International Journal of Computational Geometry & Applications

PARALLEL DELAUNAY REFINEMENT: ALGORITHMS AND ANALYSES

DANIEL A. SPIELMAN

Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
spielman@math.mit.edu

SHANG-HUA TENG

Department of Computer Science
Boston University
Boston, MA 02215, USA
steng@cs.bu.edu

ALPER ÜNGÖR

Department of Computer Science
Duke University
Durham, NC 27708-0129, USA
ungor@cs.duke.edu

Received (received date)
Revised (revised date)
Communicated by (Name)

In this paper, we analyze the complexity of natural parallelizations of Delaunay refinement methods for mesh generation. The parallelizations employ a simple strategy: at each iteration, they choose a set of "independent" points to insert into the domain, and then update the Delaunay triangulation. We show that such a set of independent points can be constructed efficiently in parallel and that the number of iterations needed is $O(\log^2(L/s))$, where L is the diameter of the domain, and s is the smallest edge in the output mesh. In addition, we show that the insertion of each independent set of points can be realized sequentially by Ruppert's method in two dimensions and Shewchuk's in three dimensions. Therefore, our parallel Delaunay refinement methods provide the same element quality and mesh size guarantees as the sequential algorithms in both two and three dimensions. For quasi-uniform meshes, such as those produced by Chew's method, we show that the number of iterations can be reduced to $O(\log(L/s))$. To the best of our knowledge, these are the first provably polylog(L/s) parallel time Delaunay meshing algorithms that generate well-shaped meshes of size optimal to within a constant.

Keywords: Delaunay refinement, simplicial meshes, parallel algorithms, computational geometry

Contents

1. Introduction

Delaunay refinement is a popular and practical technique for generating well-shaped unstructured meshes.[27,42,49] The first step of a Delaunay refinement algorithm is the construction of a constrained or conforming Delaunay triangulation of the input domain. This initial Delaunay triangulation need not be well-shaped. Delaunay refinement then iteratively adds new points to the domain to improve the quality of the mesh and to make the mesh respect the boundary of the input domain. The points inserted by the Delaunay refinement are often called *Steiner points*. A sequential Delaunay refinement algorithm typically adds one new vertex per iteration, although sometimes one may prefer to insert more than one new vertex at each iteration. Each new point or set of points is chosen from a set of potential candidates — the circumcenters of poorly conditioned simplices (to improve mesh quality) and the diameter-centers of boundary simplices (to conform to the domain boundary). Ruppert[42] was the first to show that the proper application of Delaunay refinement produces well-shaped meshes in two dimensions whose size is within a small constant factor of the best possible. Ruppert's result was then extended to three dimensions by Shewchuk[49] and Li and Teng.[27] Efficient sequential Delaunay refinement software has been developed both in two[42,47] and three dimensions[49]. Chrisochiedes and Nave[12,13] and Okusanya and Peraire[37] developed parallel software using Delaunay refinement, for which they have reported good performance. Recently, Nave *et al.*[35] presented a parallel Delaunay refinement algorithm and showed that it produces quality-guaranteed meshes. The complexity of their algorithm as well as the size of the mesh it outputs remains unanalyzed.

In this paper, we study the parallel complexity of a natural parallelization of Delaunay refinement. One of the main ingredients of our parallel method is a notion of independence among potential candidates for Delaunay insertion at each iteration. Our parallel Delaunay method performs the following steps during each iteration:

(1) Generate an independent set of points for parallel insertion;
(2) Update the Delaunay triangulation in parallel.

Our independent sets have the following properties:

- Their insertion can be realized sequentially by Ruppert's method[42] in 2D and Shewchuk's[49] in 3D. Hence, an algorithm that inserts all their points in parallel will inherit the guarantees of Ruppert's and Shewchuk's methods that the output mesh be quality-guaranteed and size optimal up to a constant.
- The independent sets can be generated efficiently in parallel. In addition, they are "large enough" so that the number of parallel iterations needed is $O(\log^2(L/s))$, where L and s are the diameter of the domain and the smallest edge in the output mesh, respectively.
- When a quasi-uniform mesh is desired as in Chew's method,[8] the number of iterations can be reduced to $O(\log(L/s))$.

We should emphasize here that our analyses focus on the number of parallel iterations of Delaunay refinement. The independence of the new points do not necessarily imply a straightforward parallel insertion scheme at each iteration. There are several existing parallel Delaunay triangulation algorithms that we can employ at each iteration. For example, in two dimensions we can use the divide-and-conquer parallel algorithm developed by Blelloch *et al.* [5] for Delaunay triangulation. Their algorithm uses $O(n \log n)$ work and $O(\log^3 n)$ parallel time. We can alternatively use the randomized parallel algorithms of Reif and Sen [40], or by Amato *et al.* [1], in both two and three dimensions. Both of these randomized parallel Delaunay triangulation algorithms have expected parallel running time $O(\log n)$. Using one of these adds a logarithmic factor to our worst-case total parallel time complexity analysis. To the best of our knowledge, these are the first provably polylog(L/s) parallel time Delaunay meshing algorithms that generate well-shaped meshes of size optimal to within a constant.

1.1. *Motivation and Related Work*

Advances in computer hardware and parallel architecture has given way to the simulation of very large scale engineering and physics problems. Effective use of parallel architecture also requires development of efficient parallel algorithms.[6,25] Significant progress has been made in developing parallel numerical solvers.[22,36,54] However, the research in parallel mesh generation is still in its early stage. There are applications, where adaptive meshing process is reported to take more than half the total simulation time.[28] In such cases, inefficient meshers are the bottleneck. To overcome this, several heuristic solutions addressing parallel mesh generation problem have been suggested.[13,24,28,37,38] In this work, we focus on Delaunay refinement algorithms because both sequential and parallel implementations of Delaunay refinement algorithms seem to produce the best meshes in practice. Our goal is to analyze the parallel complexity of the Delaunay refinement algorithms.

Quadtree-based methods are an alternative to Delaunay refinement. They also generate well-shaped meshes whose size is within a constant factor of the best possible.[3,33] In practice, however, they often generate meshes larger than Delaunay refinement on the same input. The parallel complexity of the quadtree-based methods is nevertheless better understood.[4]

Figures 1 and 2 illustrate the working of quadtree and Delaunay refinement algorithms. Essential part of the quadtree refinement methods involves subdivision of the cells covering the input domain into four equal sized cells in order to adapt the input feature size and maintain the balance of the tree hierarchy.[3,33] Figure 1 shows the status of the quad-tree refinement in four (inherently parallel) consecutive steps. (The input is a point set –not shown explicitly– in a square domain.) Once the refinement is complete, a quad-tree (hence the name) structure, where each interior node has four children, represents the refinement hierarchy. In contrast, the Delaunay refinement hierarchy is not that easy to track. Figure 2 shows

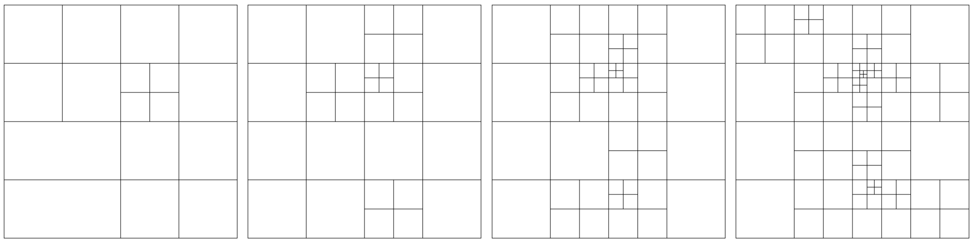


Fig. 1. A sequence of quadtree refinement iterations.

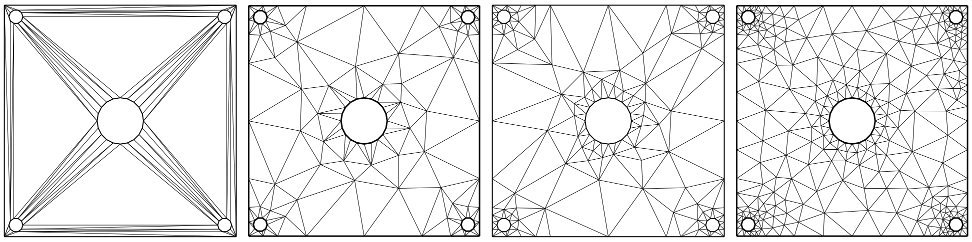


Fig. 2. A sample of iterations of Delaunay refinement.

four intermediate triangulations as points are inserted into a square input domain with holes. This contrast perhaps was the main reason for the time complexity of Delaunay refinement algorithms being an open problem for over a decade.

Several parallel mesh generation algorithms have been developed. On the theoretical extreme, Bern, Eppstein and Teng[4] gave a parallel $O(\log n)$ time algorithm using $K/\log n$ processors to compute a well-shaped quadtree mesh, where K is the final mesh size. There is also a simple level-by-level quadtree-based method that is used in practice[46,51]. One can easily show that this level-by-level based method takes $O(\log(L/s) + K/p)$ parallel time, using p processors[51].

Building upon the work of Bern *et al.*,[4] Miller *et al.*[31] developed a parallel sphere-packing based Delaunay meshing algorithm that generates well-shaped Delaunay meshes of optimal size in $O(\log n)$ parallel time using $K/\log n$ processors. Their method uses a parallel maximal independent set algorithm[29] to directly generate the set of final mesh points, and then constructs the Delaunay mesh using parallel Delaunay triangulation. As this algorithm has not been implemented, we do not know how the meshes it produces will compare.

Another line of research considered the parallelization of Rivara's longest edge refinement method.[41] Jones and Plassman[24] described a parallel edge refinement algorithm with expected polylogarithmic running time. Later, Özturan[38] improved this result giving a polylogarithmic worst case complexity analysis for a similar parallel edge refinement algorithm. Currently, both of these algorithms are limited to two dimensional input. Moreover, these results do not guarantee size optimality and element shape quality for output meshes.

Various parallel Delaunay refinement methods have been implemented and been seen to have good performance.[10,12,13,26,28,35,37] These methods address some important issues such as how to partition the domain so as to minimize the communication cost among the processors. Our new analysis on the number of parallel iterations of Delaunay refinement could potentially provide provable bounds on their parallel complexity.

Our work also helps explain the performance of some sequential implementations of Delaunay refinement, especially those which use a Delaunay triangulator as a black-box. In such situations, it is often desirable to minimize the number of calls to the black-box Delaunay triangulator by inserting multiple points at each iteration. Our bounds on the number of iterations provide a bound on the number of calls to the Delaunay triangulator. We should also note that, our analysis framework, already presented in the proceedings of the International Meshing Roundtable[52] and also in the third author's Ph.D. thesis[55], motivated the recent work of Miller on the timing analysis of the sequential Delaunay refinement.[30]

1.2. *Outline*

In Section 2, we review basic concepts in Delaunay mesh generation.[2,16,53] In Section 3, we discuss the sequential Delaunay refinement method.[8,42] In Section 4, we present our parallel algorithms and their analyses in two dimensions. We discuss the extension to three dimensions in Section 5. We conclude in Section 6 with directions for future research.

2. Preliminaries

2.1. *Input Domain*

In two dimensions, the input domain Ω is represented as a *planar straight line graph* (PSLG)[42] — a proper planar drawing in which each edge is mapped to a straight line segment between its two endpoints. The segments express the *boundaries* of Ω and the endpoints are the *vertices* of Ω. The vertices and boundary segments of Ω will be referred to as *input features* of Ω. A vertex is incident to a segment if it is one of the endpoints of the segment. Two segments are incident if they share a common vertex. In general, if the domain is given as a collection of vertices only, then the boundary of its convex hull is taken to be the boundary of the input.

Miller *et al.*[32] presented a natural extension of PSLGs, called *piecewise linear complexes* (PLCs), to describe domains in three and higher dimensions. In three dimensions, the domain Ω is a collection of vertices, segments, and facets where (i) all lower dimensional elements on the boundary of an element in Ω also belongs to Ω, and (ii) if any two elements intersect, then their intersection is a lower dimensional element in Ω. In other words, a PLC in d dimensions is a cell complex with polyhedral cells from 0 to d dimensions.

2.2. *Delaunay Triangulation*

Let P be a point set in $\mathbb{R}^d$. A simplex τ formed by a subset of P points is a *Delaunay simplex* if there exists a circumsphere of τ whose interior does not contain any points in P. The Delaunay triangulation of P, denoted $Del(P)$, is a PLC that contains all Delaunay simplices. If the points are in general position, that is, if no $d+2$ points in P are co-spherical, then $Del(P)$ is a simplicial complex.

The Delaunay triangulation of a point set can be constructed in $O(n \log n)$ time in two dimensions[14,23,45,39,20] and in $O(n^{\lceil d/2 \rceil})$ time in d dimensions.[14,44] Also, Fortune[21] presents a nice survey of these algorithms.

One way to obtain a triangulation that conforms to the boundary of a PSLG domain is to use a *Constrained Delaunay triangulation*. Let P be the set of vertices of a PSLG Ω. Two points p and q in P are said to be visible from each other if the line segment pq does not intersect the interior of any segment in Ω. Three points form a constrained Delaunay triangle if the interior of their circumcircle contains no point from P that is visible from all three points. The union of all constrained Delaunay triangles forms a *constrained Delaunay triangulation* $CDT(\Omega)$. Chew[11] developed an algorithm for computing constrained Delaunay triangulations.

A Delaunay triangulation T of input and Steiner points is a *conforming Delaunay triangulation* of a PLC Ω if every face of Ω is a union of faces of T. In two dimensions, Edelsbrunner and Tan[19] proved that $O(n^3)$ additional points are sufficient to generate a conforming triangulation of a PSLG of complexity n. Another two-dimensional solution proposed by Saalfeld[43] is extended to three dimensions by Murphy *et al.*[34] and Cohen-Steiner *et al.*[15] However, it remains open whether the size of their output is polynomial in the input size or local feature size. (The definition of local feature size will be given in Section 3.) Moreover, these methods do not guarantee the quality of the elements, making them less appealing for mesh generation. When the angle between the faces of a PLC is bounded from below, say for example by $\pi/2$, then one can apply Delaunay refinement to generate quality-guaranteed conforming triangulations whose size is close to optimal both in two[8,42] and three dimensions.[9,27,49]

3. Sequential Delaunay Refinement in Two Dimensions

In this section, we recall Ruppert's and Chew's algorithms for constructing Delaunay meshes of PLSGs in 2D. Following Ruppert,[42] we assume that the angle between two adjacent input segments is at least $\pi/2$. Boundary treatments that relax this assumption are discussed in the literature.[42,50]

In the process of Delaunay refinement, one could either maintain a constrained Delaunay triangulation, or one just keeps track of the set of input segments that are not respected. The first approach does not extend to three dimensions because, in 3D, some PLCs do not have a constrained Delaunay triangulation. We therefore use the second approach.

At each iteration, we choose a new point for insertion from a set of candidate points. There are two kinds of candidate points: (1) the circumcenters of existing triangles, and (2) the midpoints of existing boundary segments.

Let the diametral circle of a segment be the circle whose diameter is the segment. A point is said to *encroach* a segment if it is inside the segment's diametral circle. (See Figure 3.) At iteration i, the circumcenter of a triangle is a *potential candidate* for insertion if the triangle is *poorly shaped*. For example, in Ruppert's algorithm, a triangle is considered poorly shaped if the ratio of its circumradius to the length of its shortest side is larger than a pre-specified constant $\beta_R \geq \sqrt{2}$. Let $\dot{\mathcal{C}}^{(i)}$ denote the set of all potential candidate circumcenters that do not encroach any segment.

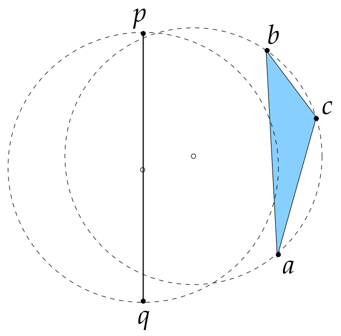


Fig. 3. Circumcenter of triangle abc encroaches the segment pq.

Let $\mathcal{C}^{(i)}$ denote their corresponding circumcircles. Similarly, let $\dot{\mathcal{B}}^{(i)}$ denote the set of all potential candidate circumcenters that do encroach some segment. Let $\mathcal{B}^{(i)}$ denote their corresponding circumcircles.

The midpoint of a boundary segment is a *candidate* for insertion if (1) the segment is not part of the current Delaunay triangulation, that is, its diametral circle is encroached by some existing mesh points, or (2) its segment is encroached by a circumcenter in $\dot{\mathcal{B}}$. In the latter case, this potential circumcenter candidate is *rejected* from insertion. Let $\dot{\mathcal{D}}_T^{(i)}$ be all midpoint candidates of type (1) and let $\dot{\mathcal{D}}_{\mathcal{B}}^{(i)}$ be all midpoint candidates of type (2).

Algorithm 1 Sequential Delaunay Refinement

Input: A PSLG domain Ω in $\mathbb{R}^2$
  Let T be the Delaunay triangulation of the vertices of Ω.
  Let $i = 0$ and compute $\dot{\mathcal{B}}^{(i)}$, $\dot{\mathcal{C}}^{(i)}$, $\dot{\mathcal{D}}_T^{(i)}$, and $\dot{\mathcal{D}}_{\mathcal{B}}^{(i)}$;
  while $\dot{\mathcal{C}}^{(i)} \cup \dot{\mathcal{D}}_T^{(i)} \cup \dot{\mathcal{D}}_{\mathcal{B}}^{(i)}$ is not empty **do**
    Choose a point q from $\dot{\mathcal{C}}^{(i)} \cup \dot{\mathcal{D}}_T^{(i)} \cup \dot{\mathcal{D}}_{\mathcal{B}}^{(i)}$ and insert q into the triangulation. If q is a midpoint of a segment s, remove s from the segment list and replace it with two segments from q to each endpoint of s;
    Update the Delaunay triangulation T; $i = i + 1$;
    Compute $\dot{\mathcal{B}}^{(i)}$, $\dot{\mathcal{C}}^{(i)}$, $\dot{\mathcal{D}}_T^{(i)}$, and $\dot{\mathcal{D}}_{\mathcal{B}}^{(i)}$.
  end while

If a quasi-uniform mesh, such as that produced by Chew's method,[8] is desired, then we use the following notion of poorly shaped triangle: A triangle is *poorly-shaped* if the ratio of its circumradius to the length of the shortest edge in the current Delaunay triangulation T is more than a pre-specified constant $\beta_C \geq \sqrt{2}$.

Figure 4 shows the output of the Delaunay refinement illustrating the differ-

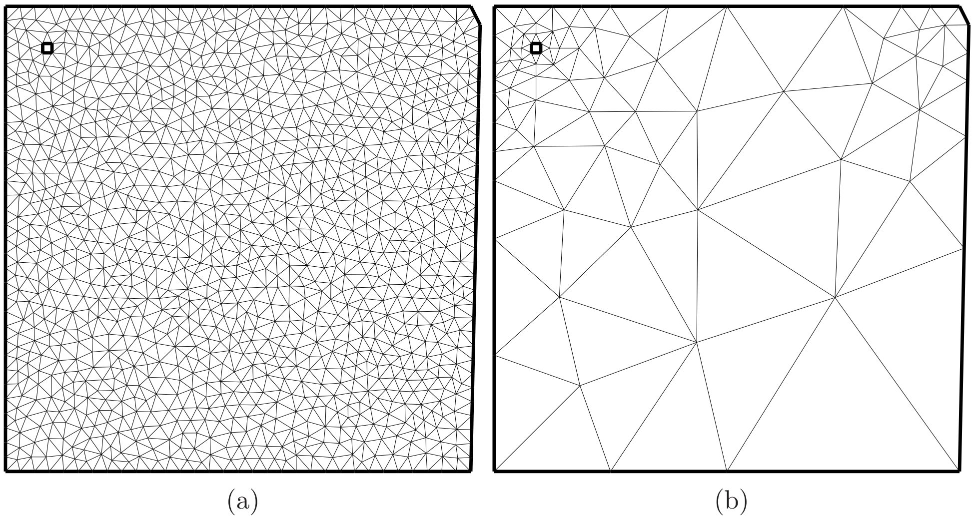


Fig. 4. The output of (a) Chew's and (b) Ruppert's algorithm on the same input. Both of these meshes have minimum angle $> 29°$. The first mesh has 2246 and the second has 131 elements.

ence between Chew's and Ruppert's refinement. We call these two variants of the Delaunay refinement algorithm *Chew's algorithm* and *Ruppert's algorithm*.

In their original papers,[8,42] Chew and Ruppert presented their Delaunay refinement algorithms as particular variations of Algorithm 1 —they specified how to choose the next point at each iteration from the set of candidates. In this paper, we will consider the following variation of Algorithm 1 which is more aggressive in adding boundary points — we choose this variation to parallelize because its analysis is relatively simpler to present.

In this variation, $\mathcal{B}^{(i)}$, $\mathcal{C}^{(i)}$, and $\mathcal{D}_T^{(i)}$ are the same as in Algorithm 1. The set $\mathcal{D}_{\mathcal{B}}^{(i)}$ is built incrementally. At iteration i, we compute $\mathcal{B}^{(i)}$ first and let $\mathcal{D}^{(i)}$ be the set of diametral circles that are encroached by some circumcenters of $\mathcal{B}^{(i)}$. We then set $\mathcal{D}_{\mathcal{B}}^{(i)} = \mathcal{D}_{\mathcal{B}}^{(i-1)} \cup \mathcal{D}^{(i)}$.

In other words, if a segment is encroached by a circumcenter of a poorly-shaped Delaunay triangle, its midpoint will be added to the set of candidate midpoints and remains candidate thereafter. This is in contrast with Algorithm 1, in which an encroached midpoint is added to the set of candidate midpoints only for the next iteration. If another candidate is chosen that is in a circumcircle whose center encroaches the segment, the circumcircle will no longer be in the Delaunay triangulation at the end of the iteration, and hence the segment might not be encroached in the triangulation at the end of the iteration. So, its midpoint might not be a candidate in future iterations.

Assuming that the angle between two adjacent input segments is at least $\pi/2$, Chew's algorithm terminates with well-shaped quasi-uniform meshes, while Ruppert's algorithm[42] terminates with a well-shaped Delaunay mesh of the input domain whose elements adapt to the local geometry of the domain. The number of triangles in the mesh generated by Ruppert's algorithm is asymptotically optimal up to a constant. The proofs of Ruppert's[42] and Chew's[8] that their algorithms terminate with a well-shaped mesh of size within a constant factor of optimal can be easily extended to our variation of Algorithm 1 discussed above. We refer interested readers to the original work by Ruppert[42] and Shewchuk.[48] Here we give a high level argument and introduce an important concept that will be used in Section 4.2.3 for preprocessing an input domain in parallel.

Given a domain Ω, the *local feature size* of each point x in Ω, denoted by $\mathrm{lfs}_\Omega(x)$, is the radius of the smallest disk centered at x that touches two non-incident input features. Ruppert showed that every Delaunay triangle in the final mesh is well-shaped and that the length of the longest edge in each Delaunay triangle is within a constant factor of $\mathrm{lfs}_\Omega(x)$ for each x in the interior of the triangle.

Suppose M is a mesh generated by our variation of Algorithm 1. Let Ω' be the domain obtained from Ω by adding to Ω all mesh points in M that are on the boundary segments of Ω. Then it is easy to show that (i) for all x in Ω, $\mathrm{lfs}_\Omega(x)$ and $\mathrm{lfs}_{\Omega'}(x)$ are within a small constant factor of each other; and (ii) M can be obtained by applying Ruppert's (or Chew's) variations of Algorithm 1 to Ω'. Therefore, the mesh produced by our variation of Algorithm 1 has size within a small constant factor of the one generated by Ruppert's (or Chew's) refinement method.

4. Parallel Delaunay Refinement in Two Dimensions

To better illustrate our analysis of parallel Delaunay refinement, we first focus on the case in which the input is a periodic point set.[7,16] Note that in such case, Ω and Ω', as described in the previous section, are same. We will then extend some of our results to produce boundary conforming meshes when the input domain is a PSLG. We would like to mention here that conforming to the boundary is a very challenging issue in mesh generation. Sequential Delaunay refinement methods need to make certain assumptions in order to handle the boundary.[42,48,27,50] We will make similar assumptions in Section 4.2. Also, it is worth to note that some of the pioneering provably good mesh generation work, such as sliver removal algorithms, are first studied on periodic point sets.[7,18]

4.1. *Parallel Delaunay Refinement for Periodic Point Sets*

If P is a finite set of points in the half open unit square $[0,1)^2$ and $\mathbb{Z}^2$ is the two dimensional integer grid, then $S = P + \mathbb{Z}^2$ is a periodic point set.[16] The periodic set S contains all points $p + v$, where $p \in P$ and v is an integer vector. The Delaunay triangulation of a periodic point set is also periodic. As P is contained in the unit square, the diameter of P is $L \leq \sqrt{2}$.

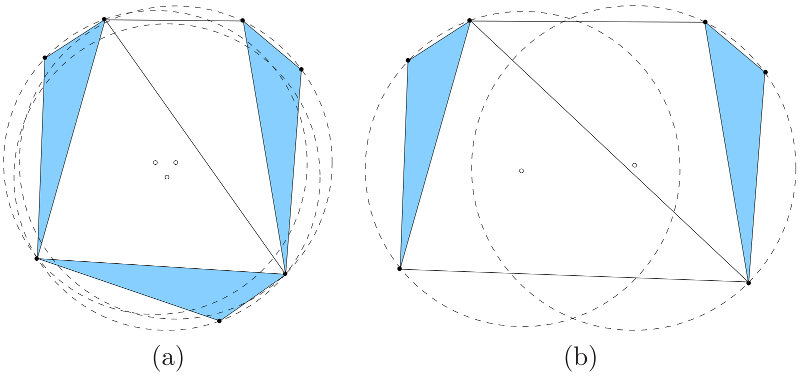


Fig. 5. Circumcenters of bad triangles (shaded) are in conflict. (a) they are too close to each other; (b) they are not too close but can not be both inserted by a sequential algorithm.

4.1.1. *A generic parallel algorithm*

For a periodic point set, the only candidates for insertion are the circumcenters of poorly shaped triangles. Naturally, when parallelizing a Delaunay refinement algorithm, at each iteration we try to insert as many Steiner points as possible. However, some points in $\dot{\mathcal{C}}$ can be arbitrarily close to each other (see Figure 5 (a)), hence neither a sequential nor a parallel refinement algorithm can insert them all without sacrificing the termination guarantee. Furthermore, we would like to insert a set of points that has a sequential realization, i.e. there exist a provably good sequential algorithm that inserts the same set of points. The insertion of the two candidate circumcenters in Figure 5 (b), for instance, is not a threat for termination because no edge smaller than the feature size is introduced. However, insertion of one would eliminate the other from being a candidate, in a sequential execution of the Delaunay refinement algorithm described in Section 3. We need a rule for choosing a large subset of the candidates with a sequential realization. Our rule is derived from the following notion of *independence* among candidates.

Definition 1 (Independence). Two circumcenters $\dot{c}$ and $\dot{c}'$ (and also the corresponding circles c and c') are said to *conflict* if both c and c' contain each other's center. Otherwise, $\dot{c}$ and $\dot{c}'$ (respectively c and c') are said to be *independent*.

If two candidates conflict, at most one of them can be inserted. Our rule is to insert a maximal independent set of candidates at each iteration. We will show that if an algorithm follows this rule, then it will terminate after a polylogarithmic number of rounds.

In the next few subsections, we will discuss how to generate the independent sets used by the algorithm. But first, we prove that regardless of how one chooses the independent set, our parallel algorithm (Algorithm 2) can be sequentialized. This implies that the algorithm inherits the guarantee of its sequential counterpart that

Algorithm 2 Generic Parallel Delaunay Refinement

Input: A periodic point set P in $\mathbb{R}^2$
  Let T be the Delaunay triangulation of P
  Compute $\dot{\mathcal{C}}$, the set of all candidate circumcenters in T
  while $\dot{\mathcal{C}}$ is not empty **do**
    Let $\mathcal{I}$ be an independent subset of $\dot{\mathcal{C}}$
    Update Delaunay triangulation inserting all points in $\mathcal{I}$ in parallel
    Update $\dot{\mathcal{C}}$
  end while

it generates a well-shaped mesh of size that is within a constant factor of optimal.

Theorem 1. *Suppose M is a mesh produced by an execution of the Generic Parallel Delaunay Refinement algorithm. Then M can be obtained by some execution of one of the sequential Delaunay refinement algorithms discussed in Section 3.*

Proof. Let $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$ be the sets of vertices inserted by the parallel algorithm above at iterations $1, \ldots, k$, respectively. We describe a sequential execution that inserts all the points in $\mathcal{I}_i$ before any point of $\mathcal{I}_j$ for $i < j$. In other words, first all the points in set $\mathcal{I}_1$ are inserted sequentially, then all the points in set $\mathcal{I}_2$, and so on. To determine the order within each maximal independent set, we will use a dependency graph. For any two circumcenters $\dot{a}, \dot{b} \in \mathcal{I}_i$, $\dot{a}$ has to wait for the insertion of $\dot{b}$ if $\dot{a}$ is inside the circle b. In the dependency graph we put an edge from $\dot{a}$ to $\dot{b}$. Because each edge is directed from a smaller circle to a larger circle, this dependency graph is acyclic. Topological sorting of the vertices of the dependency graph gives us a valid sequential insertion within each $\mathcal{I}_i$.

Furthermore, observe that in any sequential execution, a point $p \in \mathcal{I}_i$ can not eliminate a point $q \in \mathcal{I}_j$ for any $i < j$. Otherwise, q would not be inserted in the j^{th} iteration of the parallel execution.

Finally, the parallel and sequential executions terminate after inserting exactly the same set of points. An extra element in the insertion set of one would indicate the execution of the other one is not terminated, as this implies that there exists a bad element in one execution but not the other. Since the same point set is inserted effectively in the same order, the output mesh is the same. □

To minimize the number of iterations, intuitively, we should choose a maximal independent set of candidates at each iteration. In Section 4.1.3, we will give a geometric algorithm that computes a maximal independent set of candidates efficiently in parallel. Our algorithm makes use of the following observation.

Lemma 1. *Suppose c_a and c_b are two conflicting circumcircles at iteration i, and let r_a and r_b be their circumradii. Then $r_b/2 < r_a < 2r_b$.*

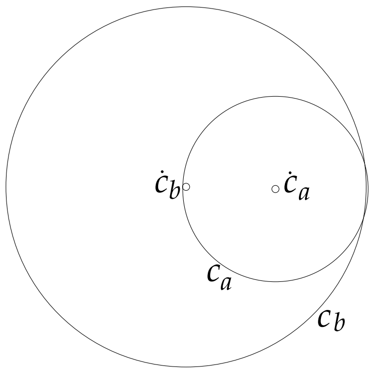


Fig. 6. The larger of two conflicting circumcenters can be at most twice as large as the smaller.

Proof. Circumcircle c_a contains $\dot{c}_b$. As some point on c_a lies outside c_b, the diameter of c_a is greater than the radius of c_b. Thus, $r_a > r_b/2$. A symmetric argument implies $2r_b > r_a$. □

4.1.2. *Parallelizing Chew's Refinement*

In this section, we present a parallel version of Chew's refinement algorithm and analyze its complexity. We define the "bad" triangles as in the sequential version. Our algorithm is a derived from Algorithm 2, the generic parallel Delaunay refinement. We use a *maximal* independent subset of the candidate points at each iteration. We will refer to this specific version as the *parallel Chew's refinement algorithm*.

At any iteration we consider the largest Delaunay circle in the triangulation. We prove a technical lemma which states that the radius of the largest Delaunay circle is reduced by a constant factor after a constant number of iterations. Let r_i denote the radius of the largest Delaunay circle at the end of iteration i.

Lemma 2. *For all i, $r_{i+98} \leq 3r_i/4$.*

Proof. Let c_i be the largest circle in the i^{th} iteration with center $\dot{c}_i$ and radius r_i. Suppose to the contrary that $r_k \geq \beta_C \bar{\boldsymbol{s}}$ ($\bar{\boldsymbol{s}}$ is the smallest feature size in the input) and $r_k > 3r_i/4$ for some i and k with $k - i = 98$. The former inequality simply states that there are still bad elements in iteration k and hence the algorithm has not terminated yet. We will first prove that for each iteration j, where $i < j \leq k$, there exists a bad triangle with circumcirle c'_j, circumcenter $\dot{c}'_j$, and circumradius r'_j in the vicinity of $\dot{c}_k$. Specifically, we will show that (i) $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$ and (ii) $r'_j \geq 3r_i/4$.

Since we never remove vertices during the parallel refinement process, an empty circle in iteration k is also empty in an earlier iteration j. Indeed, vertices p_k, q_k and t_k that c_k is passing through might not exist in iteration j. (See Figure 7 (a).)

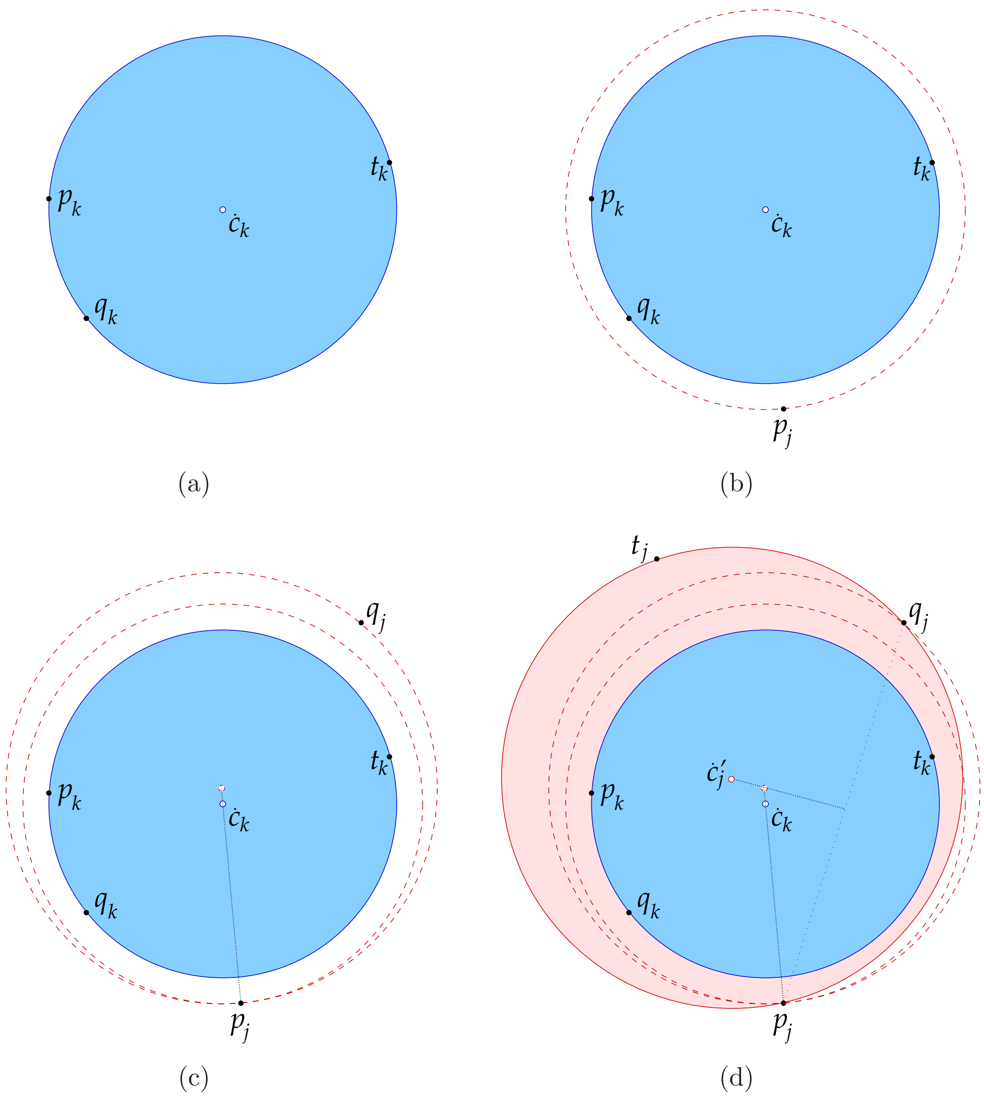


Fig. 7. Dilate the largest Delaunay circle of iteration k (a) until it touches a vertex p_j existed in iteration j (b). Grow the circle further so that it touches two more vertices q_j and t_j existed in iteration j ((c) and (d)).

We dilate c_k in three steps, to locate a larger circumcircle c'_j passing through points p_j, q_j and t_j that all exist in iteration j.

(i) Grow the circle c_k, keeping $\dot{c}_k$ fixed, until it touches a vertex p_j (Figure 7 (b)).

(ii) Further grow the circle, keeping the tangency at p_j fixed and moving the center away from p_j (along the ray $\overrightarrow{p_j \dot{c}_k}$), until it touches a vertex q_j (Figure 7 (c)).

(iii) Further grow the circle, keeping the touch points at p_j and q_j fixed, and moving the center away from $p_j q_j$ (along the bisector of $p_j q_j$), until it touches a vertex t_j (Figure 7 (d)).

The resulting circle c'_j is a circumcircle of a Delaunay triangle $p_j q_j t_j$ at iteration j. Moreover, $p_j q_j t_j$ is a bad triangle because its circumradius r'_j is at least r_k, which by assumption is at least $\beta_C \bar{\boldsymbol{s}}$. Thus, the circumcenter $\dot{c}'_j$ is a candidate vertex in iteration j. By construction, $r'_j \geq r_k \geq 3r_i/4$. We next prove that $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$.

Consider the triangle $p_j \dot{c}_k \dot{c}'_j$, which is non-acute at vertex $\dot{c}_k$. Let $x = |\dot{c}_k p_j|$ and $y = |\dot{c}_k \dot{c}'_j|$. See Figure 8. By cosine theorem, $r_j^2 = x^2 + y^2 - 2xy\cos(|\angle \dot{c}'_j \dot{c}_k p_j|)$. Since $|\angle \dot{c}'_j \dot{c}_k p_j|$ is non-acute $r_j^2 \geq x^2 + y^2$. Also, $x \geq r_k > 3r_i/4$ and thus $r_j < x + r_i/4$; otherwise, $r'_j > r_i$ which contradicts the monotonicity of the radius of the largest Delaunay triangle through refinement. So we can write, $(x + r_i/4)^2 \geq x^2 + y^2$. By simplification, $xr_i/2 + r_i^2/16 \geq y^2$. Substituting $x \leq r_i$, we derive $9r_i^2/16 \geq y^2$. Hence, $y = ||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$. This implies that $\dot{c}'_j$ is rejected in iteration j of the algorithm; otherwise, c_k would not be empty in iteration k.

At each iteration we select a maximal independent subset of the candidates. By Lemma 1, there is a center in conflict with $\dot{c}_j$. Let this center be $\dot{c}''_j$. Again by Lemma 1, $\dot{c}''_j$ is chosen by the parallel algorithm for insertion and the radius of c''_j

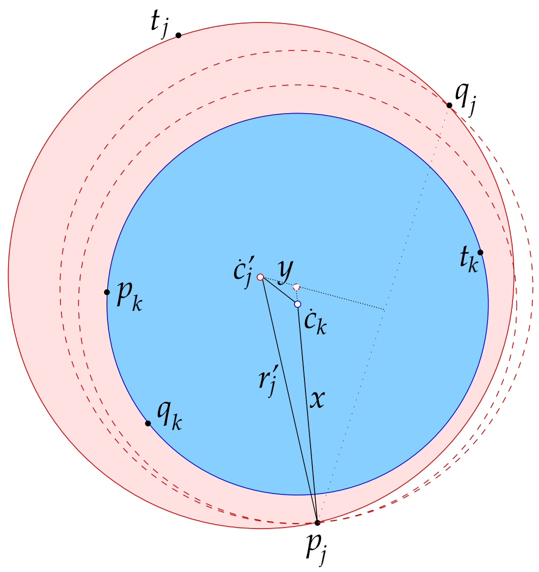


Fig. 8. Two centers $\dot{c}'_j$ and $\dot{c}_k$ are nearby.

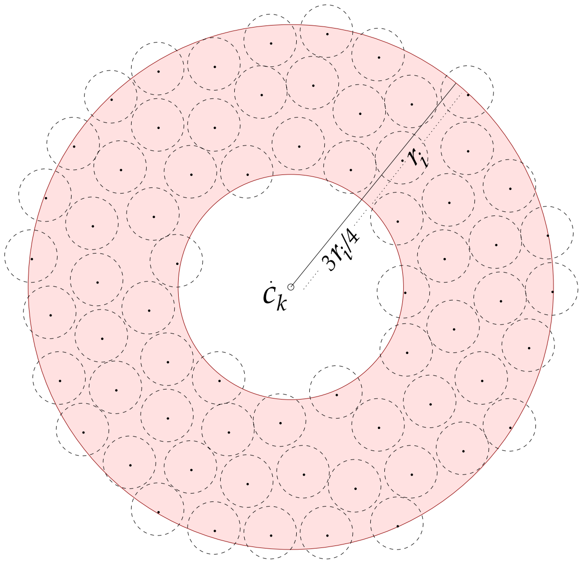


Fig. 9. Packing non-overlapping disks (dashed) with radius $3r_i/16$ and centers located inside the shaded region.

is at most twice and at least half the the radius of c'_j. Therefore, the radius of c''_j is at least $|r'_j|/2 \geq 3r_i/8$. We also know that the radius of c''_j, is at most r_i because r''_j is at most the radius of the largest Delaunay circle at iteration j which is at most r_i. So $\dot{c}''_j$ and $\dot{c}'_j$ are at most r_i away from each other. Hence,

$$||\dot{c}''_j - \dot{c}_k|| \leq ||\dot{c}''_j - \dot{c}'_j|| + ||\dot{c}'_j - \dot{c}_k|| \leq r_i + 3r_i/4 \leq 7r_i/4.$$

Let $\dot{C}'' = \{\dot{c}''_{i+1}, \dot{c}''_{i+2}, \ldots, \dot{c}''_j, \ldots, \dot{c}''_k\}$ be the set of centers we have shown that were inserted in iterations $j = i+1, \ldots, k$. Let C'' be the corresponding set of circles. Due to the empty circle property, each circle $c''_j \in C''$ is empty of all the centers $\dot{c}''_l$ where $l < j$. So the centers in $\dot{C}''$ are pairwise at least $3r_i/8$ apart from each other, implying that the circles of radius $3r_i/16$ on these vertices are pairwise disjoint. The annulus containing these circles has area at most $\pi(7/4+3/16)^2r_i^2 - \pi(3/4-3/16)^2r_i^2$. See Figure 9. One can pack at most

$$\left\lfloor \frac{\pi[(7/4+3/16)^2 - (3/4-3/16)^2]r_i^2}{\pi(3/16)^2r_i^2} \right\rfloor = 97$$

circles in this region. So the size of the set C'', $k - i$, is at most 97, which is a contradiction. □

Theorem 2. *Parallel Chew's refinement algorithm takes $O(\log(L/s))$ iterations to generate a size-optimal well-shaped uniform mesh, where L is the diameter of the input and s is the length of the shortest edge in the output mesh.*

Proof. Initially, the diameter of the largest Delaunay circle is at most L. By Lemma 2, the size of the largest Delaunay circle shrinks at least by a factor of 3/4 at every $k = 98$ iterations. The diameter of the largest Delaunay circle in the resulting mesh is at least s. Thus, our parallel refinement algorithm terminates after at most $98 \log_{4/3}(L/s)$ iterations. The size optimality and quality guarantee of the algorithm follows from Theorem 1, where we proved that there is a sequential algorithm that outputs the same mesh as our parallel algorithm. □

4.1.3. *Parallel Computation of Maximal Independent Sets*

One can use the parallel maximal independent set algorithm of Luby[29] to compute a parallel independent set of candidates for each iteration in $O(\log^2 n)$ parallel time. In this section, we will explain how we can exploit the geometric structure of the independence relation to compute a maximal independent set in a constant parallel time.

We will make extensive use of the result of Lemma 1 that two circumcircles are conflicting at iteration j only if their radii are within a factor of 2 of each other.

Lemma 3. *At iteration j, if there are n_j circumcircles, then a maximal independent set of candidates for Delaunay refinement can be computed in constant parallel time using n_j processors.*

Proof. Let C_h^j be the set of circumcircles of radius more than $L/2^{h+1}$ and less than or equal to $L/2^h$, where h ranges from 0 to $\log(L/s_j)$ and s_j is the smallest circumradius at iteration j. Note that a circumcircle in C_h^j does not conflict with any circumcircle in C_l^j if $l > h + 1$.

To compute a maximal set of non-conflicting candidates, we first in parallel find a maximal independent sets of circumcircles in C_h^j, independently for all even h. We will show below that a maximal independent set of circumcircles in C_h^j can be computed in constant time in parallel. Let I_{even}^j be the set of independent circumcircles computed. Then in one parallel step, we can eliminate all conflicting circumcircles in $\cup_{h:odd} C_h^j$. We then compute a maximal independent set for remaining circumcircles in C_h^j for all odd h. Let this set be I_{odd}^j. Then $I_{even}^j \cup I_{odd}^j$ is a maximal independent set of circumcircles for iteration j.

Note that all circumcircles in C_h^j have radius between $L/2^{h+1}$ and $L/2^h$. If we divide the square containing all circumcenters into a 2^h-by-2^h grid, then any circumcenter that is conflict with a circumcenter in the grid box (x, y) must lie either in grid box (x, y) or one of its eight grid neighbors.

We color grid boxes (x, y) with color $(x \bmod 3, y \bmod 3)$. We then cycle through the 9 color classes and, by a method we will explain momentarily, find a maximal independent set of the candidates in each grid-box of the current color in parallel. We then eliminate in parallel the conflicting circumcenters that are in the color classes that have not yet been processed.

Finally, we explain how to compute a maximal independent set among the candidates that lie in a given grid-box. First notice that any maximal independent set of candidates in a grid-box can have at most a constant number of members, and hence a maximal independent set can be found by a constant number of parallel selection-elimination operations: choose a center that has not been eliminated, and in parallel eliminate any centers with which it conflicts.

In a parallel system that supports primitives such fetch_and_add, test_and_set, or parallel scan, we can use such a primitive to select in constant time a candidate in a grid box. The processor that holds this candidate becomes a "leader" in that round and broadcasts its candidate so that the conflicting candidates can be eliminated. With these primitives, our algorithm can be implemented in parallel constant time. However, if the parallel system does not support these primitives, then for each grid cell we can emulate parallel scan to select a leader in $O(\log n)$ time, where n is the number of candidate centers in the cell.

In general, many grid cells are empty and there is no need to generate them at all. We can use hashing to select grid cells that are not empty. The idea is very simple, each candidate center can compute the coordinates of its grid cell from the coordinates of its center and its radius. We can hash grid cells using their coordinates and therefore, all candidate centers belonging to a grid cell can independently generate the hash identity of the cell. We can then use parallel primitives discussed in the paragraph above to support the computation of a maximal independent set of candidates for all non-empty grid cells. □

4.1.4. *Parallelizing Ruppert's Refinement*

In this section, we show that our parallelization of Ruppert's method for periodic point sets in 2D takes $O(\log^2(L/s))$ iterations. For simplicity, we give an analysis for the case $\beta_R = \sqrt{2}$, although our analysis can be easily extended to the case when $\beta_R = 1 + \epsilon$, for any $\epsilon > 0$. We recall that β_R is the threshold of the ratio of the radius to shortest edge-length defining a poorly shaped triangle. Thus, for $\beta_R = \sqrt{2}$, inserting the circumcenter of a poorly shaped triangle whose shortest edge is h introduces new Delaunay edges of length at least $\sqrt{2}h$.

For analysis purposes, we integrate an artificial outer loop into the algorithm. The outer loop is related to a classification of the edges by their length. Let h be the shortest edge length in the initial Delaunay triangulation. The class $\mathcal{E}_i$ contains the edges whose length are in the half open interval $\left[\sqrt{2}^{i-1}h, \sqrt{2}^{i}h\right)$. See Figure

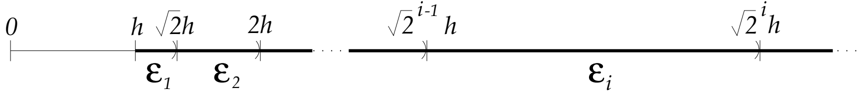


Fig. 10. Edge classes

10. Therefore, there are at most $\lceil \log_{\sqrt{2}}(L/h) \rceil$ classes. Each triangle is associated with its shortest edge and classified accordingly. A triangle is said to be associated with edge class $\mathcal{E}_i$ if its shortest edge is in $\mathcal{E}_i$. The i^{th} iteration of the outer loop in the following algorithm removes all the bad triangles associated with $\mathcal{E}_i$.

Algorithm 3 Parallel Ruppert's refinement

Input: A periodic point set P in $\mathbb{R}^2$
  Let T be the Delaunay triangulation of P
  for i=1 to $\lceil \log_{\sqrt{2}}(L/h) \rceil$ **do**
    Let $\dot{\mathcal{C}} = \{\dot{c}_1, \ldots, \dot{c}_n\}$ be all circumcenters of bad triangles associated with $\mathcal{E}_i$
    while $\dot{\mathcal{C}}$ is not empty **do**
      Let $\mathcal{I}$ be a maximal independent subset of $\dot{\mathcal{C}}$
      Update Delaunay triangulation inserting all points in $\mathcal{I}$ in parallel
      Update $\dot{\mathcal{C}}$
    end while
  end for

We first show that the inner loop of the algorithm takes $O(\log(L/s))$ iterations.

Lemma 4. *Suppose the shortest edge associated with a bad triangle is in $\mathcal{E}_i$. Let $e \in \mathcal{E}_i$ be an edge exists at the end of the j^{th} iteration of the inner loop. Let r_j be the circumradius of the triangle with largest circumcircle among those associated with e. Then after 81 iterations, one of the following is true*

(i) e does not exist anymore;
(ii) all the triangles associated with e are good;
(iii) the largest circumcircle associated with e has radius $r_{j+81} \leq 3r_j/4$.

Proof. We will show that if the neither the first nor the second case holds then the third must be true. Say at the end of iteration l for some l and j with $l - j = 81$, e exists and not all the triangles associated with e are good. Let t_l be the worst quality triangle associated with e at iteration l. Suppose to the contrary that the circumradius of t_l, $r_l > 3r_j/4$. For any k, such that $j < k \leq l$, let t_k be the worst Delaunay triangle associated with e at the k^{th} iteration. Let r_k be the circumradius of t_k. We have $r_k \geq r_l > 3r_j/4$. So, at iteration k either circumcenter $\dot{c}_k$ of t_k or a circumcenter $\dot{c}'_k$ in conflict with $\dot{c}_k$ is inserted. Moreover, by Lemma 1 the radius of the corresponding circle c'_k is at least half the size of c_l.

Let $\dot{C}'' = \{\dot{c}''_{j+1}, \dot{c}''_{j+2}, \ldots, \dot{c}''_l\}$ be the set of centers that were inserted in iterations $k = i+1, \ldots, l$. Let C'' be the corresponding set of circles. The radius of each one of these circles is at least $3r_j/8$. Due to empty circle property, each circle $c''_k \in C''$ is empty of all the centers in C'' inserted prior to $\dot{c}''_k$. So the centers in $\dot{C}''$ are pairwise at least $3r_j/8$ apart from each other. This in turn implies that the circles of radius $3r_j/16$ on the centers in $\dot{C}''$ do not overlap. These circles can spread into an area of size $2\pi(r_j + 3r_j/16)^3$. One can fit at most

$$\left\lfloor \frac{2\pi(r_j + 3r_j/16)^2}{\pi(3r_j/16)^2} \right\rfloor = \left\lfloor \frac{2(19/16)^2}{(3/16)^2} \right\rfloor = \left\lfloor \frac{722}{9} \right\rfloor = 80$$

circles in that region. Therefore, the number of centers in $\dot{C}''$, $l - j$ is at most 80, which is a contradiction. □

Our algorithm, one at a time, handles the bad triangles associated with each edge class. At the end of iteration i, there are no bad triangles left associated with the edge class $\mathcal{E}_i$. In the next Lemma we justify the double loop structure of our parallel Chew's refinement algorithm. We prove that during and after the i^{th} iteration of the outer loop no bad triangle associated with an edge in $\mathcal{E}_i$ is introduced.

Lemma 5. *Suppose the shortest edge associated with a triangle is in $\mathcal{E}_i$. During and after the i^{th} iteration of the outer loop of the parallel Ruppert's refinement, the following are true:*

- *(i) if an edge $e \in \mathcal{E}_i$ disappears it never appears again;*
- *(ii) no new edges are introduced to edge class $\mathcal{E}_i$;*
- *(iii) the radius-edge ratio of a triangle associated with an an edge $e \in \mathcal{E}_i$ does not increase.*

Proof. (i) An edge e is part of Delaunay triangulation if and only if there is an empty circle that goes through e. Once there no such circle remains, point insertions is no help to edge e for having a new empty circle. (ii) During and after the i^{th} iteration of the outer loop the smallest edge that can be introduced has length at least $\sqrt{2}^i h$. (iii) Consider an edge $ab \in \mathcal{E}_i$ and a triangle abc associated with ab

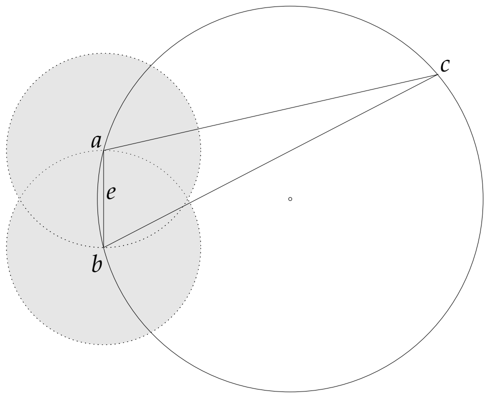


Fig. 11. The quality of the triangles incident to an edge $e \in \mathcal{E}_i$ can only be improved during and after the i^{th} iteration of the outer loop.

(Figure 11). We claim that the quality of a triangle abc' replacing abc is better than that of abc. The new vertex c' must be inside the circumcircle of abc. Otherwise, abc is intact. On the other hand, c' can not be too close to edge e (in the shaded region in Figure 11). Otherwise, a shorter edge than ab would be introduced contradicting (ii). The radius of the circumcircle of abc' is smaller than that of abc when c' is inside the circumcircle of abc but outside the shaded region. This in turn, implies that the radius-edge ratio of a triangle associated to an edge in $\mathcal{E}_i$ can only be improved through refinement. □

Theorem 3. *For a given 2D periodic point set of diameter at most L, parallel Ruppert refinement takes $O(\log^2(L/s))$ iterations, where s is the length of the shortest edge in the mesh generated by Ruppert's refinement. The output mesh is well-shaped and size-optimal.*

Proof. By Lemma 5, during and after the i^{th} iteration of the outer loop, no triangles associated with an edge class $\mathcal{E}_j$ are introduced for any $j < i$. So when the parallel Ruppert's algorithm terminates, no bad triangles are left in the mesh. On the other hand, Lemma 4 together with Lemma 5 implies that each outer loop of the parallel Ruppert's algorithm takes at most $81 \log_{4/3}(L/s)$ iterations of the inner loop. This gives an upper bound of $O(\log^2(L/s))$ on the number of iterations of the parallel Ruppert's refinement algorithm. The quality guarantee and size optimality of the mesh is due Theorem 1. □

Remark 1. We assumed $\beta_R = \sqrt{2}$ in the above discussion. When the input is a periodic point set, the sequential Ruppert algorithm terminates with the optimal size and quality, if $\beta_R \geq 1$ [42]. Our arguments for parallel Ruppert's algorithm still holds if $\beta_R \geq 1 + \epsilon$ for any constant $\epsilon > 0$. In the generalized argument, we need $O(\log_{1+\epsilon}(L/s))$ classes. Class $\mathcal{E}_i$ contains edges whose length are in the half open interval $\big((1+\epsilon)^i h, (1+\epsilon)^{i+1} h)\big]$. The number of inner loop iterations do not depend on ϵ. So the parallel algorithm runs in $O(\log_{1+\epsilon}(L/s)\log_{4/3}(L/s))$ time for any $\beta_R \geq 1 + \epsilon$.

4.2. *Parallel Delaunay Refinement for PSLGs*

In this subsection, we extend our parallel algorithm for generating a Delaunay mesh from a domain given by a periodic point set to a domain defined by a planar straight-line graph. Following Ruppert, we assume that the angle between two adjacent input segments is at least $\pi/2$. A key step in Delaunay refinement for a domain specified by a PSLG is to properly add points to the boundary segments so that the Delaunay mesh is conforming to the boundary. In our parallel algorithm, we make our mesh conform to the boundary in two steps: First, we use, presented in Section 4.2.3, an $O(\log L/s)$ time parallel preprocessing algorithm to insert points on the input segments so that the initial Delaunay mesh is conforming to the boundary and no diametral circle intersects any other non-incident input features. Second when a segment is encroached during parallel Delaunay refinement, we include its midpoint as candidates for insertion.

The preprocessing step might not be needed to implement our parallel algorithm: one could probably add points to the boundary as needed. However, the preprocessing step simplifies our analysis in this section by greatly reducing the number of cases in the analysis.

4.2.1. *A generic parallel algorithm*

After applying the preprocessing step, the initial Delaunay triangulation is conforming to the boundary and no diameter circle contains any point of the triangulation. We will maintain this invariant in our algorithm.

In order to perform parallel refinement, as in Section 4.1.1, we need a rule of *independence* among candidates for refining boundary segments and poorly shaped triangles. We first recall the set of candidates for insertion defined in Section 3.

Let $\mathcal{B}$ be the set of circumcircles of poorly shaped triangles whose centers $\dot{\mathcal{B}}$ encroach some boundary segments. Let $\mathcal{C}$ be the set of circumcircles of poorly shaped triangles whose centers $\dot{\mathcal{C}}$ don't encroach any boundary segments. Let $\mathcal{D}$ is the set of diametral circles that are encroached by some centers in $\dot{\mathcal{B}}$. So, $\dot{\mathcal{C}} \cup \dot{\mathcal{D}}$ are candidate points for insertion.

We will still apply Definition 1 to determine whether two circumcenters from $\dot{\mathcal{C}}$ are independent. Because the angle between two adjacent input segments is at least $\pi/2$, after preprocessing, any two diametral circles from $\dot{\mathcal{D}}$ are not in conflict. Also, diametral centers in $\dot{\mathcal{B}}$ are not in conflict because they are not candidates.

We will use the following definition of independence between a diametral center in $\dot{\mathcal{D}}$ and a circumcenter in $\dot{\mathcal{C}}$. Note that because a circumcenter in $\dot{\mathcal{C}}$ does not encroach any boundary segment, a diametral circle of $\mathcal{D}$ does not contain any center in $\dot{\mathcal{C}}$.

Definition 2. A circumcenter $\dot{c} \in \dot{\mathcal{C}}$ and a diametral center $\dot{d} \in \dot{\mathcal{D}}$ are said to *conflict* if (i) $\dot{d}$ is inside c; and (ii) the radius of c is smaller than $\sqrt{2}$ times the radius of d. Otherwise, $\dot{c}$ and $\dot{d}$ (also c and d) are *independent*.

This definition of independence is motivated by the following lemma first proved by Ruppert.[42,48]

Lemma 6. *If a circumcircle c of radius r_c encroaches a diametral circle d of radius r_d, then $r_d \geq r_c/\sqrt{2}$.*

The following theorem extends Theorem 1 for domains given by PSLGs.

Theorem 4. *For a domain Ω specified by a PSLG, suppose M is a mesh produced by an execution of the parallel algorithm above. Then M can be obtained by some execution of one of the sequential Delaunay refinement algorithms discussed in Section 3.*

Algorithm 4 Generic Parallel Delaunay Refinement

Input: A domain Ω given by a PSLG in $\mathbb{R}^2$

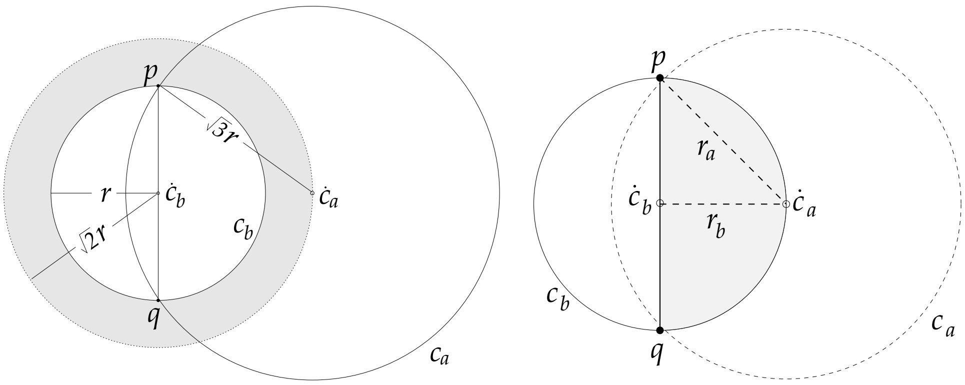
Apply the parallel preprocessing algorithm of Section 4.2.3
Let T be the initial Delaunay triangulation.
Compute $\dot{\mathcal{BC}}$, an independent subset of $\dot{\mathcal{B}} \cup \dot{\mathcal{C}}$
Let $\dot{\mathcal{D}}$ be the set of centers of diametral circles encroached by the centers in $\dot{\mathcal{BC}}$
while $(\dot{\mathcal{BC}} \cap \dot{\mathcal{C}}) \cup \dot{\mathcal{D}}$ is not empty **do**
  Let $\mathcal{I}$ be an independent subset of $(\dot{\mathcal{BC}} \cap \dot{\mathcal{C}}) \cup \dot{\mathcal{D}}$
  Update Delaunay triangulation inserting all points in $\mathcal{I}$ in parallel
  Update $\dot{\mathcal{B}}$, $\dot{\mathcal{C}}$, $\dot{\mathcal{BC}}$ and $\dot{\mathcal{D}}$
end while

Proof. The proof is similar to that of Theorem 1. Let $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$ be the sets of vertices inserted at all k iterations of our parallel algorithm. Again, we suggest a sequential algorithm that inserts all points in $\mathcal{I}_i$ before any points in $\mathcal{I}_j$ for all $i < j$. The key difference is in showing that the centers in a single parallel iteration can be ordered so that all are inserted by a sequential algorithm. As before, we construct a dependency graph among the centers inserted in the same parallel iteration and prove that this graph is acyclic.

Let $\mathcal{B}_i$ be the set of circumcenters encroaching to the diametral circles in $\mathcal{I}_i$. For any two centers $\dot{a}, \dot{b} \in \mathcal{B}_i \cup \mathcal{I}_i$, $\dot{a}$ has to wait for the insertion of $\dot{b}$ if $\dot{a}$ is inside b. An arc from $\dot{a}$ to $\dot{b}$ in the dependency graph represents this. Furthermore, we label each arc by a number indicating the factor of difference between the size of spheres at each end of the arc. Specifically, a number x on an arc from a to b indicates that the ratio of radius of b to the radius of a is larger than x. The conflict rule in Definition 1 implies that the number on an arc from a circumcenter to another circumcenter is 1. The conflict rule in Definition 2 implies that the number on an arc from a diametral center to a circumcenter is $\sqrt{2}$. We prove in Lemma 7 that the size of an encroached diametral circle to its encroaching center is at least $1/\sqrt{2}$. Notice that there is no arc from non-encroaching centers to diametral centers because a diametral center does not contain a non-encroaching center. It is not possible to have a cycle in this generic dependency graph such that the edge weights multiply to a number less than 1. So we conclude that there is no cycle on any instance of the dependency graph. As before, topological sorting gives us a valid sequential insertion order within each $\mathcal{I}_i$.

Rest of the proof is identical to that of Theorem 1. □

For clarity of the presentation and analysis, we will assume that the diametral circles of PSLG segments do not intersect any other segment. In Section 4.2.3, we will present a parallel preprocessing algorithm that will convert an input PSLG to this form. We will also show that the local feature size function of the modified PSLG is at most a constant factor smaller than the local feature size of the



(a) (b)

Fig. 12. Bounds on the size of the rejected circles. (a) a circumcircle c_a in conflict with a diametral circle cannot be much larger than the diametral circle; (b) a circumcircle c_a whose center encroaches a diametral circle cannot be much larger than the diametral circle.

original PSLG. So the size optimal mesh for the modified PSLG is also size optimal for the original. Moreover, we will show that preprocessing can be performed asymptotically as efficiently as the parallel refinement process.

At each iteration of the generic parallel refinement algorithm, a circumcenter of a bad triangle can be rejected due either (i) because it conflicts with another center which is chosen as a Steiner point; or (ii) because it encroaches a segment. In the following lemma we give lower and upper bounds on the size of conflicting circles. We will use these bounds later when analyzing the specific parallel Delaunay refinement algorithms suggested in the next subsection.

Lemma 7. *Let c_a be a circumcircle rejected by the parallel Delaunay refinement algorithm at iteration i. Then, either one of the following is true.*

(i) there is a conflicting circumcircle c_b whose center is chosen as a Steiner point at iteration i and $r_b/2 \leq r_a \leq r_b$.
(ii) there is a conflicting diametral circle c_b whose center is chosen as a Steiner point at iteration i and $r_b \leq r_a \leq \sqrt{3}r_b$.
(iii) there is a diametral circle c_b that c_a encroaches. Moreover, the center of c_b is chosen as a Steiner point and $r_a/\sqrt{2} \leq r_b$.

Proof. If $\dot{c_a}$ does not encroach a segment, then some conflicting center $\dot{c_b}$ is chosen as a Steiner point; otherwise, $\dot{c_a}$ would not be rejected. The conflicting center $\dot{c_b}$ is either a circumcenter or a diametral center. (i) The first case is identical to Lemma 7.

(ii) Now suppose the conflicting center $\dot{c_b}$ is a diametral center. (See Figure 12 (a)). By the conflict rule definition (Definition 2) $r_b \leq r_a$. The endpoints of the encroached segment, p and q, are not inside c_a; otherwise, c_a would not be empty. This implies that c_a is largest when its center $\dot{c}_a$ is furthest point from both p and q. The worst location for $\dot{c}_a$ is the intersection point of the bisector of pq and the circle of size $\sqrt{2}r_b$ centered at $\dot{c}_b$. This implies $r_a \leq \sqrt{3}r_b$.

(iii) Finally, if $\dot{c_a}$ encroaches a segment, then some diametral circle c_b is chosen as a Steiner point. Let pq be the encroached segment (see Figure 12 (b)). The points p and q are not inside c_a; otherwise, c_a would not be empty. This implies that c_a is largest when its center $\dot{c}_a$ is furthest from both p and q. The worst location for $\dot{c}_a$ is the intersection point of the bisector of pq and the circle c_b. Hence, $r_a \leq \sqrt{2}r_b$.□

4.2.2. *Parallelizing Chew's Refinement for PSLG*

In this section we give an analysis of parallel Chew refinement algorithm for input PSLGs. The algorithm is derived from Algorithm 4 by using a *maximal* independent subset of the candidate centers instead of an arbitrary independent subset. We will refer to this specific version of the algorithm as the parallel Chew's refinement algorithm. It is similar to our parallel algorithm (discussed in Section 4.1.2) that handles the point set inputs and bares the same name.

The following lemma extends Lemma 2 for the PSLG input. We will refer to illustrations in Section 4.1.2. The proof is very similar but more involved due to larger number of cases. Let r_i be the radius of the largest Delaunay circle c_i at the end of iteration i.

Lemma 8. *$r_{i+237} \leq 3r_i/4$.*

Proof. To prove the lemma by contradiction, suppose $r_k \geq \beta_C \bar{\boldsymbol{s}}$ and $r_k > \frac{3}{4}r_i$ for some i and k with $k - i = 237$. Since we pre-process the input and never insert the encroaching centers, $\dot{c}_k$ is in the triangulated domain.

We use the same dilation technique as in Lemma 2 to show that for each iteration j, where $i < j \leq k$, there exists a bad simplex with circumcirle c'_j, circumcenter $\dot{c}'_j$ and circumradius r'_j such that (i) $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$ and (ii) $r'_j \geq 3r_i/4$. (See Figure 7 and Lemma 2.)

During this dilation process the moving center do not cross out the boundary of the domain. Otherwise, there would be an encroaching vertex. The resulting circle c'_j is a circumcircle of a Delaunay triangle $p_j q_j t_j$. Furthermore, $p_j q_j t_j$ is a bad triangle because its circumradius at least $\beta_C \bar{\boldsymbol{s}}$. As shown in the proof of Lemma 2 $\dot{c}'_j$, a candidate vertex in iteration j, is not inserted in iteration j. In the case of an input PSLG, $\dot{c}'_j$ is rejected either because it encroaches a segment or conflicts with another candidate circumcenter.

In the first case, let d'_j be the diametral circle of the encroached segment. By Lemma 7, the size of d'_j is at most a factor smaller than the size of c'_j. We know by Lemma 7 that either $\dot{d}'_j$ is inserted or the parallel algorithm inserts another center $\dot{cd}_j$ in conflict with $\dot{d}'_j$. Furthermore, cd_j is at least the same size as d'_j. Let $\dot{d}''_j$

be the choice of insertion among $\dot{d}'_j$ and $\dot{c}d_j$. An upper bound on the size of d''_j is simply r_i, the size of the largest Delaunay circle at iteration i. So there is a Steiner vertex, the center of d''_j, inserted at iteration j. This new vertex is at most $2r_i$ away from $\dot{c}_j$.

In the second case, let $\dot{c}''_j$ be the center of the conflicting circle c''_j. By Lemma 7, the size of c''_j is at most twice and at least half the the size of c'_j. Moreover, the radius of c''_j can not be larger than r_i. So $\dot{c}''_j$ and $\dot{c}'_j$ are at most r_j away from each other.

Combining the two cases, at iteration j a Steiner vertex $\dot{c}'''_j$ ($\dot{c}''_j$ or $\dot{d}''_j$), center of an empty circle c'''_j (c''_j or d''_j), is inserted by the parallel Chew's refinement algorithm. Vertex $\dot{c}'''_j$ is at most $2r_i$ away from $\dot{c}'_j$. So $||\dot{c}'''_j - \dot{c}_k|| \leq ||\dot{c}'''_j - \dot{c}'_j|| + ||\dot{c}'_j - \dot{c}_k|| \leq 2r_i + 3r_i/4 \leq 11r_i/4$.

In addition, c'''_j does not contain any centers inserted before iteration j. Let $\dot{C}''' = \{\dot{c}'''_{i+1}, \dot{c}'''_{i+2}, \ldots, \dot{c}'''_j, \ldots, \dot{c}'''_k\}$ be the set of centers that were inserted in iterations $j = i+1, \ldots, k$. The rest of the proof is same as in Lemma 2 except the area of the packing region. The spheres of radius $3r_i/16$ centered at the members of $\dot{C}'''$ do not overlap and can spread to an area of size at most $\pi(11/4 + 3/16)^2 r_i^2 - \pi(3/4 - 3/16)^2 r_i^2$. One can pack at most

$$\left\lfloor \frac{\pi[(11/4+3/16)^2 - (3/4-3/16)^2] r_i^2}{\pi(3/16)^2 r_i^2} \right\rfloor = 236$$

circles in this region. Hence, there could be at most 236 iterations before the size of the largest Delaunay circle gets a factor of 3/4 smaller, a contradiction. □

The following Theorem extends Theorem 2 to PSLGs. The proof is almost identical, except we use Lemma 8 in place of Lemma 2.

Theorem 5. *Parallel Chew's refinement algorithm takes $O(\log(L/s))$ iterations to generate a size-optimal well-shaped uniform mesh, where L is the diameter of the input PSLG Ω and s is the length of the shortest edge in the output mesh.*

4.2.3. *Parallel Preprocessing*

In the algorithm and proof presented in the last subsection, we assume that the boundary of the domain has been preprocessed to satisfy the following property.

Definition 3 (Strongly Conforming). A domain Ω specified by a PSLG is *strongly conforming* if no diametral circle contains any vertex or intersects any other non-incident input features.

Clearly, if Ω is strongly conforming, then the Delaunay triangulation of the vertices of Ω is conforming to Ω.

We will use the following parallel method to preprocess a domain Ω to make it strongly conforming. This method repeatedly adds midpoints to boundary segments whose diametral circles intersect non-incident input features. See also Figure 13.

Algorithm 5 Parallel Boundary Preprocessing

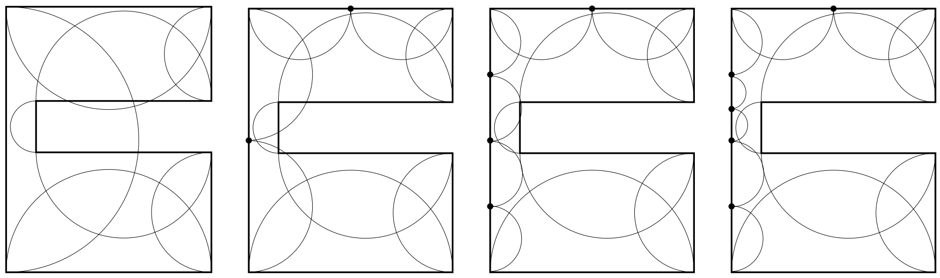
Input: A PSLG domain Ω in $\mathbb{R}^2$

Let $\mathcal{G}$ be the set of segments in Ω whose diametral circles intersect non-incident input features.

while $\mathcal{G}$ is not empty **do**

  Split all the segments in $\mathcal{G}$ in parallel by midpoint insertion and update $\mathcal{G}$.

end while



(a) (b) (c) (d)

Fig. 13. Boundary preprocessing. At each iteration we split all the segments whose diametral sphere is not empty of other input features.

Lemma 9. *Parallel Boundary Preprocessing terminates in* $O(\log(L/\boldsymbol{s}))$ *iterations.*

Proof. Assuming that the angle between two adjacent input segments is at least $\pi/2$, Lemma 10 below show that no segment of length less than $s/3$ will be introduced, where s is the smallest local feature size of Ω. In addition, the decision of whether a segment should be divided is independent of whether any other segment should be divided. So, we can process all segments in parallel. Each split of a segment reduces length of the segment by half, implying that the number of parallel iterations is $O(\log(L/\boldsymbol{s}))$. □

In the scheme above, we can grow a quadtree level by level to support the query of whether the diametral circle of a segment intersects another non-incident feature. The number of levels of the quadtree that we need to grow is at most $\log(L/s)$. As shown by Bern *et al.*,[3,4] one can use balanced quadtree to approximate local feature size function of Ω to within a constant factor. Therefore, using a balanced quadtree,[3,4] we can preprocess the domain in $\log(L/s)$ parallel time so that the preprocessed domain is *strongly feature conforming* as defined below.

Definition 4 (Strongly Feature Conforming). Let $\alpha > 2$ be a constant. A domain Ω specified by a PSLG is *strongly feature conforming* with parameter α if it is strongly conforming, and in addition, the length of each segment is no more than α times the local feature size of its midpoint.

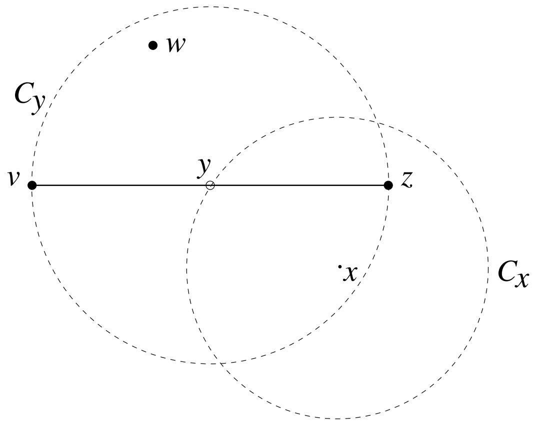


Fig. 14. Local feature size after the preprocessing.

In the next subsection, we will present a parallel implementation of Ruppert's algorithm for domains that are strongly feature conforming and show that it terminates in $O(\log^2(L/s))$ iterations.

We use the following lemma to show that the size optimality of our results are not affected much by the preprocessing.

Lemma 10. *Let Ω and Ω' denote the input before and after preprocessing, respectively. Then, for any point x in these domains,* $\mathrm{lfs}_\Omega(x)/3 \leq \mathrm{lfs}_{\Omega'}(x) \leq \mathrm{lfs}_\Omega(x)$.

Proof. During the preprocessing we only introduce new vertices on the segments and never remove any features. So $\mathrm{lfs}_{\Omega'}(x) \leq \mathrm{lfs}_\Omega(x)$ for all x in Ω. To prove the lower bound, consider a point x and the circle C_x centered at x with radius $\mathrm{lfs}_{\Omega'}(x)$. By the definition of local feature size, there are at least two non-incident features y and z inside C_x. At least one of these is introduced during preprocessing, or else $\mathrm{lfs}_{\Omega'}(x) = \mathrm{lfs}_\Omega(x)$. Moreover, y and z must be on the same input segment; otherwise, $\mathrm{lfs}_{\Omega'}(x) = \mathrm{lfs}_\Omega(x)$. Without loss of generality, let y be a midpoint inserted during preprocessing. (See Figure 14.) Insertion of y implies that there is an original input feature w inside the circle C_y centered at y with radius $|yz|$. The feature w is not incident to y, z, or to the original segment they belong to. Thus, $\mathrm{lfs}_\Omega(x)$ is at most $|xw|$. Hence, $\mathrm{lfs}_\Omega(x) \leq |xw| \leq |xy| + |yw| \leq \mathrm{lfs}_{\Omega'}(x) + |yz| \leq \mathrm{lfs}_{\Omega'}(x) + 2\mathrm{lfs}_{\Omega'}(x) \leq 3\mathrm{lfs}_{\Omega'}(x)$. □

4.2.4. *Parallelizing Ruppert's Refinement for PSLG*

In this section, we show that our parallelization of Ruppert's method for a domain given by a PSLG takes $O(\log^2(L/s))$ iterations. Again, for simplicity, we will only give an analysis for the case when $\beta_R = \sqrt{2}$.

The parallel algorithm follows basic steps of the parallel Ruppert's Refinement presented earlier in Section 4.1.4. But first, we apply the parallel preprocessing algorithm of Section 4.2.3 so that the preprocessed domain is strongly feature conforming. So below we can assume that Ω is strongly conforming.

As before we associate Delaunay triangles with edge classes. We assign an edge to class $\mathcal{E}_i$ if its length is in $\left[\sqrt{2}^{i-1}h, \sqrt{2}^{i}h\right)$, where h is the shortest edge in the initial triangulation.

Algorithm 6 Parallel Ruppert's Refinement

Input: A domain Ω given by a PSLG that is strongly feature conforming.
  Let T be the initial Delaunay triangulation.
  for i=1 to $\lceil \log_{\sqrt{2}}(L/s) \rceil$ **do**
    Let $\dot{\mathcal{B}}$ be encroaching candidate circumcenters and $\dot{\mathcal{C}}$ be the non-encroaching candidate circumcenters whose triangles is in class $\mathcal{E}_i$.
    Compute $\dot{\mathcal{BC}}$, an independent set of $\dot{\mathcal{B}} \cup \dot{\mathcal{C}}$.
    Let $\dot{\mathcal{D}}$ be the set of centers of diametral circles encroached by the centers in $\dot{\mathcal{BC}}$
    while $(\mathcal{BC} \cap \mathcal{C}) \cup \mathcal{D}$ is not empty **do**
      Let $\mathcal{I}$ be an maximal independent subset of $(\dot{\mathcal{BC}} \cap \dot{\mathcal{C}}) \cup \dot{\mathcal{D}}$
      Update Delaunay triangulation inserting all points in $\mathcal{I}$ in parallel
      Update $\dot{\mathcal{B}}$, $\dot{\mathcal{C}}$, $\dot{\mathcal{BC}}$ and $\dot{\mathcal{D}}$
    end while
  end for

Theorem 6. *Given a domain specified by a PSLG, the Parallel Ruppert's Refinement Algorithm takes $O(\log^2(L/s))$ iterations.*

The proof of Theorem 6 is essentially the same as the proof of Theorem 3 where we need to address the following the two issues.

(1) The center of a circumcircle could potentially encroach a boundary segment whose length is much larger than that the circumradius.
(2) The insertion of a midpoint on the boundary could potentially introduce smaller edges.

To address the first issue, we apply parallel processing algorithm of Section 4.2.3 and hence assume Ω is strongly feature conforming. Hence if a circumcenter encroaches a boundary segment, the circumradius and the length of the segment are with a constant factor of each other. In addition, because each boundary segment can only be split at most a constant times in the refinement (this is due to the size optimality proof of Ruppert[42], it can not introduce smaller edges too many times.

5. Delaunay Refinement in Three Dimensions

The three dimensional input for sequential Delaunay refinement algorithms is usually described as a PLC (see Section 2.1 for its definition). The angle between any two intersecting elements, when one is not contained in the other, is assumed to be at least 90°. The definition of segment encroachment given in the previous section extends to three dimensions by simply replacing circles with spheres. In three dimensions triangular faces of a PLC can also be encroached.

Definition 5. The *equatorial sphere* of a triangle is the smallest sphere that passes through its vertices. A triangular subfacet of a PLC is *encroached* if the diametral sphere is not empty.

5.1. *Sequential Delaunay Refinement in Three Dimensions*

Chew's algorithm extends naturally to 3D. Shewchuk[48] developed a 3D extension of Ruppert's algorithm. Algorithm 7 presents a generic version of the sequential Delaunay refinement. In Shewchuk's refinement, a tetrahedron is *bad* if the ratio of its circumradius to its shortest edge, referred as the radius-edge ratio, is more than a pre-specified constant $\beta_S \geq 2$. On the other hand for Chew's refinement, a tetrahedron is *bad* if the ratio of its circumradius to the shortest feature size in the input is more than a pre-specified constant $\beta_C > 1$.

Algorithm 7 3D Delaunay Refinement

Input: A PLC domain Ω in $\mathbb{R}^3$

  Compute T, the Delaunay triangulation of the points of Ω
  Let $\dot{C}$ be the set of non-encroaching circumcenters of the bad tetrahedra
  Let $\dot{D}$ be the set of non-encroaching equatorial centers of the encroached triangular subfacets
  Let $\dot{E}$ be the set of diametral centers of the encroached subsegments.
  while there is center a in $\dot{C} \cup \dot{D} \cup \dot{E}$ **do**
    Insert a and update the Delaunay triangulation
    Update $\dot{C}$, $\dot{D}$, and $\dot{E}$
  end while

5.2. *Parallel Delaunay Refinement in Three Dimensions*

In this section, we show that our results can be extended to three dimensions to parallelize both Chew's and Shewchuk's algorithm. Our analyses is similar to those for the two-dimensional parallel algorithms, with circles being replaced by spheres. We consider periodic point sets as input. So far, we have not completed the analysis for domains specified by PLCs, although we think similar results can be obtained. The definition of a periodic point set given in the previous section naturally extends to higher dimensions. The following generic parallel Delaunay algorithm is almost identical to its two dimensional counterpart.

Algorithm 8 Generic 3D Parallel Delaunay Refinement

Input: A periodic point set P in $\mathbb{R}^3$
  Let T be the Delaunay triangulation of P
  Compute $\dot{\mathcal{C}}$, the set of circumcenters of bad tetrahedra in T
  while $\dot{\mathcal{C}}$ is not empty **do**
    Let $\mathcal{I}$ be an independent subset of $\dot{\mathcal{C}}$
    Update Dleaunay triangulation inserting all points in $\mathcal{I}$ in parallel
    Update $\dot{\mathcal{C}}$
  end while

The following theorem extends Theorem 1. We omit the proof because it is identical to the 2D case, once circles and triangles are replaced by spheres and tetrahedra.

Theorem 7. *There exists an execution of the sequential Delaunay refinement algorithm that generates the same mesh as our 3D parallel Delaunay refinement algorithm.*

Lemma 1 also extends with no need of technical rewriting of the proof.

Lemma 11. *Let c_a be a circumsphere rejected by the parallel Delaunay refinement algorithm at iteration i. Then, there is a conflicting circumsphere c_b whose center is chosen as a Steiner point at iteration i and $r_b/2 \leq r_a \leq 2r_b$.*

5.2.1. *Parallelizing Chew's 3D refinement*

We suggest using a *maximal* independent subset of the candidate centers in Algorithm 8, to derive an efficient parallelization the Chew's 3D algorithm.

The following Lemma is an extension of Lemma 2 for 3D point sets. Let r_i be the radius of the largest Delaunay sphere at the end of iteration i.

Lemma 12. *For all i, $r_{i+1077} \leq 3r_i/4$.*

Proof. The proof is very similar to that of Lemma 2. For the sake of contradiction, suppose $r_k \geq \beta_C \bar{\mathbf{s}}$ and $r_k > \frac{3}{4}r_i$ for some i and k with $k - i = 1077$. Let c_k be the largest Delaunay sphere at the end of iteration iteration k. We dilate c_k, this time in four steps, to find a sphere c'_j, for each iteration j, where $i < j \leq k$. Again we show that $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$ and (ii) $r'_j \geq 3r_i/4$.

First three steps of the dilation is same as in our analysis in two dimensions. After we locate three points p_j, q_j, t_j, all exist in iteration j, we grow the sphere furthermore, keeping the touch points at p_j, q_j, and t_j fixed, and moving the center away from the plane of the triangle $p_jq_jt_j$ until it touches a vertex u_j. The resulting sphere c'_j is a circumsphere of a Delaunay tetrahedron $p_jq_jt_j$. Moreover, $p_jq_jt_j$ is a bad tetrahedron. So the circumcenter $\dot{c}'_j$ is a candidate vertex in iteration j.

To prove that $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$, consider the triangle $p_j\dot{c}_k\dot{c}'_j$. Due to construction, point p_j and $\dot{c}'_j$ are on the opposite side of the plane that goes through $\dot{c}_k$ and orthogonal to $\dot{c}_k\dot{c}'_j$. Hence, the angle $\angle p_j\dot{c}_k\dot{c}'_j$ is non-acute. The rest of the argument is same as in 2D (Lemma 2), resulting the same upper bound, $3r_i/4$.

The distance between $\dot{c}'_j$ and c_k implies that $\dot{c}'_j$ is rejected in iteration j of the parallel Chew's refinement; otherwise, c_k would not be vertex-free in iteration k. At each iteration we select a maximal independent subset of the candidates. So by Lemma 11 there is a center in conflict with $\dot{c}'_j$. Let this center be $\dot{c}''_j$. Again by Lemma 11, $\dot{c}''_j$ is chosen by the parallel algorithm for insertion and the radius of c''_j is at most twice and at least half the radius of c'_j.

Let $C'' = \{c''_{i+1}, c''_{i+2}, \ldots, c''_j, \ldots, c''_k\}$ be the set of spheres whose centers were inserted in iterations $j = i+1, \ldots, k$. By the same argument used in the proof of Lemma 2), we say that the spheres with radius $3r_i/16$ centered at the points in $\dot{C}'$ do not overlap. These spheres can spread to a volume of size $(4/3)\pi(7/4 + 3/16)^3r_i^3 - (4/3)\pi(3/4 - 3/16)^3r_i^3$. One can pack at most

$$\left\lfloor \frac{(4/3)\pi[(7/4+3/16)^3 - (3/4-3/16)^3]r_i^3}{(4/3)\pi(3/16)^3r_i^3} \right\rfloor = 1076$$

spheres of radius $3r_i/16$ in this region. This implies a contradiction. □

The proof of the following theorem is based on Lemma 12 and otherwise identical to the 2D version (Theorem 2).

Theorem 8. *For a given periodic point set P in $\mathbb{R}^3$ with diameter L, if the length of the shortest edge in the mesh generated by Chew's 3D refinement is $\boldsymbol{s}$, then the parallel Chew's 3D refinement takes $O(\log(L/\boldsymbol{s}))$ iterations to generate a size optimal well-shaped uniform mesh of the input.*

5.2.2. *Parallelizing Shewchuk's refinement*

For a given periodic point set, the sequential Shewchuk's algorithm generates a mesh with bounded circumradius to shortest edge ratio if $\beta_S \geq 1$. As in Ruppert's algorithm, we present our results for $\beta_S = \sqrt{2}$ to simplify our proofs. Our arguments for parallel Shewchuk's algorithm still hold for $\beta_R \geq 1 + \epsilon$ where ϵ is an arbitrarily small positive constant. As before, we classify the edges by length and associate tetrahedra with edges and edge classes. A tetrahedron is associated with an edge class $\mathcal{E}_i$ if its shortest edge is in $\mathcal{E}_i$. We suggest an algorithm almost identical to parallel Ruppert's algorithm except the triangles are replaced by tetrahedra. Hence, we omit the algorithm to avoid repetition. We refer this new algorithm as parallel Shewchuk's algorithm. Our analysis is also very similar to that of the two-dimensional version of this algorithm. The following lemma extends Lemma 4.

Lemma 13. *Suppose there the shortest edge associated with a bad tetrahedron is in $\mathcal{E}_i$. Let $e \in \mathcal{E}_i$ be an edge exists at the end of the j^{th} iteration of the inner loop. Let r_j be the circumcradius of the tetrahedron with the largest circumcircle among those associated with e. Then after* 1588 *iterations, one of the following is true*

(i) *e does not exist anymore;*
(ii) *all tetrahedra associated with e are good;*
(iii) *the largest circumsphere associated with e has radius $r_{j+1588} \leq 3r_j/4$.*

Proof. The proof is almost identical to that of Lemma 4. The difference is in the volume of the region in the packing argument. As in two dimensions, we prove that a center is inserted each at each iteration j through $j + 1588$. These centers are at least $3r_j/8$ apart from each other. The spheres corresponding to these centers can spread to a volume of size $(4/3)\pi(2r_j + 3r_j/16)^3$. One can fit at most

$$\left\lfloor \frac{(4/3)\pi(2r_j + 3r_j/16)^3}{(4/3)\pi(3r_j/16)^3} \right\rfloor = \left\lfloor \frac{(35/16)^3}{(3/16)^3} \right\rfloor = 1587$$

spheres in that volume. Therefore, the number of centers, $l - j$, is at most 1587, which is a contradiction. □

Following Lemma extend the Lemma 5 with an almost same, hence omitted, proof.

Lemma 14. *Suppose the shortest edge associated with a triangle is in $\mathcal{E}_i$. During and after the i^{th} iteration of the outer loop of the parallel Shewchuk's refinement algorithm, the following are true:*

(i) *if an edge $e \in \mathcal{E}_i$ disappears it never appears again;*
(ii) *no new edges are introduced to edge class $\mathcal{E}_i$;*
(iii) *the radius-edge ratio of a tetrahedron associated with an an edge $e \in \mathcal{E}_i$ does not increase.*

Theorem 9. *For a given periodic point set P in $\mathbb{R}^3$ of diameter at most L, if the length of the shortest edge in the mesh generated by Shewchuk's refinement is s, then parallel Shewchuk refinement takes $O(\log^2(L/s))$ iterations to generate a bounded radius-edge ratio mesh.*

Proof. We use Lemma 13 and 14 to prove this theorem. Otherwise, the proof is same as in Theorem 3. □

6. Discussions and Conclusions

Polylogarithmic upper bounds on the number of parallel iterations presented in Sections 4 and 5 constitutes the main component of the analyses of our parallel algorithms. At each iteration, our algorithms perform two main operations: i) compute a maximal independent set of points for parallel insertion; ii) update the Delaunay triangulation inserting all these points. For the first one, we proposed a new constant time parallel algorithm. For the second, we suggested to use an existing logarithmic time parallel Delaunay triangulation algorithm. These immediately imply polylogarithmic total time complexity for our parallel Delaunay refinement algorithms.

We opted for simplicity in our analyses. So, the constants in lemmas 2 and 4 are probably not optimal and likely to be much smaller in practice than 98 and 81.

The 3D extension of Chew's and Shewchuk's algorithms do not always guarantee that the resulting mesh has an aspect-ratio bounded by a constant. However, they both guarantee a constant bound on the ratio of the circumradius to the length of the shortest edge (the radius-edge ratio) of any tetrahedra in the final mesh. So, the meshes these two algorithms generate might potentially contains slivers, which are elements with close to zero aspect-ratio but with a constant radius-edge ratio. Several quality enhancing and guaranteeing meshing algorithms have been developed recently. We summarize them below and also comment on their parallel complexity.

There are two classes of new algorithms. The first class modifies the rule for choosing new points and provides a guarantee on the aspect-ratio of the resulting mesh. Chew[9] proposed to use randomization to add points close to the circumcenters of bad tetrahedra, and sketched a proof that the aspect ratio in the resulting uniform mesh is bounded. Li and Teng[27] developed a Delaunay refinement algorithm that applied Chew's idea to Shewchuk's algorithm, and proved that their Delaunay refinement algorithm guarantees the aspect ratio. Our framework can be used to analyze parallel complexity of these methods.

The second class of new algorithms transform a Delaunay mesh with bounded radius-edge ratio into another mesh with bounded aspect-ratio. The meshes produced by these algorithms might still have slivers on the boundary. The first such algorithm, developed by Cheng *et al.*,[7] used weighted Delaunay triangulation. A second algorithm of Edelsbrunner *et al.*[18] applies perturbations to vertices. Both of these transformation algorithms can be implemented in $O(\log n)$ time using K processors where K is the mesh size.

We conclude the paper with two conjectures.

- There is a parallel implementation of Ruppert's[42] and Shewchuk's[49] algorithm that runs in $O(\log(L/s))$ iterations.
- There is a parallel Ruppert's[42] and Shewchuk's[49] algorithm that runs in $O(\log n)$ time where n is the input complexity. Notice that Bern *et al.*[4] showed that the quadtree algorithm can be implemented in $O(\log n)$ time with K processors.

We would also like to see results that establish the parallel complexity of other mesh generation algorithms such as sink insertion.[17]

7. Acknowledgments

We thank Jeff Erickson, Sariel Har-Peled and Dafna Talmor for helpful conversations and comments on the paper. We also thank Jonathan Shewchuk for helpful emails about boundary assumptions and boundary preprocessing for Ruppert's algorithm.

First author was supported by an Alfred P. Sloan Foundation Fellowship, and NSF award CCR-0112487. The second author was supported by an Alfred P. Sloan Foundation Fellowship. This research was done when the third author was a Ph.D. student at the University of Illinois at Urbana-Champaign, where he was supported by a Computational Science and Engineering Fellowship.[55]

References

1. N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. *Proc. 35th IEEE Symp. Foundations of Computer Science*, 683–694, 1994.
2. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean Geometry* 23–90. World Scientific, 1992.
3. M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comp. System Sciences* 48:384–409, 1994.
4. M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *Int. J. Computational Geometry & Applications* 9(6):517–532, 1999.
5. G. E. Blelloch, J. C. Hardwick, G. L. Miller, and D. Talmor. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica* 24(3):243–269, 1999.
6. G. E. Blelloch and B. M. Maggs. Parallel algorithms. *Handbook of Algorithms and Theory of Computation*, chapter 47, 1998. CRC Press.
7. S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *Proc. 15th ACM Symp. Comp. Geometry*, 1–13, 1999.
8. L. Chew. Guaranteed-quality triangular meshes. Tech. Rep. TR-89-983, Cornell University, Ithaca, 1989.
9. L. Chew. Guaranteed-quality Delaunay meshing in 3d (short version). *Proc. 13th ACM Symp. Comp. Geometry*, 391–393, 1997.
10. L. Chew, N. Chrisochoides, and F. Sukup. Parallel constrained Delaunay meshing. *Proc. Symp. on Trends in Unstructured Mesh Generation*, 89–96, 1997.
11. L. P. Chew. Constrained Delaunay triangulations. *Algorithmica* 4:97–108, 1989.
12. N. Chrisochoides. New approach to parallel mesh generation and partitioning problem. *Computational Science, Mathematics and Software* 319–344, 2001.
13. N. Chrisochoides and D. Nave. Simultaneous mesh generation and partitioning for Delaunay meshes. *Proc. 8th Int. Meshing Roundtable*, 55–66, 1999.
14. K. Clarkson and P. W. Shor. Applications of random sampling in computational geometry II. *Discrete & Computational Geometry* 4(1):387–421, 1989.
15. D. Cohen-Steiner, E. C. de Verdiére, and M. Yvinec. Conforming Delaunay triangulations in 3d. *Proc. 18th ACM Symp. Comp. Geometry*, 199–208, 2002.
16. H. Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge University Press, 2001.
17. H. Edelsbrunner and D. Guoy. Sink insertion for mesh improvement. *Proc. 17th ACM Symp. Comp. Geom.*, 115–123, 2001.
18. H. Edelsbrunner, X.-Y. Li, G. L. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng,

A. Üngör, and N. Walkington. Smoothing and cleaning up slivers. *Proc. 32nd Annual ACM Symposium on the Theory of Computing*, 273–277, 2000.
19. H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *Disc. and Comp. Geometry* 10:197–213, 1993.
20. S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2:153–174, 1987.
21. S. Fortune. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean Geometry* . World Scientific, 1992.
22. T. L. Freeman and C. Phillips. *Parallel Numerical Algorithms*. Prentice Hall, 1992.
23. L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica* 7:381–413, 1992. Also in *Proc. 17th Int. Colloq. Automata, Languages and Programming*, Springer-Verlag, LNCS 443, 1990, pp. 414–431.
24. M. T. Jones and P. E. Plassmann. Parallel algorithms for adaptive mesh refinement. *SIAM Journal on Scientific Computing* 18(3):686–708, 1997. ⟨`citeseer.nj.nec.com/article/jones97parallel.html`⟩.
25. R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. *Handbook of Theoretical Computer Science*, 869–942, 1990. MIT Press.
26. X.-Y. Li and S.-H. Teng. Dynamic load balancing for parallel adaptive mesh refinement on unstructured meshes. *Proc. 5th International Symposium on: Solving Irregularly Structured Problems in Parallel*, 144–155, 1998. Springer-Verlag, LNCS 1457.
27. X.-Y. Li and S.-H. Teng. Generate sliver free three dimensional meshes. *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, 28–37, 2001.
28. R. Löhner and J. R. Cebral. Parallel advancing front grid generation. *Proc. 8th International Meshing Roundtable*, 67–74, 1999. ⟨`http://www.andrew.cmu.edu/user/sowen/abstracts/Lo680.html`⟩.
29. M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing* 15(4):1036–1053, 1986.
30. G. L. Miller. A timing analysis of a Delaunay refinement mesh generation algorithm. *in preparation* , 2003.
31. G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. *Proc. 27th Annual ACM Symp. on the Theory of Computing*, 683–692, 1995.
32. G. L. Miller, D. Talmor, S.-H. Teng, N. Walkington, and H. Wang. Control volume meshes using sphere packing: generation, refinement, and coarsening. *Proc. 5th Int. Meshing Roundtable*, 47–61, 1996. ⟨`http://www.andrew.cmu.edu/user/sowen/abstracts/Mi242.html`⟩.
33. S. Mitchell and S. Vavasis. Quality mesh generation in three dimensions. *Proc. 8th ACM Symp. Comp. Geom.*, 212–221, 1992.
34. M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralization. *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, 67–74, 2000.
35. D. Nave, N. Chrisochoides, and P. Chew. Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains. *Proc. 18th ACM Symp. Comp. Geometry*, 135–144, 2002.
36. D. T. Nguyen. *Parallel-Vector Equation Solvers For Finite Element Engineering Applications*. Kluwer Academic / Plenum Publishers, 2002.
37. T. Okusanya and J. Peraire. Parallel unstructured mesh generation. *Proc. 5th Int. Conf. on Numerical Grid Generation*, 719–729, 1996. ⟨`http://www.andrew.cmu.edu/user/sowen/abstracts/Ok203.html`⟩.

38. C. Özturan. Worst case complexity of parallel triangular mesh refinement by longest edge bisection. Tech. Rep. 96-56, NASA Langley Research Center, Hampton, VA, 1996. ⟨`http://www.icase.edu/library/reports/rdp/96/96-56RDP.tex.refer.html`⟩.
39. F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. Assoc. Comput. Mach.* 20:87–93, 1977.
40. J. H. Reif and S. Sen. Parallel computational geometry: An approach using randomization. *Handbook of computational geometry*, 765–828, 1999. Elsevier Science Publishing, Amsterdam.
41. M.-C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Int. J. Numer. Meth. Eng.* 20:745–756, 1984.
42. J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. *Proc.* 4*th ACM-SIAM Symp. on Disc. Algorithms*, 83–92, 1993.
43. A. Saalfeld. Delaunay edge refinements. *Proc. 3rd Canadian Conf. Comp. Geometry*, 33–36, 1991.
44. R. Seidel. Backwards analysis of randomized geometric algorithms. *New Trends in Discrete and Computational Geometry*, 37–68, 1993. Springer-Verlag.
45. M. I. Shamos and D. Hoey. Closest-point problems. *Proc. 16th IEEE Symp. Foundations of Comp. Science*, 151–162, 1975.
46. M. S. Shephard, J. E. Flaherty, C. L. Bottasso, H. L. de Cougny, C. Ozturan, and M. L. Simone. Parallel automatic adaptive analysis. *Parallel Computing* 23(9):1327–1347, 1997.
47. J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. *Proc. First Workshop on Applied Computational Geometry*, 124–133, 1996.
48. J. R. Shewchuk. *Delaunay refinement mesh generation*. Ph.D. thesis, Carnegie Mellon, 1997. ⟨`http://reports-archive.adm.cs.cmu.edu/anon/1997/abstracts/97-137.html`⟩.
49. J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. *Proc.* 14*th Annual ACM Symposium on Computational Geometry*, 86–95, 1998.
50. J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications* 22(1–3):21–74, 2002.
51. D. A. Spielman and S.-H. Teng. Models and applications of smoothed analysis in geometry, numerical analysis and property testing. *in preparation* , 2003.
52. D. A. Spielman, S.-H. Teng, and A. Üngör. Parallel Delaunay refinement: Algorithms and analyses. *Proc.* 11*th Int. Meshing Roundtable*, 205–217, 2002.
53. S.-H. Teng and C. W. Wong. Unstructured mesh generation: Theory, practice, and perspectives. *Int. J. Computational Geometry & Applications* 10(3):227–266, 2000.
54. B. H. V. Topping and A. I. Khan. *Parallel Finite Element Computations*. Saxe-Coburg Publ., Edinburgh, 1995.
55. A. Üngör. *Parallel Delaunay refinement and space-time meshing*. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2002. ⟨`http://www.cs.uiuc.edu/Dienst/UI/2.0/Describe/ncstrl.uiuc_cs/UIUCDCS-R-2002-2266`⟩.