Delaunay refinement. As expected the upper bound on aspect ratios is similar with both methods (5.73).

One of the main problems in mesh generation is to limit the number of small angles, even when small angles are part of the input [10]. The input PSLG of Figure 9 contains 17 "spin", with inside angles ranging from 5 to 37 degrees. As depicted by the angles distributions, small angles do not hurt our algorithm, which does not produce more small angles than Delaunay refinement does. In addition, our algorithm inserts 26% fewer Steiner points than the terminator algorithm does. The practical upper bound on angles is lower, albeit the improvement is small ($128°$ vs $131°$).

As already discussed in Section 2.2, optimizing the mesh simultaneously in 1D and 2D has an important impact on the number of Steiner points inserted. For example, when the input PSLG contains two long constraints parallel and close to each other, the local improvement in terms of number of 1D Steiner points can be as high as 50%. Intuitively, Delaunay refinement is restricted to recursive constrained edge bisection, which may lead to bisect all constrained edges at the finest level, even if they are slightly longer than the local admissible sizing. In our algorithm, each refinement step is followed by an optimization step, which prevents the next refinement step to over-refine. This behavior is illustrated by Figure 10, where our method inserts 25% fewer Steiner points than Delaunay refinement alone.

For an input domain $\Omega$ and a sizing function $\mu$, one can compute the minimum number of triangles needed to cover $\Omega$. After computing the mean edge length over the domain: $\bar{\mu} = \frac{\int_\Omega \mu(x)dx}{area(\Omega)}$, the minimum number of triangles needed is given by $m_\triangle = \frac{area(\Omega)}{\sqrt{3}/4.\bar{\mu}^2}$, where the denominator is the mean area of an equilateral trian-

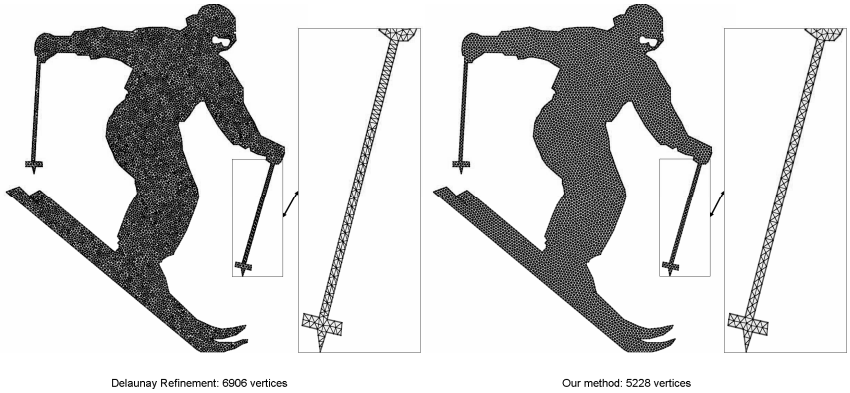


**Fig. 10.** The sizing function is uniform and equals 0.01. The input PSLG contained 248 vertices. Our method inserts 25% fewer Steiner points than Delaunay refinement does. The closeup depicts two constrained edges, parallel and close to each other. Combining 1D and 2D optimization allows inserting fewer Steiner points in this area.

gle inside the domain. In the uniform case, we observe that our method produces meshes with $1.60m_{\triangle}$ triangles, whereas Delaunay refinement produces meshes with $2.20m_{\triangle}$ triangles. These values are very consistent over all the examples we have tested. In the non-uniform case, $\bar{\mu}$ depends too much on the gradation of $\mu$ and on the domain geometry to be able to exhibit a representative average.

Figures 11, 12, 13 and 14 illustrate results from larger inputs and with non-uniform sizing functions, such as the one described by Alliez et al. [26] as: $\mu(x) = \inf_{s \in \partial\Omega}[kd(s,x) + lfs(s)]$, where $\partial\Omega$ is the domain boundary, $d$ the Euclidian distance, $lfs$ the local feature size, and $k$ a constant. This sizing function is shown [26] to be the maximum $k$-Lipschitz function that is smaller or equal to $lfs$ on $\partial\Omega$. In the non-uniform case, our algorithm inserts on average 28% fewer Steiner points than the standard Delaunay refinement.

## 4 Conclusion

We have presented a new 2D triangle mesh generation algorithm, interleaving Delaunay refinement and optimization using the Lloyd iteration. The meshes generated are bound to cover the input constraints provided as a PSLG, and to satisfy sizing constraints provided as a sizing function. Although our algorithm comes with no theoretical bounds on the triangle angles, we show experimental evidences that it produces meshes with fewer Steiner points (25% on average) than the standard Delaunay refinement algorithm, and of better quality. Another contribution of this paper is an efficient and robust algorithm for computing bounded Voronoi diagrams. We plan to integrate the latter into the CGAL library.

The main added value provided by the interleaved refinement and optimization steps in a multilevel manner is to provide the Lloyd optimization step with good

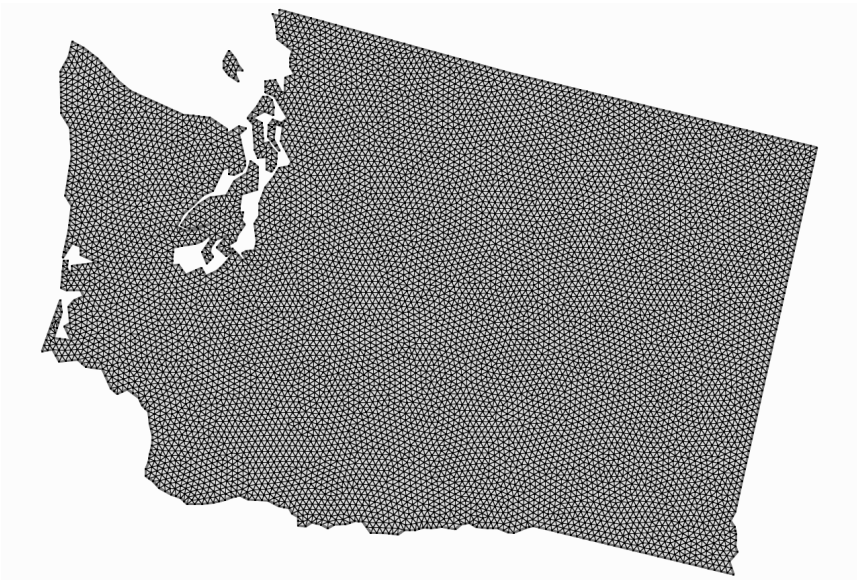

**Fig. 11.** The sizing function is uniform and equals 0.01. The initial PSLG contained 256 vertices. The final mesh contains 9761 vertices.

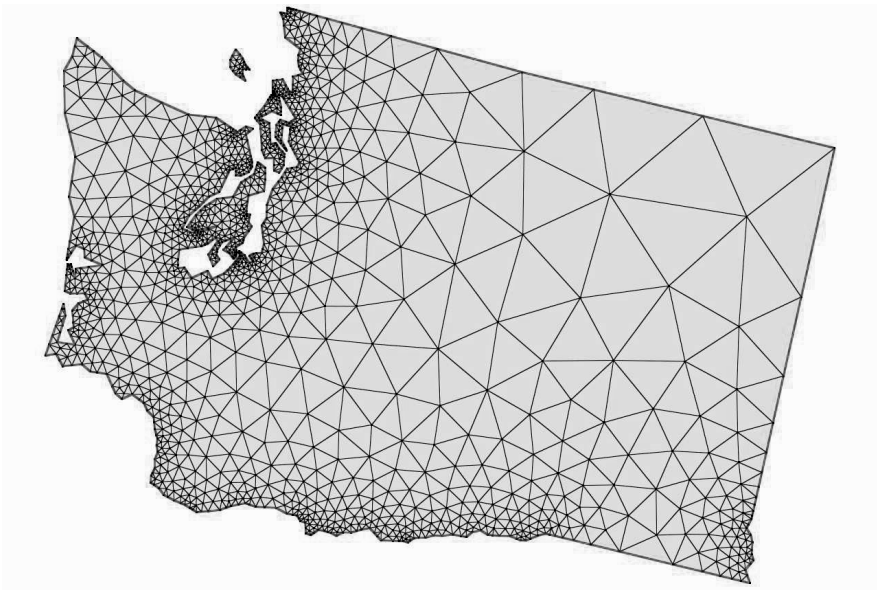

**Fig. 12.** Mesh generated with respect to the sizing function $\mu(x) = \inf_{s \in \partial\Omega}[0.45 * d(s,x) + lfs(s)]$. The initial PSLG contained 256 vertices. The final mesh contains 1440 vertices.

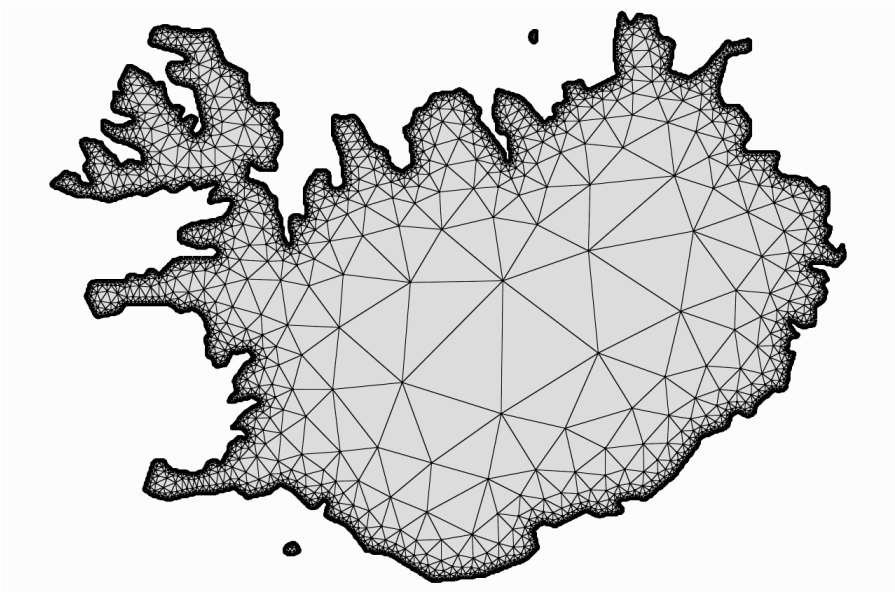

**Fig. 13.** Mesh generated with respect to the sizing function $\mu(x) = \inf_{s \in \partial\Omega}[d(s,x) + lfs(s)]$. The initial PSLG contained 3632 vertices. The final mesh contains 9691 vertices.

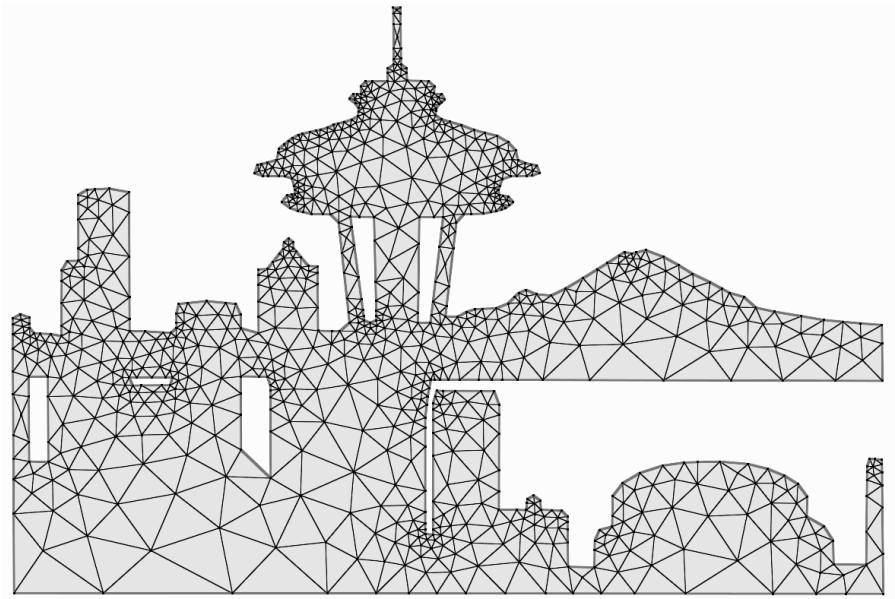

**Fig. 14.** Mesh generated with respect to the sizing function $\mu(x) = \inf_{s \in \partial\Omega}[0.6 * d(s,x) + lfs(s)]$. The initial PSLG contained 205 vertices. The final mesh contains 981 vertices.

initial solutions, and therefore to generate meshes of higher quality than the ones optimized once after refinement. More specifically, such interleaving not only contributes to obtain lower minima for the functional energy described in Section 2.2, but also prevents the Lloyd iteration to perform long range vertex relocations, and therefore to converge slowly. The final number of Steiner points is (non-trivially) related to the decreasing speed of the "target sizing" parameter used by the refinement step. The ultimate goal being to insert just enough Steiner points (and therefore to generate large well-shaped elements), it is desirable to slow down the decreasing of this parameter, especially when approaching the objective sizing function. Although our experiments in this direction were satisfactory in terms of Steiner points added, the computation times substantially increase.

As future work we plan to derive a "hill-climbing" version of our algorithm, where the Lloyd iteration would be modified so as to move each generator *toward* their centroid while staying within some fixed angle bounds. A challenging goal would be to provide theoretical bounds greater than the ones provided by Delaunay refinement. Finally, we plan to extend some of the ideas presented in this paper to isotropic tetrahedral mesh generation with constraints.

## References


1. F.J.BOSSEN and P.S.HECKBERT. A pliant method for anisotropic mesh generation. In *Proceedings of 5th International Meshing Roundtable*, pages 63–74, oct 1996.
2. N.COLL, M.GUERRIERI, and J.A.SELLARÈS. Mesh modification under local domain changes. In *Proceedings of the 15th International Meshing Roundtable*, pages 39–56, 2006.
3. S.E.PAV G.L.MILLER and N.J.WALKINGTON. When and why Delaunay refinement algorithms work. *International Journal of Computational Geometry and Applications (IJCGA)*, 15(1):25–54, feb 2005.
4. P.CHEW. Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.
5. J.RUPPERT. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.
6. J.R.SHEWCHUCK. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22:21–74, 2002.
7. A.ÜNGÖR. Off-centers: A new type of steiner points for computing size-optimal quality-guaranteed delaunay triangulations. In *LATIN: Latin American Symposium on Theoretical Informatics*, 2004.
8. A.N.CHERNIKOV and N.P.CHRISOCHOIDES. Generalized delaunay mesh refinement: From scalar to parallel. In *Proceedings of the 15th International Meshing Roundtable*, pages 563–580, 2006.
9. H.ERTEN and A.ÜNGÖR. Triangulations with locally optimal steiner points. *Eurographics Symposium on Geometry Processing*, 2007.
10. J.R.SHEWCHUCK. What is a good linear element? interpolation, conditioning, and quality measures. *11th International Meshing Roundtable*, pages 115–126, 2002.
11. M.BERN N.AMENTA and D.EPPSTEIN. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30(2):302–322, 1999.
12. D.EPPSTEIN. Global optimization of mesh quality. *Tutorial at the 10th International Meshing Roundtable*, 10, 2001.

13. L.CHEN. Mesh smoothing schemes based on optimal delaunay triangulations. In *Proceedings of 13th International Meshing Roundtable*, pages 109–120, 2004.
14. V.FABER Q.DU and M.GUNZBURGER. Centroidal voronoi tesselations: Applications and algorithms. *SIAM review*, 41(4):637–676, 1999.
15. M.EMELIANENKO Q.DU and L.JU. Convergence of the lloyd algorithm for computing centroidal voronoi tesselations. *SIAM Journal, Numerical Analysis*, 44(1):102–119, 2006.
16. Q.DU and M.EMELIANENKO. Acceleration schemes for computing centroidal Voronoi tessellations. *j-NUM-LIN-ALG-APPL*, 13(2–3):173–192, mar-apr 2006.
17. Q.DU and M.EMELIANENKO. Uniform convergence of a nonlinear energy-based multilevel quantization scheme via centroidal voronoi tessellations. *submitted to SIAM J. Numer. Anal.*, 2006.
18. S.LLOYD. Least square quantization in pcm. *IEEE Trans. Inform. Theory*, 28:129–137, 1982.
19. J.R.SHEWCHUK. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. *Applied Computational Geometry: Towards Geometric Engineering*, 1148:203–222, 1996.
20. M.GUNZBURGER Q.DU and L.JU. Constrained centroidal voronoi tessellations for surfaces. *SIJSSC: SIAM Journal on Scientific and Statistical Computing*, 24, 2003.
21. R.SEIDEL. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. pages 178–191. 1988.
22. Q.DU and D.WANG. Recent progress in robust and quality delaunay mesh generation. *Journal of Computational and Applied Mathematics*, 2005.
23. A.FABRI, G.-J.GIEZEMAN, L.KETTNER, S.SCHIRRA, and S.SCHÖNHERR. On the design of CGAL, a computational geometry algorithms library. *Softw.–Pract. Exp.*, 30(11):1167–1202, 2000.
24. J.R.SHEWCHUK. Mesh generation for domains with small angles. *Proceedings of the sixteenth annual Symposium on Computational Geometry*, pages 1–10, 2000.
25. L.RINEAU. 2d conforming triangulations and meshes. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.3 edition, 2007.
26. M.YVINEC P.ALLIEZ, D.COHEN-STEINER and M.DESBRUN. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617–625, july 2005.