PARALLEL DELAUNAY REFINEMENT: ALGORITHMS AND ANALYSES

Daniel A. Spielman[1] Shang-Hua Teng[2] Alper Üngör[3]

[1] *Dept. of Mathematics, Massachusetts Institute of Technology, spielman@math.mit.edu*

[2] *Dept. of Computer Science, Boston University and Akamai Technologies Inc., steng@cs.bu.edu*

[3] *Dept. of Computer Science, Duke University, ungor@cs.duke.edu*

ABSTRACT

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

1. INTRODUCTION

Delaunay refinement is a popular and practical technique for generating well-shaped unstructured meshes [19, 29, 34]. 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. 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 [29] 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 [34] and Li and Teng [19]. Efficient sequential Delaunay refinement software has been developed both in two [29, 33] and three dimensions [34]. Chrisochoides and Nave [9] and Okusanya and Peraire [27] developed parallel software using Delaunay refinement, for which they have reported good performance. Recently, Nave *et al.* [26] presented a parallel Delaunay refinement algorithm and proved that it produces well-shaped 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 in 2D and Shewchuk's 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 well-shaped and have 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, the number of iterations can be reduced to $O(\log(L/s))$.

We should emphasize here that our analysis focuses 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 2D we can use the divide-and-conquer parallel algorithm developed by Blelloch *et al.* [4] 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 [28], 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

This work is motivated by the observation that both sequential and parallel implementations of Delaunay refinement algorithms seem to produce the best meshes in practice. However, improvements in the speed of parallel numerical solvers are creating the need for comparable speedups in meshing software: Löhner and Cebral [20] have reported that improvements in parallel numerical solvers [37] have resulted in the simulation time of numerous practical systems being dominated by the meshing process.

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 [2, 24]. 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.

Several parallel mesh generation algorithms have been developed. On the theoretical extreme, Bern, Eppstein and Teng [3] 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 [32, 36]. One can easily show that this level-by-level based method takes $O(\log(L/s) + K/p)$ parallel time, using p processors [36].

Building upon [3], Miller *et al.* [22] 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 [21] 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.

Various parallel Delaunay refinement methods have been implemented and been seen to have good performance [9, 18, 20, 27]. 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 omit the proofs of Lemmas 8, 9, 10, 16,and 18 and Theorems 13, 14, 21, and 22 in this version due to page limitation. A full version of the paper is available at `http://www.cs.duke.edu/~ungor/abstracts/parallelDelRef.html`.

2. PRELIMINARIES

2.1 Input Domain

In 2D, the input domain Ω is represented as a *planar straight line graph* (PSLG) [29] — 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.* [23] 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 2D [10, 17, 16] and in $O(n^{\lceil d/2 \rceil})$ time in d dimensions [10, 31]. A nice survey of these algorithms can be found in [16].

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 developed an algorithm for computing constrained Delaunay triangulations [8].

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.

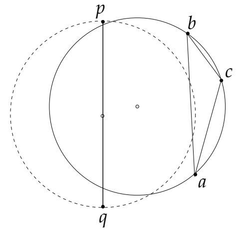

Figure 1. Circumcenter of triangle abc encroaches the segment pq.

In 2D, Edelsbrunner and Tan proved that $O(n^3)$ additional points are sufficient to generate a conforming triangulation of a PSLG of complexity n [15]. A 2D solution proposed by Saalfeld [30] is extended to 3D by Murphy *et al.* [25] and Cohen-Steiner *et al.* [11]. 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. 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 well-shaped conforming triangulations whose size is close to optimal both in two [6, 29] and three dimensions [19].

3. 2D SEQUENTIAL DELAUNAY REFINEMENT

In this section, we recall Ruppert's and Chew's algorithms for constructing Delaunay meshes of PLSGs in 2D. Following Ruppert [29], we assume that the angle between two adjacent input segments is at least $\pi/2$. Boundary treatments that relax this assumption are discussed in [29, 35].

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 1.)

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. 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 $\mathcal{B}^{(i)}$, $\mathcal{C}^{(i)}$, $\mathcal{D}_T^{(i)}$, and $\mathcal{D}_{\mathcal{B}}^{(i)}$;

while $\mathcal{C}^{(i)} \cup \mathcal{D}_T^{(i)} \cup \mathcal{B}^{(i)}$ 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

The points inserted by the Delaunay refinement are often called *Steiner points*.

If a quasi-uniform mesh, such as that produced by Chew's method, is desired [6], 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 2 shows the output of the Delaunay refinement illustrating the difference 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 [6, 29], 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

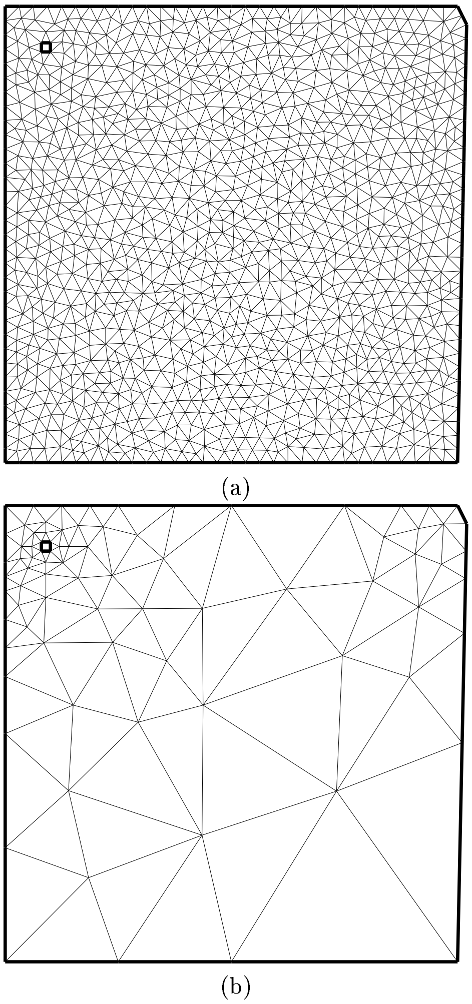

Figure 2. The output of (a) Chew's and (b) Ruppert's algorithm on the same input. Both of these meshes have minimum angle $> 29°$. The first mesh has 2246 and the second has 131 elements.

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 [29] 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 and Chew's [6, 29] 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 [29] and [35]. 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 we can show (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 2D DELAUNAY REFINEMENT

To better illustrate our analysis of parallel Delaunay refinement, we first focus on the case in which the input is a periodic point set (PPS) as introduced by Cheng *et al.* [5]. See also [12]. We will then extend our results to produce boundary conforming meshes when the input domain is a PSLG.

4.1 Input Domain: 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 [12]. 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.

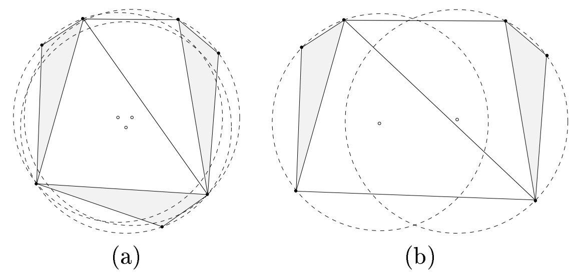

Figure 3. Circumcenters of bad triangles (shaded) are in conflict. (a) they are too close to each other; (b) they are not too close but can not be both inserted by a sequential algorithm.

As P is contained in the unit square, the diameter of P is $L \leq \sqrt{2}$. When we refer to the diameter of a periodic point set, we will mean the diameter of P.

4.1.1 A generic parallel algorithm (PPS)

For a periodic point set, the only candidates for insertion are the circumcenters of poorly shaped triangles. We need a rule for choosing a large subset of the candidates with the property that a sequential Delaunay refinement algorithm would insert each of the points in the subset. 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* conflicting *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 (MIS) 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.

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 $\dot{\mathcal{C}}$
    Insert all the points in $\mathcal{I}$ in parallel
    Update T and $\dot{\mathcal{C}}$
  end while

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 can be sequentialized. This implies that the algorithm inherits the guarantee of its sequential counterpart that it generates a well-shaped mesh of size that is within a constant factor of optimal.

Theorem 2. *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 $i < j$. For each independent set $\mathcal{I}_i$, we insert the candidates according to their circumradius in the order from largest to smallest. For any two circumcenters $\dot{a}, \dot{b} \in \mathcal{I}_i$, assume that the radius of a is larger than the radius of b. This implies that $\dot{a}$ can not be in the circumcircle of $\dot{b}$, because $\dot{a}$ and $\dot{b}$ are independent. Therefore, the insertion of $\dot{a}$ will not eliminate the triangle of $\dot{b}$.

Furthermore, observe that in any sequential execution, the insertion of point $\dot{p} \in \mathcal{I}_i$ can not eliminate the triangle corresponding to $\dot{q} \in \mathcal{I}_j$ for any $i < j$, for otherwise, $\dot{q}$ would not exist in the j^{th} iteration of the parallel execution.

Therefore, the parallel and sequential executions terminate with the same Delaunay mesh. □

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

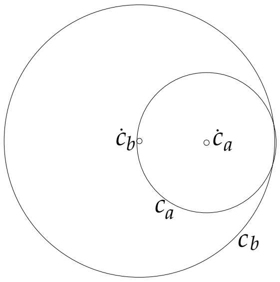

Figure 5. The larger of two conflicting circumcenters can be at most twice as large as the smaller.

Proof: Circumcircle c_a contains $\dot{c}_b$. (See Figure 5). 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 (PPS)

In this section, we show that our parallel implementation of Chew's refinement only needs $O(\log(L/s))$ iterations. The basic argument is very simple — we will show that the radius of the largest Delaunay circle reduces by a factor of 3/4 after some constant number (e.g., 98) of iterations. Because the largest circumradius initially is $O(L)$ and the largest circumradius in the final mesh is $\Omega(s)$, the iteration bound of $O(\log(L/s))$ follows immediately.

Lemma 4. *For all i, let r_i be the largest circumradius of a triangle in the Delaunay triangulation at the end of iteration i. For all $k \geq 98$, $r_k \leq 3r_{k-98}/4$.*

Proof: We assume by way of contradiction that $r_k > 3r_{k-98}/4$. Let $i = k - 98$. Let c_k be a circumcircle with radius r_k after iteration k. Let $\dot{c}_k$ be the center of c_k.

For $j \leq k$, it is clear that c_k is also an empty circle in iteration j, because the refinement process only adds new points. But, c_k might not be a circumcircle at iteration j. We now show that for each iteration j, where $i \leq j \leq k$, there exists a circumcircle c'_j with center $\dot{c}'_j$, and radius r'_j such that (1) $||\dot{c}'_j - \dot{c}_k|| \leq 3r_i/4$ and (2) $r'_j \geq 3r_i/4$.

Let p_k, q_k and t_k be the vertices of the Delaunay triangle at iteration k that defines c_k. (See Figure 4 (a)). We will alter c_k in three stages to produce a suitable c'_j that is the circumcircle of three points that exist at stage j: p_j, q_j and t_j.

i. Dilate c_k until it touches a mesh point p_j (Figure 4 (b)). Note that p_j might well be p_k, q_k, or t_k, so, c_k might not actually expand at all during this step.
ii. Grow the circle by moving its center away from p_j along the ray $\overrightarrow{p_j \dot{c}_k}$, and maintaining the property that p_j lies on the boundary of the circle, until it touches a mesh point q_j (Figure 4 (c)).
iii. Continue to grow the circle, maintaining its contact with p_j and q_j, moving its center away from the chord $\overline{p_j q_j}$, until it touches a vertex t_j (Figure 4 (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 poorly-shaped triangle because its circumradius r'_j is at least r_k. Thus, its center $\dot{c}'_j$ is a candidate at iteration j. Note also that $r'_j \geq r_k \geq 3r_i/4$.

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 $y = |\dot{c}_k \dot{c}'_j|$. Since $|\angle \dot{c}'_j \dot{c}_k p_j|$ is non-acute $(r'_j)^2 \geq x^2 + y^2$. As r'_j is the radius of a Delaunay triangle and $j \geq i$, $r'_j \leq r_i$. Combining this fact with $x \geq r_k > 3r_i/4$, we find $r'_j < x + r_i/4$. So we

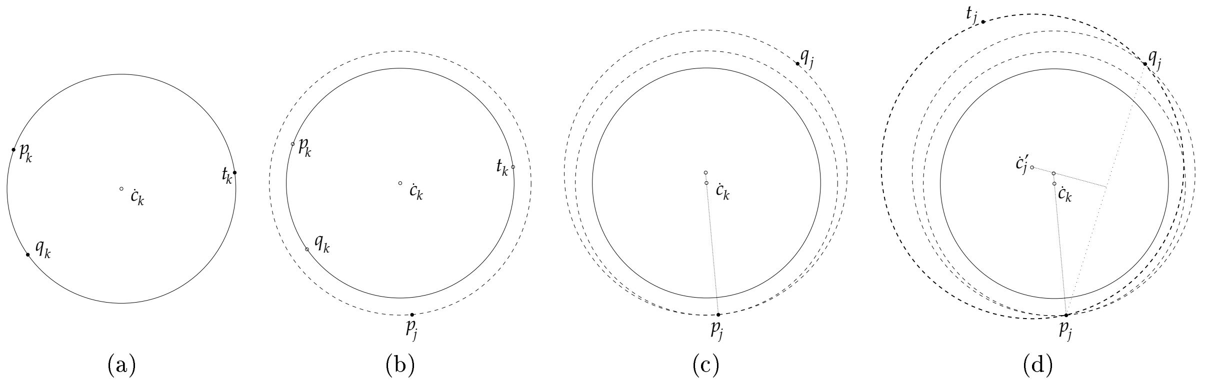

Figure 4. Dilate the largest Delaunay circle of iteration k (a) until it touches a vertex p_j existed in iteration j (b). Grow the circle further so that it touches two more vertices q_j and t_j existed in iteration j ((c) and (d)).

can write, $(x+r_i/4)^2 \geq x^2+y^2$. By simplifying this inequality to $xr_i/2+r_i^2/16 \geq y^2$ and substituting $x \leq r_i$, we derive $9r_i^2/16 \geq y^2$. Hence, $y = ||\dot{c}_j' - \dot{c}_k|| \leq 3r_i/4$.

Because c_j' is empty at the end of iteration j, we know $\dot{c}_j'$ was not chosen during iteration j. Because the independent set of candidates that we select is maximal, there must be another circumcircle c_j'' chosen in iteration j that conflicts with c_j'. By Lemma 3, the radius of c_j'' is at least one half of the radius of c_j', and so is at least $|r_j'|/2 \geq 3r_i/8$. Moreover, the radius of c_j'' is at most r_j' and hence at most r_i. So $||\dot{c}_j'' - \dot{c}_j'|| \leq r_i$. Hence,

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

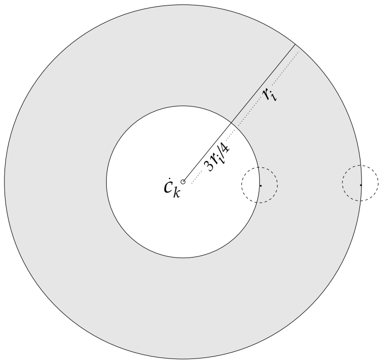

Figure 6. Packing non-overlapping disks (dashed) with radius $3r_i/16$ and centers located inside the shaded region.

Let $\dot{C}'' = \{\dot{c}_{i+1}'', \dot{c}_{i+2}'', \ldots, \dot{c}_j'', \ldots, \dot{c}_k''\}$, and let C'' be the corresponding set of circumcircles. As $\dot{c}_l''$ is inserted during round l for each l, each circle $c_j'' \in C''$ is empty of all the centers $\dot{c}_l''$ for $l < j$. So the centers in $\dot{C}''$ are pairwise at least $3r_i/8$ away from each other. Thus, one can draw disjoint circles of radius $3r_i/8$ around each of these points. The annulus containing these disjoint circles has area at most $\pi(7/4 + 3/16)^2 r_i^2 - \pi(3/4 - 3/16)^2 r_i^2$. See Figure 6. So, one can pack at most

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

disjoint circles of radius $3r_i/16$ in this region. This implies $|C''| = k - i \leq 97$, a contradiction. $\square$

Theorem 5. *Our parallel implementation of Chew's refinement algorithm takes at most* $98 \left\lceil \log_{4/3}(L/s) \right\rceil$ *iterations.*

4.1.3 Parallel Computation of MIS

One can use the parallel maximal independent set algorithm of Luby [21] 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 3 that two circumcircles are conflicting at iteration j only if their radii are within a factor of 2 of each other.

Lemma 6. *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 $(x \mod 3, y \mod 3)$. 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 (PPS)

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 circumradius 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 $\sqrt{2}h$.

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 $\lceil \log_{\sqrt{2}}(L/s) \rceil$ **do**
    Let $\dot{\mathcal{C}}$ be the set of all circumcenters of poorly-shaped triangles who are in class $\mathcal{E}_i$
    while $\dot{\mathcal{C}}$ is not empty **do**
      Let $\mathcal{I}$ be a maximal independent subset of $\dot{\mathcal{C}}$
      Insert all the points in $\mathcal{I}$ in parallel
      Update the Delaunay triangulation and $\dot{\mathcal{C}}$
    end while
  end for

Let s be the length of the shortest edge in the initial Delaunay triangulation. At each iteration, we assign an edge to class $\mathcal{E}_i$ if its length is in $\left[\sqrt{2}^{i-1}s, \sqrt{2}^i s\right)$. Similarly, we assign a Delaunay triangle to $\mathcal{E}_i$ if its shortest edge has length in $\left[\sqrt{2}^{i-1}s, \sqrt{2}^i s\right)$. There are at most $\lceil \log_{\sqrt{2}}(L/s) \rceil$ of such classes. Using this definition, we can state and analyze the Parallel Ruppert's Refinement Algorithm.

Theorem 7. *Given a periodic point set in 2D of diameter L, the Parallel Ruppert's Refinement Algorithm takes $O(\log^2(L/s))$ iterations.*

Proof: Lemmas 8 and 9 prove that after the ith iteration of the outer loop, each Delaunay triangle touching an edge in class $\mathcal{E}_i$ will be well-shaped, and successive iterations cannot degrade the shape of the Delaunay triangles touching that edge. Lemma 10 implies that during each iteration of the outer loop, the inner loop of the algorithm will execute at most $O(\log(L/s))$ times. As the outer loop is executed $O(\log(L/s))$ times, the whole algorithm takes at most $O(\log^2(L/s))$ iterations. □

Lemma 8. *During the ith iteration of the outer loop of the Parallel Ruppert's Refinement Algorithm, no Delaunay edges are added to or removed from class $\mathcal{E}_i$.*

Lemma 9. *Suppose e is an edge in $\mathcal{E}_j$ where $j \leq i$. Then during the ith outer loop, the radius-edge ratio of triangles containing e does not increase.*

Lemma 10. *Let $e \in \mathcal{E}_i$, and let r_l be the radius of the larger of the two circumcircles containing e at the end of the l^{th} iteration of the inner loop during the ith iteration of the outer loop. Then, at the end of iteration $k = l + 81$ of the inner loop, either (1) both Delaunay triangles containing e are well-shaped, or (2) $r_k \leq 3r_l/4$ where r_k is the radius of the larger of the two circumcircles containing e after iteration k.*

4.2 Input Domain: PSLG

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 give, in Section 4.2.3, an $O(\log L/s)$ time parallel preprocessing algorithm to insert points to 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 candidate 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 (PSLG)

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 $\mathcal{D}$ be 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 overlapping. Every two diametral centers from $\dot{\mathcal{B}}$ are independent.

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 11. *A circumcenter $\dot{c} \in \dot{\mathcal{C}}$ and a diametral center $\dot{d} \in \dot{\mathcal{D}}$ are* conflicting *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 [29].

Lemma 12. *If a circumcircle c of radius r_c encroaches a diametral circle d of radius r_d, then $r_d \geq r_c/\sqrt{2}$.*

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 $(\dot{\mathcal{BC}} \cap \dot{\mathcal{C}}) \cup \dot{\mathcal{D}}$
    Insert all the points in $\mathcal{I}$ in parallel
    Update the Delaunay triangulation
    Update $\dot{\mathcal{B}}$, $\dot{\mathcal{C}}$, $\dot{\mathcal{BC}}$ and $\dot{\mathcal{D}}$
  end while

The following theorem extends Theorem 2 for domains given by PSLGs.

Theorem 13. *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.*

4.2.2 Parallelizing Chew's Refinement (PSLG)

To parallelize Chew's algorithm for domain defined by a PSLG, we apply Algorithm 4 and use a maximal independent set of the candidates at each iteration. In addition, because each pair of diametral centers in $\dot{\mathcal{D}}$ is independent, we include all centers $\dot{\mathcal{D}}$ in the independent set. The parallel algorithm of Section 4.1.3 can be used to construct the maximal independent set.

Theorem 14. *Our parallel implementation of Chew's refinement algorithm takes $O(\log(L/s))$*

iterations for a domain given by a PSLG, where L is the diameter of the domain and s is smallest local feature size.

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 15 (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.

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

Lemma 16. *Parallel Boundary Preprocessing terminates in $O(\log(L/s))$ iterations.*

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 in [2, 3], one can use balanced quadtree to approximate local feature size function of Ω to within a constant factor. Therefore, using a balanced quadtree as [2, 3], we can preprocess the domain in $\log(L/s)$ parallel time so that the preprocessed domain is *strongly feature conforming* as defined below.

Definition 17 (Strongly Feature Conforming). *Let $\alpha \geq 2$ 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.*

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

4.2.4 Parallelizing Ruppert's Refinement (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.

Let s be smallest local feature of Ω. At each iteration, we assign an edge to class $\mathcal{E}_i$ if its length is in $\left[\sqrt{2}^{i-1}s, \sqrt{2}^{i}s\right)$. Similarly, we assign a Delaunay triangle to $\mathcal{E}_i$ if its shortest edge has length in $\left[\sqrt{2}^{i-1}s, \sqrt{2}^{i}s\right)$. There are at most $\lceil \log_{\sqrt{2}}(L/s) \rceil$ of such classes.

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 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 maximal independent subset of $(\dot{\mathcal{BC}} \cap \dot{\mathcal{C}}) \cup \dot{\mathcal{D}}$
   Insert all the points in $\mathcal{I}$ in parallel
   Update the Delaunay triangulation
   Update $\dot{\mathcal{B}}$, $\dot{\mathcal{C}}$, $\dot{\mathcal{BC}}$ and $\dot{\mathcal{D}}$
  end while
 end for

Theorem 19. *Given a domain specified by a PSLG, the Parallel Ruppert's Refinement Algorithm takes $O(\log^2(L/s))$ iterations.*

The proof of Theorem 19 is essentially the same as the proof of Theorem 7 where we need to address the following two issues.

1. The center of a circumcircle could potentially encroach a boundary segment whose length is much larger than 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 within a constant factor of each other. In addition, because each boundary segment can only be split at most a constant times in the refinement, it can not introduce smaller edges too many times.

5. 3D DELAUNAY REFINEMENT

A 3D domain is specified by a PLC (see Section 2.1). In this section, we assume that the angle between any two intersecting elements, when one is not contained in the other, is at least $90°$. There are three kinds of spheres associated with a 3D Delaunay mesh that we are interested: the circumspheres, the diametral spheres, and the equatorial sphere given below.

Definition 20. *The* equatorial sphere *of a triangle in 3D is the smallest sphere that passes through its vertices. A triangular subfacet of a PLC is* encroached *if the equatorial sphere is not empty.*

Chew's algorithm extends naturally to 3D. In [34], Shewchuk developed a 3D extension of Ruppert's algorithm. In Shewchuk's refinement, given below, 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 $\beta_S \geq 2$.

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 $\dot{C} \cup \dot{D} \cup \dot{E}$ is not empty **do**
    Insert a and update the Delaunay triangulation
    Update $\dot{C}$, $\dot{D}$, and $\dot{E}$
  end while

5.1 Parallel 3D Delaunay Refinement

In this subsection, we show that our results for a domain given by a periodic point set can be extended from two dimensions to three dimensions to parallelize both Chew's and Shewchuk's algorithm. So far, we have not completed the analysis for domains specified by PLCs, although we think similar results can be obtained.

The following is a parallel Delaunay refinement algorithm for domains specified by 3D periodic point sets.

Algorithm 8 Generic Parallel 3D 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 $\dot{\mathcal{C}}$
    Insert all the points in $\mathcal{I}$ in parallel
    Update T and $\dot{\mathcal{C}}$
  end while

The following theorem is analogous to Theorem 2. Its proof uses a straight-forward extension of Lemma 3.

Theorem 21. *Suppose M is a mesh produced by an execution of the Generic Parallel 3D Delaunay Refinement algorithm. Then M can be obtained by some execution of the sequential Delaunay refinement algorithm.*

To parallelize Chew's 3D refinement, we use a *maximal* independent set of candidate centers in Algorithm 8. With almost the same proof as we have presented in Section 4.1.2, we can show that the number of iterations needed is at most $1077 \left\lceil \log_{4/3}(L/s) \right\rceil$.

5.1.1 Parallelizing Shewchuk's Refinement

Our parallel implementation of Shewchuk's algorithm is analogous to our parallel implementation of Ruppert's algorithm. We assign a Delaunay tetrahedron to class $\mathcal{E}_i$ if its shortest edge has length in $\left[\sqrt{2}^{i-1}s, \sqrt{2}^{i}s\right)$. Then, we consider the classes in ascending order of indices. For each class, we repetitively insert a maximal independent set of circumcenters of bad tetrahedra until no bad tetrahedron is left in it.

Our analysis of parallel Shewchuk's refinement is also analogous to the analysis of parallel Ruppert's refinement. Slightly modifying the proof of Lemma 10, we can show that all the bad tetrahedra in each edge class is removed in at most $1588 \left\lceil \log_{4/3}(L/s) \right\rceil$ iterations. This leads to the following theorem.

Theorem 22. *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's refinement takes $O(\log^2(L/s))$ iterations to generate a bounded radius-edge ratio mesh.*

6. DISCUSSION

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 4 and 10 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 [5, 7, 14, 19] have been developed recently. Cheng *et al.* [5] and Edelsbrunner *et al.* [14] have already given parallel complexity of their sliver removal algorithms. Our framework can be used to analyze parallel complexity of the other two algorithms, by Chew [7] and Li and Teng [19].

We conclude the paper with two conjectures.

- There is a parallel implementation of Ruppert's [29] and Shewchuk's [34] algorithm that runs in $O(\log(L/s))$ iterations.
- There is a parallel Ruppert's [29] and Shewchuk's [34] algorithm that runs in $O(\log n)$ time where n is the input complexity. Notice that Bern *et al.* [3] 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 [13].

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.

The 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.

References

[1] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. *Proc. 35th IEEE Symp. Found. of Comp. Sci.*, 683–694, 1994.

[2] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comp. System Sciences* 48:384–409, 1994.

[3] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *Int. J. Comp. Geom. & Appl.* 9(6):517–532, 1999.

[4] 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.

[5] 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.

[6] L. Chew. Guaranteed-quality triangular meshes. Tech. Rep. TR-89-983, Cornell University, Ithaca, 1989.

[7] L. Chew. Guaranteed-quality Delaunay meshing in 3D (short version). *Proc. 13th ACM Symp. Comp. Geometry*, 391–393, 1997.

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

[9] N. Chrisochoides and D. Nave. Simultaneous mesh generation and partitioning for Delaunay meshes. *Proc. 8th Int. Meshing Roundtable*, 55–66, 1999.

[10] K. Clarkson and P. W. Shor. Applications of random sampling in computational geometry II. *Discrete & Computational Geometry* 4(1):387–421, 1989.

[11] D. Cohen-Steiner, E. C. de Verdiére, and M. Yvinec. Conforming Delaunay triangulations in 3d. *Proc. 18th ACM Symp. Comp. Geometry*, 199–208, 2002.

[12] H. Edelsbrunner. *Geometry and topology for mesh generation.* Cambridge University Press, 2001.

[13] H. Edelsbrunner and D. Guoy. Sink insertion for mesh improvement. *Proc. 17th ACM Symp. Comp. Geom.*, 115–123, 2001.

[14] 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 ACM Symp. on the Theory of Computing*, 273–277, 2000.

[15] H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *Disc. and Comp. Geometry* 10:197–213, 1993.

[16] S. Fortune. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean Geometry* . World Scientific, 1992.

[17] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica* 7:381–413, 1992.

[18] X.-Y. Li and S.-H. Teng. Dynamic load balancing for parallel adaptive mesh refinement on unstructured meshes. *Proc. 5th IRREGULAR*, 144–155, 1998. Springer-Verlag, LNCS 1457.

[19] X.-Y. Li and S.-H. Teng. Generate sliver free three dimensional meshes. *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, 28–37, 2001.

[20] R. Löhner and J. R. Cebral. Parallel advancing front grid generation. *Proc. 8th Int. Meshing Roundtable*, 67–74, 1999.

[21] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing* 15(4):1036–1053, 1986.

[22] 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 ACM Symp. on the Theory of Computing*, 683–692, 1995.

[23] 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.

[24] S. Mitchell and S. Vavasis. Quality mesh generation in three dimensions. *Proc. 8th ACM Symp. Comp. Geom.*, 212–221, 1992.

[25] 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.

[26] 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.

[27] T. Okusanya and J. Peraire. Parallel unstructured mesh generation. *Proc. 5th Int. Conf. on Numerical Grid Generation*, 719–729, 1996.

[28] J. H. Reif and S. Sen. Parallel computational geometry: An approach using randomization. *Handbook of computational geometry*, 765–828, 1999. Elsevier Science Publishing, Amsterdam.

[29] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. *Proc. 4th ACM-SIAM Symp. on Disc. Algorithms*, 83–92, 1993.

[30] A. Saalfeld. Delaunay edge refinements. *Proc. 3rd Canadian Conf. Comp. Geometry*, 33–36, 1991.

[31] R. Seidel. Backwards analysis of randomized geometric algorithms. *New Trends in Discrete and Computational Geometry*, 37–68, 1993. Springer-Verlag.

[32] M. S. Shephard, J. E. Flaherty, C. L. Bottasso, H. L. de Cougny, C. Özturan, and M. L. Simone. Parallel automatic adaptive analysis. *Parallel Computing* 23(9):1327–1347, 1997.

[33] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. *Proc. First Workshop on Applied Computational Geometry*, 124–133, 1996.

[34] J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. *Proc. 14th ACM Symp. on Computational Geometry*, 86–95, 1998.

[35] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications* 22(1–3):21–74, 2002.

[36] D. A. Spielman and S.-H. Teng. Models and applications of smoothed analysis in geometry, numerical analysis and property testing. *in preparation* , 2002.

[37] B. H. V. Topping and A. I. Khan. *Parallel Finite Element Computations.* Saxe-Coburg Publ., Edinburgh, 1995.