Beating the Spread: Time-Optimal Point Meshing*

[Extended Abstract] †

Gary L. Miller
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15217
glmiller@cs.cmu.edu

Todd Phillips
Google Inc.
Pittsburgh, PA
toddphillips@gmail.com

Donald R. Sheehy
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15217
dsheehy@cs.cmu.edu

ABSTRACT

We present NetMesh, a new algorithm that produces a conforming Delaunay mesh for point sets in any fixed dimension with guaranteed optimal mesh size and quality. Our comparison-based algorithm runs in $O(n \log n + m)$ time, where n is the input size and m is the output size, and with constants depending only on the dimension and the desired element quality. It can terminate early in $O(n \log n)$ time returning a $O(n)$ size Voronoi diagram of a superset of P, which again matches the known lower bounds.

The previous best results in the comparison model depended on the log of the **spread** of the input, the ratio of the largest to smallest pairwise distance. We reduce this dependence to $O(\log n)$ by using a sequence of ϵ-nets to determine input insertion order into a incremental Voronoi diagram. We generate a hierarchy of well-spaced meshes and use these to show that the complexity of the Voronoi diagram stays linear in the number of points throughout the construction.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, Geometrical problems and computations*

General Terms

Algorithms, Theory

Keywords

mesh generation, sparse Voronoi refinement, range spaces, epsilon nets, spread

*Partially supported by the National Science Foundation under grant number CCF-0635257.

†A full version of this paper is available at `www.cs.cmu.edu/~dsheehy`

1. INTRODUCTION

In this paper we present a new algorithm for meshing point sets in fixed dimension. This is the first algorithm we know of that is work-optimal in the comparison-based model in the sense of [18]. Known work-efficient algorithms for meshing are one of two types. The first of these are based on incremental refinement of the Voronoi diagram or Delaunay triangulation. The only work-efficient of these in higher dimension performs a recursive Voronoi refinement where at all times a "quality" Voronoi mesh is maintained. Unfortunately, this leads to work of $O(n \log \Delta + m)$ where n is the input size, m is the output size and Δ is the **spread**, the ratio between the size of the bounding box and the distance between the closest pair of features [10, 12]. The second type uses the quadtree to generate the mesh. Work-efficient versions use bit manipulation of the coordinates of the points to efficiently help with the point location [16, 3, 9]. These algorithms are not optimal in the comparison model and possibly more importantly, it is not known how to efficiently handle higher dimensional features (segments, facets) with these methods.

Our algorithm uses range space ϵ-nets to determine the insertion order of the input points to improve the work bound for point sets with large spread. Clarkson used a similar method for doing point location in a Voronoi diagram [5]. In our approach, since we also add some Steiner points, we can guarantee that the total size of the intermediate Voronoi diagrams are only linear size. This insertion order requires us to maintain a Voronoi diagram that need not have good aspect ratio in the usual sense. Our algorithm will generate a linear-size mesh in fixed constant dimensions. In their 1994 paper Bern, Eppstein, and Gilbert showed how to generate such a linear-size mesh with no large angles [2]. In a later paper we gave a Voronoi refinement algorithm that also generates linear size meshes, [14] but had a running time of only $O(n \log \Delta)$.

Because a standard good aspect ratio mesh is too large, we maintain a weaker but sufficient condition, bounded ply. Throughout the life of the algorithm we maintain a mesh that is of bounded ply which will be used to bound the point location work and the work to determine the insertion order:

Definition 1. *A Voronoi Diagram of a domain Ω is k-ply if for every point $x \in \Omega$ at most k circumballs of the Delaunay simplices contain x in their interior.*

Using the bounded-ply property we can afford to maintain a copy of each uninserted point in each Delaunay ball that

contains it. We pick an insertion ordering so that the number of uninserted points stored in a Delaunay ball decreases geometrically, which we achieve using ϵ-nets.

Let P be the input points and M be a points that have been inserted into the mesh so far including the Steiner points. We say that M is an ϵ-net for P if any ball whose interior is disjoint from M contains at most ϵn points from P. We show, given a mesh M that is an ϵ-net, how to pick at most a constant number of points per Delaunay ball so that after their insertion the new mesh will be a $\epsilon/2$-net. Thus, a **round** consists of adding these new input points plus a constant factor more Steiner points so that we recover a bounded-ply mesh. After $O(\log n)$ rounds the process terminates with a constant ply mesh of size $O(n)$. This output can then be finished to a standard good aspect ratio mesh in output sensitive $O(m)$ time if desired.

In this paper we only consider the case of point set inputs. We feel that the methods proposed should readily be applicable to inputs with higher-dimensional features, such as edges and faces, and with optimal runtime. But we leave this discussion to the full paper.

2. BEATING THE SPREAD

The spread of a point set is the ratio of the largest to smallest interpoint distances, and is denoted as Δ. It is a (geo)metric rather than a combinatorial property; given a set of points P, its cardinality may be n but its spread is not in general bounded by any function of n. It is not uncommon to see a dependence on the spread in the analysis of algorithms in computational geometry and finite metric spaces. Though rarely a problem in practice, it does thwart the most basic principle in the analysis of algorithms, to bound the complexity in terms of the input size.[1]

Consider two classic data structures, the quadtree and the kd-tree. The quadtree partitions space geometrically, breaking squares into 4 equally **sized** pieces. The kd-tree partitions the input points combinatorially into sets of equal **cardinality**. These data structures demonstrate the difference between geometric and combinatorial divide and conquer. The quadtree has depth $\log \Delta$ whereas the kd-tree has depth $\log n$. Unfortunately, many computational problems from nearest neighbor search to network design problems depend on (geo)metric information that is lost when doing a combinatorial divide and conquer. Thus, for many problems, the best known algorithms depend on the spread in either time or space complexity or both.

One approach to dealing with the spread is to restrict the computational model. If coordinates are restricted to be $\log n$-bit integers then the spread is $O(n)$. If we use floating point numbers, the spread is $O(2^n)$. These assumptions about the bit representation of the input also allow for fast computation of logarithms as well as the floor and ceiling functions. These computations are usually omitted from the basic operations of the real RAM model often used in computational geometry to extend the comparison sorting model from the real line to d-dimensional Euclidean space. Har-Peled and Mendel correctly argue that if one can do arithmetic in constant time, it is natural to expect also to perform other operations of size $O(\log \log \Delta)$ in constant time [8]. This is certainly the case for many practical implementations of geometric algorithms. However, it is interesting, both in theory and in practice to explore ways of eliminating the dependence on the spread without resorting to specialized bit operations–in theory because it probes the limits of an important computational model and in practice because it allows one to work with a minimal set of primitives with minimal assumptions about the low-level data representation.[2]

In mesh generation, a dependence on the spread creeps in from two different sources, in the output size and the in the cost of point location. The previous state of the art in comparison based point meshing requires $O(n \log \Delta + m)$ work, where the first term is the cost of point location and the second is the output sensitive term. Even for point set inputs, the lower bounds on quality meshes imply that m may also depend on the spread. Thus, to contest with the spread, we must both optimize point location and also relax the quality condition.

3. VORONOI REFINEMENT BASICS

Voronoi Diagrams.

The Voronoi diagram of a finite point set P in $\mathbb{R}^d$, denoted Vor_P, is the polyhedral complex decomposing $\mathbb{R}^d$ into regions based on the nearest neighbor among the points of P. The regions are called Voronoi cells. Since some cells are unbounded, we assume there is a suitably sized bounding ball around the points of P and for a point $p \in P$ we write $\mathrm{Vor}_P(p)$ to denote the intersection of the Voronoi cell with this ball.

The **in-radius** of $\mathrm{Vor}_P(p)$, denoted r_p, is the radius of the largest ball centered at p that is contained in $\mathrm{Vor}_P(p)$. The **out-radius**, R_p is the radius of the smallest ball centered at p that contains $\mathrm{Vor}_P(p)$. The **aspect ratio** of $\mathrm{Vor}_P(p)$ is $\frac{R_p}{r_p}$. When viewed as cell complex, the vertices or 0-faces of $\mathrm{Vor}_P(p)$ are called the **corners**, to avoid confusion with the notion of a mesh vertex introduced later. The out-radius is therefore, the distance from p to its farthest corner, while the in-radius is the distance from p to its nearest facet. A set of points is **τ-well-spaced** if every Voronoi cell has aspect ratio at most τ.

The Voronoi diagram is the dual of the Delaunay triangulation, which has a simplex for every subset of points on the boundary of a ball that contains no other points of P. For full-dimensional Delaunay simplices, these circumscribing balls are called **D-balls**. The corners of the Voronoi cells correspond to D-balls[3].

Voronoi Refinement.

The goal of Voronoi refinement is to produce a τ-well-spaced set M by adding new vertices called Steiner points to an input set P that is not well-spaced. We want to add as few vertices as possible. The algorithm is simple: starting with Vor_P, iteratively add the farthest corner of any cell with aspect ratio greater than τ. It is perhaps more com-

[1] One can get around this by making assumptions about the bit representations of the inputs. We will address this as well.

[2] Recall that in the popular CGAL library, all primitives are implemented for several different kernels, all use a small, unified interface.

[3] We permit corners on the boundary to represent "degenerate" D-balls, ones corresponding to simplices that are not full-dimensional.

monly known in its dual formulation, as Delaunay refinement, where the goal is to improve the Delaunay simplices rather than the Voronoi cells. But the resulting algorithms and their analysis are nearly identical for both the primal and the dual formulation.

It is not immediately obvious that Voronoi refinement should ever terminate. Indeed, for some τ, it will run forever. For a reasonable choice of τ, say $\tau = 3$ for example, not only will the algorithm terminate, it will do so with asymptotically optimal size, both in the number of points added and the total number of faces in the diagram. This latter property results from the aspect ratio condition, and is a major motivation for doing the refinement in the first place.

Sparse Voronoi Refinement.

The first obstacle to producing a refined Voronoi diagram in optimal time and space is that the input may have a large Voronoi diagram, $\Omega(n^{\lceil d/2 \rceil})$ faces in the worst case. To overcome this obstacle, the Sparse Voronoi Refinement (SVR) algorithm of Hudson et al. [10] interleaves the addition of input points to the diagram with the addition of Steiner points. In doing so, the algorithm requires two extra pieces. First, input points are only added if they are "close" to the current Voronoi diagram. Second, the Steiner points may not be added "too close" to uninserted input points. The former notion of closeness is what we call **ε-medial**, the ratio of the distances to the nearest and second nearest points must be at most ε. The latter notion of closeness causes the algorithm to **yield** by adding an input point p rather than a Steiner point v if the distance from p to v is less than γ times the radius of the empty ball around v.

By only inserting ε-medial points and yielding when appropriate, Sparse Voronoi refinement maintains a good aspect ratio Voronoi diagram at every stage of the algorithm. Consequently, the total work is output sensitive. This approach has also been generalized to more complex inputs than just point sets, considering also piecewise linear complexes [10].

Point Location, Point Location, Point Location.

The bottleneck for the running time of Voronoi refinement is point location. Recall, that in the standard incremental Voronoi (or Delaunay) algorithm, the first step to inserting a new point is to find that point in the current diagram. A natural and highly effective technique for doing this point location is to eagerly store the uninserted points in the D-balls of each Voronoi diagram as the algorithm progresses. Points are moved whenever an insertion changes a D-ball locally.

In SVR, this approach corresponds to a geometric divide and conquer, similar to quadtrees, because after a constant number of moves, the size of the balls containing any point goes down by a constant factor. Thus, in SVR a single input point may be moved $\Theta(\log \Delta)$ times. In this work, we show how to modify the algorithm so that only $O(\log n)$ moves are necessary. One way to view these results is as a way to achieve similar properties to compressed quadtrees without leaving the comparison model or privileging any fixed set of coordinate axes.

4. DEFINITIONS AND NOTATIONS

Points, Vectors, and Distances.

We will treat points of d-dimensional Euclidean space as vectors in $\mathbb{R}^d$. As such, we denote the euclidean distance between two points $x, y \in \mathbb{R}^d$ as $|x - y|$. Moreover, we allow the usual operations of scalar multiplication and addition on points and also on sets of points. So, for example, if S is the unit sphere centered at the origin, c is any point, and r is a non-negative real number, then $rS + c$ is the sphere of radius r centered at c. We will also define the distance from a point x to a set S as $\mathbf{d}(x, S) = \inf_{y \in S} |x - y|$. We write $\text{ball}(c, r)$ to denote the open ball of radius r centered at c and $\text{conv}(X)$ to denote the convex closure of $X \subset \mathbb{R}^d$.

Domains.

A **domain** $\Omega \subset \mathbb{R}^d$ is defined by a center c_Ω, a radius r_Ω, and a collection of disjoint open balls $B_1, \ldots, B_k \subset B_\Omega = \text{ball}(c_\Omega, r_\Omega)$ such that

$$\Omega = B_\Omega \setminus \left(\bigcup_{i=1}^{k} B_i \right).$$

The ball B_Ω is called the **bounding ball** of Ω and $S_\Omega = \{x \in \mathbb{R}^d : |x - c_\Omega| = r_\Omega\}$ is the **bounding sphere** of Ω.

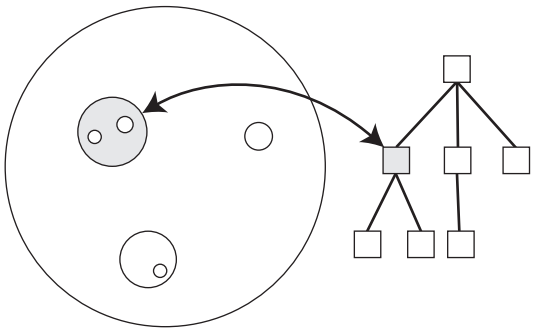

Figure 1: A domain hierarchy as a collection of sets (left) and its tree structure (right).

We get a hierarchy of domains if the balls removed from B_Ω are the bounding balls of other domains. Formally, a **domain hierarchy** is a tree H with disjoint domains as nodes rooted at Ω_{root} such that

1. for any pair $\Omega, \Omega' \in H$, $\text{p}(\Omega') = \Omega$ if and only if $S_{\Omega'} \subset \Omega$, and
2. $\bigcup_{\Omega \in H} \Omega = B_{\Omega_{\text{root}}}$.

Here, $\text{p}(\Omega)$ denotes the parent of Ω in H.

Cages.

Given a domain Ω, we want to add vertices near S_Ω to limit the interaction between the inside and the outside of Ω. We will have two parameters, δ determining the density of these points, and γ determining how nearly cospherical they are. We call such a set C_Ω of vertices a **cage** and we require the following three properties, where $\overline{r} = (1-\delta-\gamma)r_\Omega$ and $\overline{S} = (1 - \delta - \gamma)S_\Omega$.

1. **[Nearness Property]** For all $v \in C_\Omega$, $\mathbf{d}(v, \overline{S}) \leq \gamma \overline{r}$.

2. **[Covering Property]** For all $x \in \overline{S}$, $\mathbf{d}(x, C_\Omega) \leq (\delta + \gamma)\overline{r}$.
3. **[Packing Property]** For all distinct $u, v \in C_\Omega$, $|u - v| \geq (\delta - 2\gamma)\overline{r}$.

To construct such a set of points, we start with a **cage template** T of points on the unit sphere S. The points of T are a metric space δ-net on S (not to be confused with the range space nets used elsewhere in this paper). That is, for all $x \in S$, $\mathbf{d}(x, T) \leq \delta$ and for each distinct pair $u, v \in T$, $|u - v| \geq \delta$. Such sets are known to exist and can be constructed using a simple greedy algorithm [7, 13].

For a domain Ω we construct its cage by adding for each $x \in c_\Omega + \overline{r}T$, a new point x' such that $|x - x'| \leq \gamma\overline{r}$. It is easy to check that this set of points will satisfy the three properties of a cage.

DEFINITION 2. *A cage C_Ω centered at c with radius r is **ε-encroached** or simply **encroached** by a point $p \notin C_\Omega$ if either*

1. *p is an input point in* $\text{annulus}(c, \varepsilon r, r)$*, or*
2. *p is an input or Steiner point in* $\text{annulus}(c, r, \frac{2r}{\varepsilon})$.

Roughly speaking, non-encroached cages have room on the inside (w.r.t. input points) and room on the outside (w.r.t. all mesh vertices).

Hierarchical Meshes.

A **mesh** is a set of points M and its Voronoi diagram. The points of M are called the **vertices** of the mesh.

DEFINITION 3. *A **hierarchical mesh** is a mesh M along with a domain hierarchy H_M such that:*

1. *M has a vertex at the center of every domain, i.e. $c_\Omega \in M$ for all $\Omega \in H_M$*
2. *No domain is ε-encroached.*

Given a hierarchical mesh M and $\Omega \in H_M$, we define M_Ω to be the points of M contained in Ω plus the centers of the children of Ω in H_M. Formally,

$$M_\Omega = (M \cap \Omega) \cup \bigcup_{\Omega' \in \text{children}(\Omega)} \{c_{\Omega'}\}.$$

We call this the set M restricted to the domain Ω, and it is well defined for any domain Ω and any set M that contains the centers of the children of Ω. The set $P_\Omega \subset M_\Omega$ of input points added to a domain is the subset of $P \cap M_\Omega$ that were *inserted as inputs*, as opposed to those added by yielding. The one exception is that the center of Ω is always contained in P_Ω, even if it was inserted because of a yield.

In a hierarchical mesh, we can also define the Voronoi cell of a cage C_Ω as

$$\text{Vor}_M(C_\Omega) = \bigcup_{u \in M \cap B_\Omega} \text{Vor}_M(u).$$

DEFINITION 4. *We say that a hierarchical mesh M is **τ-quality** if the following conditions are met:*

1. *For every non-cage vertex $v \in M$, $\text{Vor}_M(v)$ has aspect ratio at most τ.*
2. *For all $\Omega \in H_M$, $\text{Vor}_M(C_\Omega)$ has aspect ratio at most τ.*

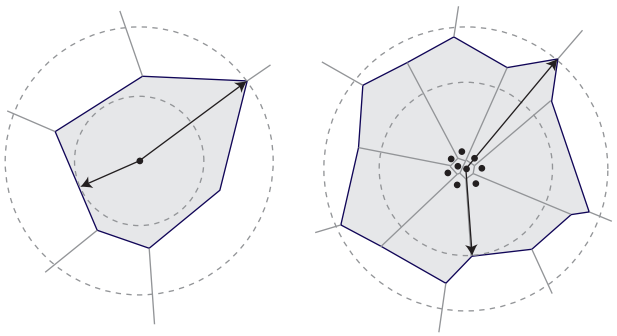

Figure 2: Quality cells of a vertex (left) and a cage (right). The dotted circles indicate the in-radii and out-radii

3. *No domain in H_M is ε-encroached.*

The four constants γ, δ, ε, and τ are called the **meshing parameters**. Throughout, they are assumed to be fixed constants independent of the dimension.

DEFINITION 5. *For a set X and a domain Ω, the **feature size** is a function $\mathbf{f}_X^\Omega : \mathbb{R}^d \to \mathbb{R}$ that maps a point x to the distance to its second nearest neighbor among the points of X_Ω and the bounding sphere of Ω.*

We are mainly interested in the feature size of the input and of the mesh, $\mathbf{f}_P^\Omega$ and $\mathbf{f}_M^\Omega$ respectively, over the domains of M_H. There is a natural relationship between the feature size and the in-radius of a Voronoi cell. For reasonable choices of ε, a non-encroached cages $C_{\Omega'}$ centered at c has an in-radius r such that $r \leq \mathbf{f}_M^\Omega(c) \leq 3r$, $\Omega = \text{p}(\Omega')$.

5. THE ALGORITHM

5.1 Overview of the Algorithm

Like Sparse Voronoi Refinement, the core of the NetMesh algorithm is an incremental construction of a Voronoi diagram with the refinement steps to maintain mesh quality. There are five main concerns. The algorithm must **(1) order the input points**. These points are added one at a time in an **(2) incremental construction**. After each insertion, Steiner points are added in a **(3) refinement** phase that recovers the quality invariant. All the while, uninserted points are organized in a **(4) point location** data structure. Once all of the inputs have been added, an optional **(5) finishing** procedure turns the linear-size hierarchical mesh into a standard well-spaced mesh. Each of these concerns will be addressed in more detail below, but first we will describe the main ideas used and how they fit together.

Point Location. The point location data structure associates each point with each D-ball that contains it. So, it is easy to report the set of D-balls containing an input point and similarly, to report the set of points in a D-ball. These associations are updated locally every time a new point changes the underlying Delaunay triangulation. We will prove that no point is ever in more than a constant number of D-balls and thus the size of this structure will not exceed $O(n)$.

Incremental updates. In Sparse Voronoi Refinement, every insertion is medial. This is critical to maintain quality in the mesh throughout the algorithm. In the NetMesh

algorithm, we change the domain hierarchy before inserting each point to guarantee that it is medial in whatever domain contains it. We show that this is sufficient to get the same guarantees as in SVR. Thus, we can insert the points in any order.

Ordering the input with ϵ-nets. The theory of range space ϵ-nets is used to choose the input insertion order. One round of the algorithm consists of generating an ϵ-net for the current mesh by generating a union of ϵ-nets for the input points of each D-ball, where the ranges are open balls. It is known that each such net per D-ball has constant size, and can be found in deterministic linear time [4]. The points in any round may be inserted in any order, after which, the next round is computed. In each round, the maximum number of points stored in any D-ball goes down by a constant factor, so the total number of rounds is $O(\log n)$.

Refinement. The refinement, or cleaning phase of the algorithm is a standard Voronoi refinement in that it adds Steiner points at the farthest corner of any cell with bad aspect ratio. As in SVR, if the Steiner point is sufficiently close to an uninserted input point p, then p is added instead. One slight change is that we maintain the aspect ratio of the Voronoi cells of cages, but do not require the cage vertices themselves to have good aspect ratio Voronoi cells.

Finishing the mesh. The algorithm produces a quality hierarchical mesh of linear size. If one wants to extend this mesh to a standard well-spaced mesh, it is a straightforward procedure to do this in $O(m)$ time, where m is the number of vertices in an optimal-size, well-spaced superset of P. This finishing process can run quickly because it need not do any point location (all of the input points have already been inserted).

5.2 Point Location Operations

Each uninserted input point stores a list of D-balls that contain it as well as a list of cages that it encroaches. Similarly, the D-balls have lists of uninserted vertices that they contain. With each change in the Voronoi diagram, these lists are updated. We say that the points are stored *in* the balls to simplify the description of this list upkeep. A point will generally be contained in several D-balls. The uninserted points are moved out of D-balls that have been destroyed and into newly created D-balls. This shuffling of points between D-balls is the work of point location. A point is touched in this process if it is moved into a new ball or even if it is considered for moving into a new ball. We count the point location work from the perspective of the uninserted input points.

There are four main point location operations needed.

1. Find the D-balls containing a point to insert it into the Voronoi diagram.
2. Find the nearest and second nearest neighbor of a point in its domain in order to compute its mediality.
3. Find any cages encroached by a given point.
4. Find a nearby input point to yield to, when inserting a Steiner point.

The first operation is trivial.

For cage vertices v in a domain Ω, let $center(v)$ be the vertex at the center of Ω. Let $B(x)$ be the set of D-balls containing x. Let $V(B)$ be the $d+1$ vertices of the Delaunay simplex corresponding to the D-ball B. Let $U(x) = \{V(B) : B \in B(x)\}$. If Ω is the domain containing x, then the nearest and second nearest neighbors of x in M_Ω are in $U(x)$ or $\{center(v) : v \in U(x)\}$, so it is easy to identify them. Call these vertices n_x and s_x respectively. Thus, $\textsc{Mediality}(x) = \frac{|x-n_x|}{|x-s_x|}$ can be computed in time $O(|B(x)|)$.

To check encroachment of input points is easy because this information is stored with the points. At the time a cage is created, any encroaching input points must be relocated, so the encroachment is discovered at that time. To check encroachment of Steiner points, it suffices to observe that if a Steiner point x encroaches a cage C, then some vertex of c must appear in $U(x)$. So, there are only $O(|U(x)|)$ cages to check and each check takes constant time.

To find a point to yield to, we simply need to check for input points in a small empty ball around the proposed input point. This is trivial for Steiner points added during refinement because the Steiner point is the center of a D-ball B and thus we only need to check $\textsc{Uninserted}(B)$. For cage vertices v, the search requires us also to check the points in $\{\textsc{Uninserted}(B) : B \in B(v)\}$. In both cases, the points checked in this process also need to be checked for relocation when the new vertex is inserted. Thus, the cost for this search is dominated by the cost of relocating points, which we analyze in detail later.

5.3 Incremental Updates to Hierarchical Meshes

The basic operation in incremental Voronoi diagrams is $\textsc{Insert}(v)$, which adds the vertex v to the Voronoi diagram and updates the point location data structures. To keep this operation constant time (not counting the cost of point location), we must guarantee that $|B(v)|$ is a constant because every D-ball in $B(v)$ is destroyed by the insertion. This is done by making sure that every new insertion is medial. Before the new point is inserted, we update the domain hierarchy. If the point was not medial, then it must be significantly closer to its nearest neighbor than it second nearest neighbor, and thus we add or expand cage around the nearest neighbor. We must also update the domain hierarchy of the new point encroaches on an existing cage.

$\textsc{Insert}(x)$
for each C in $\textsc{OutEncroach}(x)$: $\textsc{ReleaseCage}(C)$
Add x to Vor_M and update the point location structure.

$\textsc{YieldingInsert}(x)$
let v be the nearest neighbor of x in the current mesh.
if there is an input point p in ball$(x, \gamma|x-v|)$
then $\textsc{Insert}$(p) **else** $\textsc{Insert}$(x)

There are three basic cage operations:

$\textsc{NewCage}(p, r)$
Initialize a new cage.
for each $x \in T$, $\textsc{YieldingInsert}(rx + p)$.

$\textsc{ReleaseCage}(C)$
for each cage vertex v in C push v to the $\textsc{RefineList}$.
Delete the cage C

$\textsc{GrowCage}(C)$
Let x be the center of C and let r be its radius.
if in-radius$(\text{Vor}(C)) \geq \frac{r}{\varepsilon^2}$ **then** $\textsc{NewCage}(x, \frac{r}{\varepsilon})$.

ReleaseCage(C).

Equipped with the cage operations, we define the following routine. Its purpose is to rearrange the domain hierarchy by creating or growing new cages so that a new vertex v can be added to a domain in which it is medial.

InsertInput(v)
  let u be the nearest neighbor of v in M_Ω
  if Mediality(v) $\leq \varepsilon$ **then** NewCage($u, |u-p|/\varepsilon$)
  for each C in InEncroach(v): GrowCage(C)
  Insert(v)

5.4 Refinement

The algorithm maintains a list of cells with bad aspect ratio called RefineList. The cleaning procedure goes through this list and refines these cells until none are left. The RefineList is updated every time a Voronoi cell changes. The structure of the Voronoi diagram makes it easy to check the aspect ratio of a cell and Theorem 2 implies that this can be done in constant time. If a cell's aspect ratio was good but goes bad, it is added to the list. If its aspect ratio was bad but becomes good, it is removed from the list.

Clean(M: Mesh)
  while RefineList is not empty
    let $v \in$ RefineList
    let x be the far corner of Vor(v)
    YieldingInsert(x)

5.5 Input Ordering with ϵ-Nets

We employ the theory of range space ϵ-nets to order the inputs for insertion. The following is a special case of Theorem 4.6 from [4] when the range space is defined by open balls.

Theorem 1. *Let $P \subset \mathbb{R}^d$ be a set of n points and let $\epsilon \in (0,1)$. There exists an algorithm* Net(ϵ, P) *that runs in $O(n)$ time and returns a subset $N \subseteq P$ such that $|N| = O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ and any open ball that contains ϵn points of P also contains a point of N.*

Using the Net algorithm as a black box, we select the next round of points to insert as follows.

SelectRound(M: mesh)
  $N \leftarrow \emptyset$
  for each $B \in$ DBalls(M)
    $N \leftarrow N \cup$ Net($\frac{1}{2d}$, Uninserted(B))
  return N

If the maximum number of uninserted points in a D-ball of some mesh is k, then after adding the points chosen by SelectRound, this maximum is at most $\frac{k}{2}$. This follows from the fact that every new D-ball is covered by at most d of the old D-balls (see Theorem 3). So, the total number of rounds is at most $\lceil \log n \rceil$. We can now give the main loop of the algorithm.

NetMesh(P: points)
  Initialize an empty mesh M
  Uninserted $\leftarrow P$
  let c, r be such that $P \in$ ball(c, r)
  OuterCage = NewCage(c, $\frac{r}{\varepsilon}$)
  while Uninserted is not empty
    V = SelectRound(M)
    for each $v \in V$
      InsertInput(v)
      Clean(M)
  return M

5.6 Finishing the Mesh

The output of NetMesh is a quality hierarchical mesh. If the desired output is a well-spaced mesh according to the traditional definition, i.e. quality with a single domain, then some finishing procedure is required. Fortunately, it is trivial given the cage operations defined above:

FinishMesh(M: Mesh)
  while there exists a cage C other than OuterCage
    GrowCage(C)
    Clean(M)

Since the cages are not encroached, they have some space around them. The FinishMesh procedure simply grows the cages until this space is filled. No new cages are formed and no point location work on input points is required.

Note that finishing the hierarchical mesh in this way may result in a mesh with more than a linear number of points because well-spaced meshes are subject to potentially superlinear (or even superpolynomial!) lowerbounds. This is why we consider the finishing operation to be optional.

6. OVERVIEW OF THE ANALYSIS

An intermediate mesh, M_i, is the mesh after i vertices or cages have been inserted in during the incremental construction. To analyze the NetMesh algorithm, we will prove that two invariants are maintained for each intermediate mesh: **the feature size invariant** and **the quality invariant**.

Definition 6. *A hierarchical mesh M satisfies the **quality invariant** if each intermediate mesh M_i is τ'-quality for some constant τ' depending only on the meshing parameters.*

Definition 7. *A hierarchical mesh M of an input set P satisfies the **feature size invariant** if for all domains $\Omega \in H_M$ and all vertices $v \in M_\Omega$*

$$\mathbf{f}^\Omega_P(v) \leq K f^\Omega_M(v),$$

where K is a constant that depend only on the mesh parameters.

The quality invariant is useful because of several properties of quality meshes.

Theorem 2. *If M is a τ-quality mesh, then*

1. *no point of $\mathbb{R}^d$ is contained in more than $O(1)$ D-balls,*
2. *no D-ball intersects more than $O(1)$ other D-balls, and*
3. *no vertex of M has more than $O(1)$ Delaunay neighbors.*

These structural results about quality meshes are known for the case of a single domain [15, 10]. To extend them to the case of a quality hierarchical meshes follows the same

methods as in previous work. Detailed proofs may be found in the full paper.

Over a single domain, standard results in mesh size analysis imply that the feature size invariant suffices to prove that the number of vertices is bounded (up to constants) by the feature size integral [17]:

$$\int_\Omega \frac{dx}{\mathbf{f}^\Omega_P(x)^d}.$$

In previous work [14], we showed that the feature size integral is $O(n)$ when the input points satisfy a certain spacing condition. We prove that in each domain Ω of the hierarchy, the points of P_Ω satisfy this spacing condition (Lemma 7), allowing us to prove that the total output size is $O(n)$ (Theorem 8).

Theorem 2 and the quality invariant imply that the cost to update the Voronoi diagram for a single insertion is constant. That is, the number of combinatorial changes to the Voronoi diagram is constant for each each insertion. Thus, since the total number of points added is $O(n)$, the total work is $O(n)$, not counting the cost of point location.

To bound the cost of point location, we first show that at most a constant number of vertices are added to any D-ball in the course of a round (Lemma 12). This is then used to show that the total amount of point location work is $O(n)$ per round. Since there are only $O(\log n)$ rounds, the total work is $O(n \log n)$ as desired.

7. RANGE SPACES AND ϵ-NETS

In this section we discuss ideas and definitions from hypergraph and range space theory that we will need in our meshing algorithm. We will also give a distance measure derived from a range space that is useful for our analysis. A **range space** or **hypergraph** is a pair $(X, \mathcal{R})$ where X is a set and $\mathcal{R}$ is a collection of sets called **ranges**. A **range space ϵ-net** for $(X, \mathcal{R})$ is a subset N of X such that $N \cap R \neq \emptyset$ for all $R \in \mathcal{R}$ such that $|R \cap X| \geq \epsilon |X|$.

Throughout this discussion the ranges will be open balls in $\mathbb{R}^d$ including those with infinite radius, i.e. halfspaces. For a subset $M \subset \mathbb{R}^d$, define:

$$\mathcal{B}_M = \{B \ : \ B \text{ is a ball and } B \cap M = \emptyset\}.$$

A useful subset of $\mathcal{B}_M$ is the set of D-balls of M:

$$\mathcal{D}_M = \{B \in \mathcal{B}_M \ : \ B \text{ is a D-ball of } M\}.$$

The following geometric lemma is useful for translating between statements about D-balls and statements about arbitrary empty balls in the space.

THEOREM 3. *If $M \subset \mathbb{R}^d$ and $B \in \mathcal{B}_M$ then $B \cap \text{conv}(M)$ is covered by at most d D-balls of $\mathcal{D}_M$ and these d balls all share a common point.*

The proof appears in the full version.

Let $\mathcal{G}_{\mathcal{B}_M}$ be the graph with vertex set $\mathcal{B}_M$ and edges for each pair of balls that intersect. For any $x, y \in R_d \setminus M$, let $\mathbf{d}_{\mathcal{B}_M}(x, y)$ be the length of the shortest path in $\mathcal{G}_{\mathcal{B}_M}$ between a ball containing x to a ball containing y. Define $\mathcal{G}_{\mathcal{D}_M}$ and $\mathbf{d}_{\mathcal{D}_M}$ similarly. These distances are related by the following lemma.

LEMMA 4. *If $M \subset \mathbb{R}^d$ is finite then $\mathbf{d}_{\mathcal{D}_M}(x, y) \leq 2\mathbf{d}_{\mathcal{B}_M}(x, y)$ for all $x, y \in \text{conv}(M)$.*

PROOF. Fix a pair of points $x, y \in \text{conv}(M)$ and let $s = \mathbf{d}_{\mathcal{B}_M}(x, y)$. It will suffice to find D-balls $E_1, \ldots, E_{2s} \in \mathcal{D}_M$ such that $x \in E_1$, $y \in E_{2s}$, and each $E_i \cap E_{i+1}$ is nonempty. By the definition of $\mathbf{d}_{\mathcal{B}_M}$, there exists balls $B_1, \ldots, B_s \in \mathcal{B}_M$ such that $x \in B_1$, $y \in B_s$, and each $B_i \cap B_{i+1}$ is nonempty. Let z_i be a point in $B_i \cap B_{i+1}$ for $i = 1 \ldots s-1$ and define $z_0 := x$ and $z_s := y$. Now, by Theorem 3, there are d D-balls covering each B_i and they all have a common intersection. So, letting E_{2i-1} and E_{2i} be the D-balls among these that contain z_{i-1} and z_i gives the desired path of length at most $2s$ in $\mathcal{G}_{\mathcal{D}_M}$. □

8. SIZE BOUNDS

In this section we will show that the output of NETMESH has linear size. The analysis will follow a straightforward strategy. We will argue that the algorithm never inserts a vertex too close to an existing vertex. This is known as the **insertion radius invariant**, and it allows us to prove that the **feature size invariant** holds for all intermediate meshes. We use this to prove that for all domains Ω, M_Ω has size linear in $|P_\Omega|$ from which the overall bound follows. This strategy is not new; it parallels closely the approach of Ruppert [17] for Delaunay refinement and its sparse version introduced by Hudson, Miller, and Phillips [10]. We have adapted it to the case of hierarchical meshes.

We say that a hierarchical mesh M is constructed **incrementally** if the vertices are added one at a time and the domains are adjusted before every insertion so that no domain is encroached. In particular, the algorithm given is such an incremental construction. The intermediate mesh after i points and cages have been added is denoted by M_i, its domain hierarchy H_{M_i} is denoted H_i, and $P_i = P \cap M_i$ is the set of inputs inserted thus far. Define the **insertion radius** of the ith vertex added v as $\lambda_v = \mathbf{f}^\Omega_{M_i}(v)$, where $\Omega \in H_i$ is the domain into which v was inserted.

DEFINITION 8. *A hierarchical mesh M of an input set P constructed incrementally satisfies the **insertion radius invariant** if for all domains $\Omega \in H_i$ for all i and all vertices $v \in M_{i\Omega}$*

$$\mathbf{f}^\Omega_{P_i}(v) \leq \begin{cases} K'_C \lambda_v & \text{if } v \text{ is inserted as a cage vertex,} \\ K'_S \lambda_v & \text{if } v \text{ is inserted as a circumcenter, and} \\ K'_I \lambda_v & \text{if } v \text{ is inserted as an input vertex} \end{cases}$$

where K'_C,K'_S, and K'_I are constants that depend only on the mesh parameters.

The following lemma states that as long as the insertion radius of every vertex is not too small then the distance to its nearest neighbor is also not too small. Its proof is straightforward and reserved for the full paper.

LEMMA 5. *If M is a hierarchical mesh constructed incrementally that satisfies the insertion radius invariant, then M also satisfies the feature size invariant.*

Lemma 5 implies that in order to prove that the spacing of the points in the final mesh is good, it will suffice to show that the algorithm maintains the insertion radius invariant throughout. This is proven in the following lemma.

LEMMA 6. *The hierarchical mesh M constructed by the* NETMESH *algorithm satisfies the insertion radius invariant.*

PROOF. We proceed by induction on the total number of vertices added. Let v be the ith vertex added and let Ω be the domain it is inserted into.

Case 1: v is a cage vertex. Since P_Ω contains at least the center of Ω, the feature size is bounded as $\mathbf{f}^\Omega_P(v) \le r_\Omega$. By construction, adjacent cage vertices are at least αr_Ω apart, where $\alpha = (\delta - 2\gamma)(1 - \delta - \gamma)$. So, $\lambda_v \ge \alpha r_\Omega$. Combining these two facts and choosing $K'_C \ge \frac{1+\varepsilon}{\alpha}$ yields $\mathbf{f}^\Omega_{P_i}(v) \le K'_C\lambda_v$ as desired.

Case 2: v is a clean move. Steiner points are added when some vertex (or cage) $u \in M_{i-1\Omega}$ has aspect ratio greater than τ. Let V_u denote this poor aspect ratio cell. Let w be the nearest neighbor of u in M_Ω, so $\mathbf{f}^\Omega_M(u) = |u - w|$. In case we yielded in order to insert v, let v' be the true circumcenter that we tried to insert. The yielding condition guarantees that

$$|v - v'| \le \gamma|u - v'|. \tag{1}$$

Since u or w or both can be the center of a child domain of Ω, we need to also consider vertices u', w' of M that define the insertion radius of v and the in-radius of V_u respectively. Since w does not encroach a domain at u and $|u - w| \le |u - v'|$, it follows that

$$|u - u'| \le \varepsilon|u - v'|. \tag{2}$$

The D-ball centered at v' has radius $|u' - v'|$ and is empty of vertices, so $\lambda_v \ge |u' - v'| - |v - v'|$. Using the triangle inequality, (1), and (2), we can bound the insertion radius as follows.

$$|u - v'| \le \beta\lambda_v. \tag{3}$$

where, $\beta = \frac{1}{1-\varepsilon-\gamma}$. Since w is closer to u than v, $\mathbf{f}^\Omega_{M_{i-1}}(u) = \mathbf{f}^\Omega_{M_i}(u)$ and $\mathbf{f}^\Omega_{P_{i-1}}(u) = \mathbf{f}^\Omega_{P_i}(u)$. So, we can use induction and Lemma 5 to get that

$$\mathbf{f}^\Omega_{P_i}(u) \le K'_I|u - w|. \tag{4}$$

We use K'_I because it is the largest of the K' constants.

We may now derive a bound on $\mathbf{f}^\Omega_{P_i}(v)$ as follows.

$$\begin{aligned}
\mathbf{f}^\Omega_{P_i}(v) &\le \mathbf{f}^\Omega_{P_i}(u) + |u - v| && \left[\mathbf{f}^\Omega_{P_i} \text{ is 1-Lipschitz}\right] \\
&\le \mathbf{f}^\Omega_{P_i}(u) + |u - v'| + |v' - v| && [\text{triangle inequality}] \\
&\le \mathbf{f}^\Omega_{P_i}(u) + (1+\gamma)|u - v'| && [\text{by (1)}] \\
&\le K'_I|u - w| + (1+\gamma)|u - v'| && [\text{by (4)}] \\
&\le \left(\frac{3K'_I}{\tau} + 1 + \gamma\right)|u - v'| && [V_u \text{ aspect ratio} > \tau] \\
&\le \left(\frac{3K'_I}{\tau} + 1 + \gamma\right)\beta\lambda_v && [\text{by (3)}]
\end{aligned}$$

So, setting $K'_S \ge \left(\frac{3K'_I}{\tau} + 1 + \gamma\right)\beta$ yields the desired bound.

Case 3: v is an input. Choose u such that $\lambda_v = |u - v|$ and let j and Ω_j be the time that u was inserted and the domain it was inserted into respectively. If $u \in C_\Omega$, then v encroaches on Ω, which is impossible. If u is an input vertex then $\lambda_v = f^\Omega_P(v)$ so we are done. So, we may assume that u is a Steiner point, inserted either as either a circumcenter or as a cage vertex that was later released.

The feature size of a point in the domain that contains it cannot go down in the course of the algorithm, because the mesh in each domain only gets larger. Thus, for example

$$\mathbf{f}^\Omega_{P_i}(u) \le \mathbf{f}^{\Omega_j}_{P_j}(u). \tag{5}$$

We define $K'_u = K'_S$ in the former case and $K'_u = K'_C$ in the latter. By choosing $K'_I \ge \frac{K'_u}{\gamma} + 1$, we can now derive the following bound.

$$\begin{aligned}
\mathbf{f}^\Omega_{P_i}(v) &\le \mathbf{f}^\Omega_{P_i}(u) + |u - v| && \left[\mathbf{f}^\Omega_{P_i} \text{ is 1-Lipschitz}\right] \\
&\le \mathbf{f}^{\Omega_j}_{P_j}(u) + |u - v| && [\text{by (5)}] \\
&\le K'_u\lambda_u + |u - v| && [\text{by induction}] \\
&\le \left(\frac{K'_u}{\gamma} + 1\right)|u - v| && [\text{because } u \text{ did not yield to } v] \\
&\le K'_I|u - v| && \left[\frac{K'_u}{\gamma} + 1 \le K\right] \\
&= K'_I\lambda_v. && [\lambda_v = |u - v|]
\end{aligned}$$

□

LEMMA 7. *Let q and q' be any two input points and let r be the distance between them. If $A = \text{annulus}(q, 2r, \frac{6r}{\varepsilon^3})$ contains no input points, then q and q' are inside some cage contained in A for all intermediate meshes after each has been inserted.*

PROOF SKETCH. Let $p_1, \ldots, p_k$ be all input points in ball$(p, 2r)$ ordered by the order in which they were inserted. Clearly q and q' are among the p_i's. The proof is a straightforward induction on k, requiring us only to show that each insertion leaves the desired cage around the previous set. The constant $\frac{6}{\varepsilon^3}$ was carefully chosen to make this work. The details may be found in the full paper. □

We can now prove that the output mesh has size linear in the input size.

THEOREM 8. *If M is the output of the* NETMESH *algorithm for an input set P, then $|M| = O(|P|)$.*

PROOF SKETCH. Let Ω be any domain in the output. Let $p_1 \ldots, p_j$ be the vertices of P_Ω ordered such that for each $i = 3 \ldots j$, $\mathbf{f}_{P^\Omega_i}(p_i)/\mathbf{f}_{P^\Omega_{i-1}}(p_i) \ge \frac{12}{\varepsilon^3} + 1$, where $P_i = \{p_1, \ldots, p_i\}$. Lemma 7 guarantees that such an ordering can be found by a trivial greedy algorithm (see the full paper for details of the construction).

In previous work [14], we showed if P_Ω can be ordered this way then any well-spaced superset satisfying the bound in Lemma 6 has size $O(|P_\Omega|)$. So, in particular $|M_\Omega| = O(|P_\Omega|)$. Now, we observe that because every domain contains at least 2 input points, $\sum_\Omega |P_\Omega| < 2|P|$. So, the total mesh size can be bounded as $|M| \le \sum_\Omega |M_\Omega| = O(\sum_\Omega |P_\Omega|) = O(|P|)$. □

9. QUALITY AND PLY

THEOREM 9. *For any input, the intermediate meshes of the* NETMESH *algorithm are τ''-quality, where τ'' depends only on the mesh parameters.*

We give only a sketch of the main ideas of the proof here and refer the reader to the full paper for details. The proof is based directly on the proof from [10] that SVR maintains a quality mesh throughout. We have only a few extra cases because of the extra work involved in creating and releasing cages. The key lemma is as follows.

LEMMA 10. *Let M be a τ-quality hierarchical mesh and let Ω be any domain in H_M. If B is a ball of radius r centered in B_Ω empty of points in M and $x \in B$, then*

$$\mathbf{f}^{\Omega}_{M}(x) \geq c_{10} r,$$

where $c_{10} = \frac{1}{24\tau^2}$

It guarantees that as long as there is a large empty ball nearby, the feature size function cannot be too small. The corresponding lemma from [10] is Lemma 6.1 and again, the same proof carries over easily to hierarchical meshes. Lemma 10 is also critical in the proof of Theorem 2, which guarantees that quality meshes have constant ply (the details of which can also be found in the full version).

10. POINT LOCATION ANALYSIS

DEFINITION 9. *A vertex $v \in M$ **touches** an uninserted point $u \in P \setminus M$ if when v was inserted into M there were intersecting D-balls B_u and B_v containing u and v respectively.*

The quality invariant and Theorem 2 guarantee that only a constant number of balls are created or destroyed during an insertion, so the total amount of point location work done on any input point is $O(t)$, where t is the number of times it was touched.

THEOREM 11. *The total cost of point location in the* NETMESH *algorithm is $O(n \log n)$.*

PROOF. As noted before, it suffices to count the number of touches on uninserted input points throughout the algorithm. Since there are only $O(\log n)$ rounds, it will suffice to show that no input point can be touched more than a constant number of times in a single round.

Let M be the mesh at the start of a round. Consider any point $p \in P$. We will show that p cannot be touched more than a constant number of times in this round. By definition, a point x touches p if $\mathbf{d}_{\mathcal{D}_{M'}}(p, x) \leq 1$ in the mesh M' just prior to inserting x. So, it follows that $\mathbf{d}_{\mathcal{B}_M}(p, x) \leq 1$ because D-balls in M' are empty of points of M. Moreover, by Lemma 4, $\mathbf{d}_{\mathcal{D}_M}(p, x) \leq 2$. Therefore, the set of points that can touch p this round are all contained in one of the constant number of D-balls that are within 2 hops of p in $\mathcal{G}_{\mathcal{D}_M}$. In Lemma 12 below, we show that only a constant number of points are added to any D-ball in a single round. Thus, the total number of points that can touch p in a round is at most a constant. □

LEMMA 12. *In any round starting with a mesh M, at most a constant number of points are added to any D-ball of M.*

PROOF. Fix a particular round and let B be a D-ball of M. Let P' denote the input points inserted this round. Let $Q \subseteq P'$ be the points $q \in P'$ such that $\mathbf{d}_{\mathcal{B}_M}(q, x) \leq 5$ for some $x \in B$. It is easily checked that Theorem 2 and Lemma 4 implies that $|Q| = O(1)$.

Let M' denote the mesh at the end of the round. We wish to upper bound the number of points of M' in B. By standard mesh size analysis,

$$|M' \cap B| = \sum_{\Omega \in H_{M'}} O\left(\int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}^{\Omega}_{P' \cup M}(x)^d} \right). \qquad (6)$$

Now, we observe that for some constant α, the points of $P' \cap M'_\Omega$ are α-well-paced with respect to M_Ω. This means that they may be ordered so $p_1, \ldots, p_k$, so that $\mathbf{f}^{\Omega}_{P'_{i-1} \cup M}(p_i) \leq \alpha \mathbf{f}^{\Omega}_{P'_i \cup M}(p_i)$ for each $i = 1 \ldots k$, where $P'_i = \{p_1, \ldots, p_i\}$. As shown in [14], this implies that there is a constant β, depending only on α and d, such that for all i,

$$\int_{\mathbb{R}^d} \left(\frac{1}{\mathbf{f}^{\Omega}_{P'_i \cup M}(x)^d} - \frac{1}{\mathbf{f}^{\Omega}_{P'_{i-1} \cup M}(x)^d} \right) dx \leq \beta. \qquad (7)$$

If $p_i \notin Q$ then $\mathbf{f}^{\Omega}_{P'_i \cup M}(x) = \mathbf{f}^{\Omega}_{P'_{i-1} \cup M}(x)$ for all $x \in B_\Omega \cap B$ (see Lemma 13 below). So, writing the integral from (6) as a telescoping sum in terms of (7) as

$$\int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}^{\Omega}_{P' \cup M}(x)^d} = \int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}^{\Omega}_{M}(x)^d} + \sum_{i=1}^{k} \left(\int_{B_\Omega \cap B} \frac{1}{\mathbf{f}^{\Omega}_{P'_i \cup M}(x)^d} - \frac{1}{\mathbf{f}^{\Omega}_{P'_{i-1} \cup M}(x)^d} \right) dx$$

results in at most $|Q|$ nonzero terms in the sum on the right. So, by (7) and (6),

$$|M' \cap B| = \sum_{\Omega \in H_{M'}} O\left(\beta |Q| + \int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}^{\Omega}_{M}(x)^d} \right). \qquad (8)$$

Moreover, the number of domains in $H_{M'}$ that intersect B is only a constant (see Lemma 14). For each domain Ω intersecting B, $\int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}^{\Omega}_{M}(x)^d} = O(1)$ (see Lemma 15). Applying this fact to (8), implies $|M' \cap B| = O(1)$ as desired. □

LEMMA 13. *Let M be a mesh. For all $\Omega \in H_M$ and all $x, y \in \Omega \setminus M$, if $\mathbf{d}_{\mathcal{B}_M}(x, y) > 5$ then $\mathbf{f}^{\Omega}_{M}(x) = \mathbf{f}^{\Omega}_{M \cup \{y\}}(x)$.*

PROOF. Fix a domain Ω. We will prove the contrapositive. Suppose $\mathbf{f}^{\Omega}_{M}(x) \neq \mathbf{f}^{\Omega}_{M \cup \{y\}}(x)$ for some $x, y \in \Omega$. Then, $B = \text{ball}(x, |x - y|)$ contains at most one point z of M_Ω. Let B_x be the largest ball centered at x that contains no point of M. Let B_y be the largest ball interior tangent to B at y that contains no point of M. If B_x and B_y intersect then we are done. Otherwise, the two balls must be obstructed by a cage around z. It is any easy exercise to show that any two points outer encroaching a cage have ball-distance at most 3 for $\varepsilon < \frac{1}{3}$. So, the total ball-distance from x to y is at most 5. □

LEMMA 14. *Let M and M' be the meshes before and after a round of the* NETMESH *algorithm. For any D-ball B in M', at most a constant number of domains of $H_{M'}$ intersect B.*

PROOF. Let x be the center of B. There are only a constant number of domains of H_M intersecting B, because each contains a D-ball intersecting B and Theorem 2 implies there can only be a constant number of such balls. Any newly created domains must have been caused by the insertion of an input point $y \in M' \setminus M$. However, if the new domain intersects B then either y caused a cage from M to grow or $\mathbf{d}_{\mathcal{B}_M}(x, y) \leq 5$. In either case, there are only a constant number of new domains intersecting B. □

LEMMA 15. *Let M and M' be the meshes before and after a round of the* NETMESH *algorithm. If B is a D-ball of M, then for all $\Omega \in H_{M'}$,*

$$\int_{B_\Omega \cap B} \frac{dx}{\mathbf{f}_M^\Omega(x)^d} = O(1).$$

PROOF. There are four types of domains to consider: the smallest domain Ω such that $B \subset B_\Omega$, domains Ω such that $|M_\Omega| = 0$, domains Ω such that $|M_\Omega| = 1$, and domains Ω such that M_Ω contains an entire cage $C_{\Omega'}$ of some domain $\Omega' \in H_M$. In the first case, the result follows easily from Lemma 10. In the second case, $\mathbf{f}_M^\Omega = \infty$, and thus, the integral evaluates to 0. In the third case, it is easy to evaluate the integral directly using polar coordinates to find that it is constant.

The last case is the interesting one. We use the coarse bounds that $\mathbf{f}_M^\Omega(x) \geq \delta r_{\Omega'}$ for $(1-\delta-\gamma)r_{\Omega'} \leq |x - c_\Omega| \leq 2r_{\Omega'}$ and $\mathbf{f}_M^\Omega(x) \geq \frac{1}{2}|x - c_\Omega|$ for $|x - c_\Omega| > 2r_{\Omega'}$. Integrating with polar coordinates centered at c_Ω yields an answer $O(\log \frac{r_\Omega}{r_{\Omega'}})$. Only a constant number of points in a round may cause Ω' to grow because all but one must lie in the Voronoi cell of C_Ω and thus they are all within a constant D-ball distance of one another. So, $\frac{r_\Omega}{r_{\Omega'}} = O(1)$ and thus the integral also evaluates to $O(1)$. □

11. CONCLUSION AND FUTURE WORK

In this paper, we have given an algorithm for generating quality hierarchical meshes of point sets with size $O(n)$ in $O(n \log n)$ time. We also showed how to extend these hierarchical meshes to traditional well-spaced meshes in optimal output-sensitive time $O(n \log n + m)$. The algorithm and its analysis introduce novel uses of ϵ-nets and the linear-size meshing theory introduced in [14].

Future Work.

We have restricted our discussion to the rarefied case of point set inputs. We expect it should now be possible to design a work efficient algorithm for inputs with higher dimensional features such as segments and faces. The algorithm presented is basically a work efficient parallel algorithm. It should be possible to show the present algorithm runs in polylog parallel time with no increase in work and thus beating the time and work bounds in parallel SVR [12].

Yet another issue is integrating ideas from the NETMESH algorithm into the already relatively fast SVR code [1]. Future experiments in this direction are in order. The algorithm removes the spread term in the run time for the mesh based persistent homology algorithms [11]. It may also have applications for efficient surface reconstruction especially in the higher dimensional cases[6].

12. REFERENCES

[1] U. A. Acar, B. Hudson, G. L. Miller, and T. Phillips. SVR: Practical engineering of a fast 3D meshing algorithm. In *Proc. 16th International Meshing Roundtable*, pages 45–62, 2007.

[2] M. Bern, D. Eppstein, and J. R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[3] M. W. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, 9(6):517–532, 1999.

[4] B. Chazelle. *The Discrepancy Method*. Cambridge University Press, 2000.

[5] K. L. Clarkson. A Randomized Algorithm for Closest-Point Queries. *SIAM Journal on Computing*, 17(4):830–847, Aug. 1988.

[6] T. K. Dey. *Curve and Surface Reconstruction : Algorithms with Mathematical Analysis*. Cambridge University Press, 2007.

[7] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[8] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[9] S. Har-Peled and A. Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. In *Symposium on Computational Geometry*, 2005.

[10] B. Hudson, G. Miller, and T. Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, Birmingham, Alabama, 2006. Long version available as Carnegie Mellon University Technical Report CMU-CS-06-132.

[11] B. Hudson, G. L. Miller, S. Y. Oudot, and D. R. Sheehy. Topological inference via meshing. In *Symposium on Computational Geometry*, 2010.

[12] B. Hudson, G. L. Miller, and T. Phillips. Sparse Parallel Delaunay Refinement. In *19th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 339–347, San Diego, June 2007.

[13] J. Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag, 2002.

[14] G. L. Miller, T. Phillips, and D. R. Sheehy. Linear-size meshes. In *CCCG: Canadian Conference in Computational Geometry*, 2008.

[15] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. On the radius-edge condition in the control volume method. *SIAM J. on Numerical Analysis*, 36(6):1690–1708, 1999.

[16] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370 (electronic), 2000.

[17] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993).

[18] A. C.-C. Yao. A lower bound to finding convex hulls. *J. ACM*, 28:780–787, October 1981.