Sparse Voronoi Refinement

Benoît Hudson **Gary Miller**
Todd Phillips

December 2006
CMU-CS-06-132

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

This work was supported in part by the National Science Foundation under grants CCR-9902091, CCR-9706572, ACI 0086093, CCR-0085982 and CCR-0122581.

Keywords: Meshing, Delaunay, Voronoi, Triangulation, Tetrahedralization

Abstract

We present a new algorithm, Sparse Voronoi Refinement, that produces a conformal Delaunay mesh in arbitrary dimension with guaranteed mesh size and quality. Our algorithm runs in output-sensitive time $O(n \log(L/s)+m)$, with constants depending only on dimension and on prescribed element shape quality bounds. For a large class of inputs, including integer coordinates, this matches the optimal time bound of $\Theta(n \log n + m)$. Our new technique uses interleaving: we maintain a sparse mesh as we mix the recovery of input features with the addition of Steiner vertices for quality improvement.
This technical report is the long version of an article [HMP06] presented at IMR 2006, and contains full proofs.

Contents

1 Introduction

One of the main missing components in current tetrahedral meshing algorithms research is the existence of refinement algorithms with good run time analysis. Runtime analysis for some methods (notably those involving quad-trees or octrees) has been straightforward [MV00, HPU05] due to the very structured spatial decomposition. However, there are many practical meshing algorithms with very poor run time guarantees.

The goal in designing Sparse Voronoi Refinement (SVR) was to create a meshing algorithm that was similar in implementation and style to many widely used meshing algorithms, but with the added benefit of very strong worst-case bounds on the runtime complexity and space usage. An additional achievement of SVR is that the algorithm can work in any fixed dimension d. The most practical implementations of SVR will probably take advantage of fixing $d = 3$, but higher dimensional meshing (spacetime methods, etc.) is a growing area of future research to which SVR can contribute.

The three important aspects of a mesh that are addressed by SVR are that the mesh resolve all the input features (conforming), that mesh elements be well-shaped (quality), and that the number of elements be small (size-guarantee).

Quality: In this work, we will refer to a **quality** mesh as one in which the radius-edge ratio is bounded by a constant for any mesh element. In three dimensions, this metric is somewhat lacking, as it can admit a family of poorly-shaped elements known as *slivers*, although it is known how to post-process a radius-edge mesh and eliminate slivers. On the other hand, this criterion allows better proof generalization and yields strong bounds on the aspect ratio of the *Voronoi Diagram*, the dual of the Delaunay triangulation. Thus most of the SVR operations and metrics are defined on the Voronoi diagram. This vastly simplifies most of the proofs, especially for $d \geq 3$. An actual implementation of SVR would likely only use the Delaunay, but the conceptual framework of the Voronoi is essential to understanding the algorithmic design of SVR. We further discuss mesh quality, Voronoi quality, and the elimination of slivers in Section 2.1.

Conforming: Sparse Voronoi Refinement produces a Voronoi diagram such that the dual Delaunay triangulation *conforms* to the input. That is, every input feature is represented in the output Delaunay mesh by one or several segments, triangles, and higher-dimensional facets. For Delaunay meshes, a careful selection of Steiner points is used to force that the mesh connectivity implicitly conforms to the input.

Size Guarantee:

Let m_{opt} be the number of vertices in the smallest conforming radius-edge quality Delaunay mesh of the input. A meshing algorithm is said to be *size-optimal* if it outputs a mesh of size $O(m_{\text{opt}})$. Viewing meshing as an optimization problem (minimize the number of total vertices, subject to the constraints), a size-optimal algorithm is one that produces a constant-factor approximation to the minimum mesh size. We show that SVR is size-optimal on point sets.

SVR's output mesh is not optimal among all radius-edge meshes in three or more dimensions: consider a perpendicular pair of edges with minium distance ϵ between them. The optimal radius-edge mesh would form a sliver. The optimal aspect-ratio mesh will resolve the feature. We cannot guarantee that we will or that we will not resolve the tiny feature, thus we do not know exactly how to compare SVR's size guarantee to any reasonable optimality condition. Note, however, that this is typical for Delaunay refinement algorithms: our size guarantee is neither better nor worse than the usual.

Time and Space Usage: Given a Piecewise Linear Complex (PLC) as input [MTTW99], we will use n to denote the total number of input features (vertices, segments, triangles and larger facets, etc). We will use L/s to denote the the ratio of the diameter of the PLC to the smallest pairwise distance between two disjoint features of the PLC. The notation L/s is historic, and the concept appears in many works under many names; the current most common name is the *spread* of the input ([Eri01] contains a long list of references).

Sparse Voronoi Refinement has worst case runtime bounded by $O(n \log L/s + m)$, where m is the number of output vertices. This runtime bound is a vast improvement over prior meshing algorithms for three and higher dimensions. For almost all interesting inputs, this bound is equivalent to $O(n \log n + m)$, which is optimal (using a sorting lower bound). SVR also has optimal output-sensitive space usage $O(m)$.

Overview: Most versions of Delaunay refinement algorithms (Section 2 contains a short survey of such methods) first construct the Delaunay triangulation of the input points as a preprocess. Because such algorithms initially construct the Delaunay triangulation, their time and space complexity is bounded from below by the size of the initial Delaunay triangulation (the number of edges, triangles, tetrahedra, etc.) Unfortunately, in dimension higher than two, the complexity of the Delaunay triangulation (that is, the number of edges, triangles, tetrahedra, *etc.*) can be super-linear in the number of input vertices: $\Omega(n^{\lceil d/2 \rceil})$ in the worst case. More concretely, in three dimensions this implies that any such algorithm has worse case space and runtime $\Omega(n^2)$, which is impractical for large inputs – furthermore, some of the classic worst-case examples look very much like engineered surfaces one might want to mesh.

However, it is well known that a quality Delaunay mesh with m vertices has only $O(m)$ elements [MTTW99, Tal97], implying that it should be possible to break the $O(n^{\lceil d/2 \rceil})$ runtime barrier by avoiding the preprocessing step.

Sparse Voronoi Refinement accomplishes this by interleaving the traditional preprocess with the main body of the algorithm. We maintain a Delaunay mesh throughout the algorithm, but enforce that it is *always* a quality radius-edge mesh. The input point set and features are gradually recovered over the life of the algorithm, *rather than at initialization.*

By maintaining the quality of the mesh, SVR ensures that it is *sparse*: the degree of any vertex is at most constant. This allows all the mesh modifications to be performed in constant time, so that the runtime work for the refinement process is only $O(m)$ (Section 8), plus bookkeeping of $O(n \log(L/s))$ term arising from point location costs (Section 9).

Other meshing algorithms that have achieved related runtime bounds are discussed in Section 2.3.

SVR expands the capabilities of existing analyzed meshing algorithms in terms of practical output size and generalization to higher dimension. SVR also represents the first analyzed

3D Delaunay refinement algorithm to achieve the near-optimal bounds we claim. Nonetheless, SVR does not yet completely solve the meshing problem because it has overly strict restrictions on the geometry of the input. Future work in this regard is discussed in Section 10.

Some notation needed to understand the algorithms, proofs, and input restrictions is presented in Section 3.

Our description of SVR will start with a simplified, toy version of SVR that operates on an input point set without input features or boundary concerns. Important new algorithmic and proof techniques are discussed in this section. The goal is to present the salient novel features of SVR, so that the reader can better understand the general method behind SVR and the geometric intuitions behind the proofs.

Having described the simplest case, we will carefully present the full SVR algorithm, with the added complications for handling boundaries and input features. The approaches taken by SVR to address these complexities are highly similar to previous works [MPW02].

The proofs of SVR's spacing and runtime all build on some structural results about quality Voronoi diagrams. We have a section (Section 6) that lays out some proofs of these results, wherein we find that dealing only with the Voronoi diagram dramatically simplifies the life of the theorist. The implementer should keep in mind that all the statements about the Voronoi diagram have equivalent dual statements on the Delaunay triangulation.

2 Related Work

2.1 Mesh Quality and Well-Spaced Points

It has long been known that the Finite Element Method converges to a solution if there are no large (almost 180°) angles in the mesh. The closely related problem of ensuring there are no small (almost 0°) angles in the mesh has proven to be more tractable using the Delaunay triangulation, which has well-understood geometric and topological properties. In general, the smallest angle in the Delaunay triangulation can be still very small, though it is maximal over all possible triangulations; therefore, to ensure a quality mesh, we must add Steiner vertices. When all the angles in a simplex are large, the simplex has good **aspect ratio**, which is variously described as the ratio between the radius (or volume) of the minimum circumscribing ball to the maximum inscribed ball, or the ratio between the volume of the simplex to the length of its shortest edge.

In generalizing to higher dimension, it is much more convenient to use a different definition of mesh quality than the aspect ratio. The **radius-edge ratio** of a simplex is the ratio of the circumradius to the length of the smallest edge of the simplex, where the circumradius of a simplex is the radius of the d-dimensional circumball that passes through its vertices. It was observed more than ten years ago that meshes with good radius-edge have the property that the points from the mesh are well spaced in a precise technical sense. In particular, the Voronoi diagram of the point set has the property that each Voronoi polytope has good aspect ratio (since the Voronoi region may be unbounded one has to carefully define its aspect ratio; see Definition 3.1). This work has motivated research into efficient algorithms to generate these good radius-edges meshes. Indeed, most methods related to Ruppert's Delaunay refinement algorithm generate meshes that explicitly satisfy the radius-edge condition rather than the aspect ratio condition.

It was observed more than ten years ago [MTTW99] that meshes with good radius-edge have the property that the points from the mesh are well spaced in a precise technical sense. In particular, the Voronoi diagram of the point set has the property that each Voronoi polytope has good aspect ratio, since the Voronoi region may be unbounded one must be careful with the definition (see Definition 3.1). This work has motivated research into efficient algorithms to generate these good radius-edges meshes.

In two dimensions, the radius-edge ratio corresponds exactly with the aspect ratio. In three and higher dimension, this is not true: there are simplices with good (small, near one) radius-edge ratio that have bad (large, near infinite) aspect ratio. Because of their shape in 3D, these types of elements are known as **slivers**.

Historically, many algorithms have been proposed that take as input a good radius-edge 3D mesh, and refine it into a sliver-free, good aspect-ratio mesh [Che97, ELM$^+$00, LT01].

In particular, using the results and techniques of our analysis, a very simple timing analysis of the Li-Teng algorithm for sliver removal ([LT01], extended by Li to higher dimension [Li03]) shows that it will work with linear time and space requirements on point sets. Such an algorithm can easily be run as a post-process to SVR in order to generate a sliver-free mesh. Therefore, we henceforth ignore the issue of slivers.

The notion of "slivers" in more than three dimensions is not well understood to our knowledge. This is partly the motivation for SVR to generate a quality Voronoi diagram, a notion which easily generalizes to higher dimensions.

2.2 Delaunay Refinement Algorithms

There have been several different approaches to the meshing problem. The idea of generating a mesh whose size is within a constant factor of optimal was first considered by Bern, Epstein, and Gilbert [BEG94] using a quadtree approach. A 3D extension was given by Mitchell and Vavasis [Mit93].

Chew introduced a 2D Delauanay refinement algorithm [Che89] and showed termination. The quality of the initial triangulation was improved by adding the circumcenters of poor quality triangles as Steiner vertices. This produced a mesh with no small angles, but inserted many more Steiner vertices than necessary.

Ruppert [Rup95] extended this idea of adding circumcenters for 2D meshing to produce a mesh that was within a constant factor in size from the optimal and also handled line segments as input features. The extension of this algorithm to 3D has been ongoing research. Some methods assume that that Ruppert's local feature size function is given [MTT$^+$96]. Others refine a bad aspect ratio mesh directly [She98, MPW02]. These methods by them selves do not give constant approximation size meshes since the may include slivers. But they do produce meshes that have size bounded below by a constant times the local feature size.

Many algorithms for Delaunay refinement in 3D have been proposed [She02, CP03, PW04, CD03], but so far have eluded nontrivial runtime analysis. Trivial runtime bounds such as $O(m^3)$ be found in most cases. Simple examples can usually give bad worst-case performance for naive implementations of these algorithms. As mentioned, they will all suffer from intermediate size $\Omega(n^2)$ in the worst case.

2.3 Optimal Time Meshing Algorithms

Finding refinement algorithms that have provably good run times has also been of interest. Spielman, Teng, and Üngör [STÜ02] proved that Ruppert's and Shewchuk's algorithms can be made to run in $O(\lg^2 L/s)$ parallel steps. They did not, however, prove a work bound. Miller [Mil04] provided the first sub-quadratic time bound in 2D with a sequential work bound of $O((n \lg \Gamma + m) \lg m)$, in which Γ is a localized version of L/s (in particular, $\Gamma \leq L/s$). The extra $\lg m$ factor is related to a priority queue that was required for the runtime proof in the presence of input segments.

Sparse Voronoi Refinement achieves a runtime bound of $O(n \lg L/s + m)$. For any inputs where L/s is bounded by a polynomial of n, this bound is $O(n \lg n + m)$, which is optimal. The most commonly analyzed class of inputs which meet this criterion are inputs with integer coordinates.

Quadtree meshing algorithms in 2D and octree algorithms in 3D have been shown to run in optimal time for integer point-set input [BEG90, MV00, BET99]. While octree algorithms and Delaunay refinement algorithms both produce a mesh which is within a constant factor of the local feature size, Delaunay refinement algorithms have been shown to produce far fewer vertices than octree methods in practice, thus motivating the need for faster Delaunay based algorithms like Sparse Voronoi Refinement.

Recently, Har-Peled and Üngör [HPU05] used a quadtree method as a pre-processing step to produce a size-guaranteed triangulation in optimal time for integer point-set input in 2D. Their algorithm presents a mix between Delaunay refinement methods and quadtree methods. Published versions of this algorithm do not yet handle features or higher dimension. Their use of a quadtree is essentially as an auxiliary structure for point location type operations. We expect that Sparse Voronoi refinement will have an advantage in that the mesh itself is used as a point location structure, requiring less overhead both in runtime and in development time.

2.4 Complexity of the Delaunay Triangulation

There are many simple examples of points sets in three dimensions whose Delaunay triangulation realizes the worst-case quadratic complexity. Some families of 3D input, notably those for which $L/s \in o(n)$, can be shown to have subquadratic Delaunay complexity. However, their worst-case complexity is still $\omega((L/s)^3)$. For example, when $L/s \in \Theta(\sqrt{n})$, then the Delaunay complexity has worst case $\Omega(n^{3/2})$, which is unacceptable for large inputs [Eri01].

Independently, even if the input is favorable enough that the Delaunay has linear complexity, and even in just two dimensions, a poor ordering of refinement steps could still give quadratic runtime [Bar02].

3 Preliminaries

Throughout this paper, unless explicitly stated otherwise, all objects are in $\mathbb{E}^d$. Given a set S, the **convex closure** of S, denoted $CC(S)$, is the smallest convex set containing S, while the **convex hull** is the boundary of $CC(S)$.

Throughout, all **balls** will refer to topologically open balls. We say that a point x **encroaches** on a ball B if $x \in B$; that is, if x is interior to the ball.

A **simplex** is the convex closure of an affine independent set of points. Given a **simplex** it has a well-defined **circumcircle** (the minimum radius circle) containing its points. It in turn has a well-defined **circumradius** and **circumcenter**.

The **diameter** of a compact set is the maximum distance between any pair of points in the set.

A *polytope* is the convex combination of finite set of points P. The dimension of the polytope is the dimension of the affine subspace generated by P. The boundaries as well as the domain we consider are a collection of polytopes. We formally use the formal definition of a Piecewise Linear Complex (PLC) from [MTTW99].

For this paper, we place additional requirements on the input; we expect that most if not all of these can be lifted with additional work drawing on existing techniques. First, we make the usual Draconian requirement that the angle between any two intersecting polytopes, when one is not contained in the other, is at least 90°. We also require that polytopes have a convex hull that is defined by $O(1)$ vertices (although we place no restriction on the number of interior vertices); and furthermore that the hull vertices are well-spaced. Finally, the definition of a PLC requires that all input facets are convex. This would be a minor restriction were input angles unrestricted, because one can always decompose the PLC facets as needed to fulfill this condition. In the Future Work Section 10 we discuss how to lift some of these restrictions.

We define a mesh M as a set of vertices in $\mathbb{E}^d$ and the Voronoi diagram of the vertices. The Voronoi cell of a mesh vertex v is denoted $V_M(v)$. We will call the set of nodes of the Voronoi diagram the Voronoi nodes (these are *not* the vertices of M). We define the **outradius** $R_M(v)$ to be the distance from v to the farthest Voronoi node of its cell. We define the **inradius** $r_M(v)$ to be the distance from v to the point of closest approach of the boundary of the Voronoi cell. When the mesh in question is clear, we may write R_v or r_v for brevity.

Given a set of points P, let $CC(P)$ denote the convex closure of P, i.e., the smallest convex set containing P.

Definition 3.1 *Given a mesh M and a mesh vertex v, the* **aspect ratio** *of the Voronoi cell $V_M(v)$ is $R_M(v)/r_M(v)$.*

We say M has aspect ratio τ if for each vertex $v \in \text{vertices}(M)$ the aspect ratio of $V(p)$ is at most τ. In this case, we will refer to M as a τ-quality Voronoi diagram.

We will assume throughout there are no degeneracies, that is, no sphere of dimension $k \leq d$ containing $k+2$ points. Algorithms to address degeneracies have been considered; incorporating these method into our framework could be easily addressed following Miller, Pav, and Walkington [MPW02].

A crucial definition is that of **local feature size**, as defined by Ruppert [Rup95]. Let $\text{lfs}(x)$ be the minimum distance from any point $x \in CC(P)$ to two disjoint polytopes of the input PLC. We also define a closely related function, the **current feature size** $\text{cfs}_M(x)$, which is the distance from x to the second-nearest mesh vertex of M. We will simply write cfs when it is clear that only one mesh M is involved. This notion of cfs has been written elsewhere as lfs_M, lfs_0, or $\text{lfs}_{0,M}$, however, in this work we will use cfs to strongly disambiguate the two notions. The lfs never changes and counts distance to any polytope. The cfs decreases between intermediate meshes as refinement proceeds, and only counts distance to vertices.

Given any point p in the convex closure of G we consider the lowest dimensional cell containing p. We call this the **containing dimension of** p, denoted $\text{CD}(p)$.

We will need a spacing function defined everywhere for our runtime bounds. We use the *gap size* definition originally defined for mesh coarsening [MTT99, Tal97].

Definition 3.2 *Let P be a point set in $\mathbb{E}^d$. A* **gap-ball** *B of P is any d-ball meeting the following two criteria:*

- *B is not encroached by any point in P.*
- *The center of B lies inside the convex closure $CC(P)$.*

Definition 3.3 *Let P be a point set in $\mathbb{E}^d$. Let x be a point in $CC(P)$. The* **gap size** *$G_P(x)$ is the radius of the largest gap-ball B of P such that x lies on the surface of B.*

For brevity, we define that $G_M(x) \equiv G_{\text{vertices}(M)}(x)$.

Clearly, the gap size is a monotone decreasing function as we add vertices to the mesh: the shape of the convex closure does not change, but it gets harder to satisfy the non-encroachment requirement.

Definition 3.4 *Let a mesh M and a point $x \in CC(M)$ be given. We define the* **grading** *of M at x as:*

$$\Gamma_M(x) = \frac{G_M(x)}{\text{cfs}_M(x)}$$

This notion of grading is useful for capturing the relative quality of a mesh, with the advantage that Γ is defined everywhere in the convex closure, rather than just at vertices like many mesh metrics. This notion and nearly-equivalent notions for grading have used before [MTT99, Tal97, Mil04, HPU05].

Simplified-SVR(P: d-dim point set, τ: mesh quality constant, k: $0 < k \leq 1$)

Assume an initial Voronoi mesh M exists.
Let U be the set of uninserted input vertices: $U = P - V(M)$
while U is non-empty **do**
    Perform a *break* move
    while M has aspect ratio worse than τ **do**
        Perform a *clean* move
    end while
end while

TryInsert(p: d-dim point, M d-dim mesh, U: set of uninserted d-dim points)

if $\exists u \in U \text{s.t.} |pu| \leq kNN_M(p)$ **then**
    Yield: Insert u into M, updating U and the Voronoi diagram.
else
    Insert p into M, updating the Voronoi diagram.
end if

Figure 1: The simplified SVR algorithm on a periodic point set. The break and clean moves are described in the text.

4 Simplified Algorithm

In this section, we describe a simplified version of our algorithm without any input feature complications or boundary concerns. For the simple algorithm, the input is a set of points in the infinite plane $\mathbb{E}^d$ where the hypercube $[0, L^{1/d}]^d$ repeats. While this is not a particularly realistic model of computation, it does allow us to develop the intuition we use for the full algorithm, while avoiding considerable distractions. Furthermore, the periodic point set model can be simulated by embedding the input hypercube within a larger bounding box.

For the purposes of this simplified exposition, we will not discuss initialization, except to note that it is only linear work, assigning uninserted vertices to one of a constant number Voronoi cells. At a basic level, SVR operates incrementally. At any point in time, we have a Voronoi diagram. Some of the input points are Voronoi vertices, some of the input points are not yet recovered. The Voronoi diagram also has Steiner vertices. Every input point that is not yet recovered is contained in some Voronoi cell.

The algorithm iteratively plays one of three moves until none of the moves apply:

- *clean* move: Pick a Voronoi cell of bad aspect ratio. Take its farthest Voronoi node, and try to insert it using TryInsert.
- *break* move on an input: Pick an input vertex that has been recovered in the mesh and whose Voronoi cell includes an input point. Take its farthest Voronoi node, and try to insert it using TryInsert.
- *break* move on a Steiner vertex: Pick a mesh vertex that is not an input (it is a Steiner

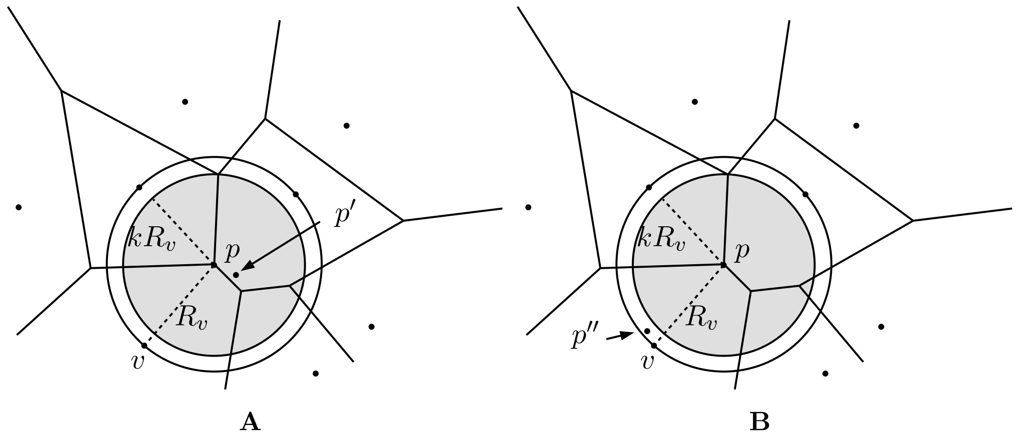

Figure 2: The cell of v is being destroyed by SVR (either because it is skinny, or because p'' is inside); SVR tries to add the farthest Voronoi node p. The outer circle around p is of radius R_v; the inner disc, in grey, is of radius kR_v. **(A)** Because the uninserted input point p' is near p, SVR must yield – otherwise, an artificially small feature will have been created, violating our size guarantees. **(B)** The uninserted input point p'' is too far from p, so SVR does not yield. Our size guarantees would be maintained were SVR to yield to p'', but the time bounds would be violated because p'' may be arbitrarily close to v.

point) and whose Voronoi cell includes an input point. Unconditionally insert one of the input points.

Assuming this algorithm terminates, at that point there will be no cells of bad aspect ratio, and all the input vertices will have been recovered, so the algorithm will have produced a quality conforming mesh.

We add the additional constraint that clean moves take precedence over break moves. One can then think of the algorithm as operating in phases. Each phase, SVR starts with a quality mesh, "breaks" it with a break move; then "cleans" using only clean moves, until quality is re-established.

When the algorithm considers adding a Steiner point, it may instead *yield* to an input vertex if there is one nearby. This ensures that no Steiner point ever is inserted too close to an input vertex, which is critical for guaranteed mesh size. Suppose SVR is cleaning a skinny Voronoi cell $V(v)$ and considers the addition of Steiner p. There is an empty ball around p of radius R_v. This provides a natural definition of "nearby," for yielding purposes. For reasons related to runtime analysis, we must reduce the radius of this neighborhood by a constant factor k, slightly smaller than 1. Expressed concisely, we will yield to an unrecovered input point p' if $|pp'| < kR_v$ (see Figure 2).

Why does the runtime argument need $k < 1$? If we relaxed to $k = 1$, p' might be arbitrarily close to v (or some other vertex). Such a situation would cause an arbitrarily bad break in the quality of the mesh. This would cause us to lose sparsity guarantees, which would in turn destroy runtime guarantees. For this reason, the constants in the runtime analysis will depend on $1/(1-k)$, greatly increasing as k approaches 1. On the other hand, guarantees about spacing and mesh size work best when we are most aggressive about yielding to input vertices rather than adding more Steiner vertices: smaller k makes for larger output. There is a tradeoff here, which we expect argues for setting k close to 1.

The proof that the simplified SVR terminates with guaranteed mesh size is almost verbatim from Ruppert [Rup95], albeit adapted to higher dimension and with an additional case for yielding to input vertices. What is novel are the proofs that bound the runtime and intermediate mesh quality.

Under appropriate settings of the user-given parameters τ and k (namely, $k\tau > 2\sqrt{2}$ and $0 < k < 1$), we can guarantee that M never has aspect ratio worse than some constant τ' that depends only on τ, k, and d – even during the clean and break phases. It is well known [MTTW99, Tal97] that if the mesh vertices are well-spaced, then the Delaunay and its dual Voronoi have size linear in the number of vertices. In particular, every vertex has constant degree (its Voronoi cell has a constant number of facets). This further implies that inserting a vertex into such a mesh will take only constant time using an incremental algorithm; this gives us the $O(m)$ bound for refinement work.

The only charge left, then, is the point location: determining whether any unrecovered vertices are included in the k-ball around p. The point location structure we use is almost trivial: the algorithm simply stores the list of unrecovered input vertices that each cell includes. When a new mesh vertex is inserted, the insertion algorithm also recomputes the lists of affected Voronoi cells. Then, to determine whether any unrecovered vertices are in the k-ball, the algorithm merely queries each neighboring Voronoi cell in turn.

We use an amortized analysis for the point location charges that is reminiscent of the two-dimensional analysis of Miller [Mil04]: we charge the point location not to the Voronoi node that is prompting the test, but to the unrecovered vertex being tested. There are two sub-charges: the relocation charge for updating point location information, and the charge for point location queries. Both charges are computed similarly, so in this section we will merely sketch out the former case.

Consider an unrecovered vertex u. Whenever the Voronoi cell that contains u changes, a new vertex p was inserted nearby – at a distance related to (within a constant factor of) $\mathrm{cfs}(u)$. Furthermore, when p is inserted, SVR ensures via the yielding rules that the nearest neighbor of p is no closer than a distance related to $\mathrm{cfs}(p)$. Finally, because u and p are close, $\mathrm{cfs}(p)$ is related to $\mathrm{cfs}(u)$. Thus, for every p whose insertion affects u, p is both close (distance $O(\mathrm{cfs}(u))$ and has a large (radius $\Omega(\mathrm{cfs}(u))$) empty ball around it. Therefore, an adversary can only pack a constant number of vertices around u before the feature size at u falls by half.

Later, we will prove that $\mathrm{cfs}(u)$ is at most the diameter L of the periodic region, and never goes below $\Omega(s)$. Thus, the adversary can force the algorithm to halve the feature size at most $\log(L/s)$ times around u. So the number of times that u is relocated will be at most $O(\log(L/s))$. Precise statements of the lemmas are found in Section 9.

As we develop the full algorithm, the important techniques of this section will remain the same: as long as SVR ensures that no new vertex is ever inserted too close to an existing mesh vertex, the algorithm always has a quality mesh (not too broken). In turn, this implies that the insertion is fast and that the gap size around unrecovered features falls exponentially, so that the point location costs are also small.

5 Full Algorithm

This section describes the full algorithm in excrutiating detail. The input is described as a PLC, and must meet the criteria described in Section 3.

As scratch data structures, we maintain the following:

- A mesh M_F for *every feature* F. The entire domain Ω is treated as merely the biggest feature.
- A set of work queues. Elements on each work queue are balls – triples of a Voronoi node c, its associated radius r (which is equal to $NN_{M_F}(c)$ when the ball is created), and the feature F which contains c.

  The work queues are of three types: SKINNY, ENCROACHED, and UNRESOLVED and are processed in that order. For each type, we maintain an independent queue for each dimension. Skinny elements are processed from lowest dimension to highest; the other elements are processed in the reverse order.
- A bipartite mapping for each dimension i between balls of lower dimension $j < i$, which must be protected, and Voronoi cells of dimension i. The mapping is maintained in the trivial way with the operations LINK and UNLINK. Operation $\text{CELLS}_i(b)$ returns the set of cells in dimension i associated with a ball, and $\text{BALLS}_i(v)$ returns the set of balls associated with a Voronoi cell $V(v)$ in dimension i.

In the bootstrapping phase of the algorithm, we INITIALIZE the data structures for each feature. We generate the Voronoi diagram of the convex hull of each feature F, then distribute each sub-feature's protecting balls among the cells of M_F, and we populate the work queues. Because we have required each feature to have a small, well-spaced convex hull, initialization runs in linear time and produces quality meshes for each feature.

In a nice input (as we assume we are given), the initial SKINNY and ENCROACHED work queues will all be empty. However, M_Ω will not correspond to the desired input. This motivates the need for the UNRESOLVED queues. An i-ball is defined by $i + 1$ vertices. We say an i-ball is *resolved* in M_F if it appears as a Delaunay subfacet of M_F – that is, if all $i+1$ vertices are, pairwise, each other's neighbours. Thus a 0-ball is resolved if the corresponding vertex is in the mesh; a 1-ball is resolved if both its vertices are in the mesh, and also the corresponding edge links two neighbours in the Voronoi graph; etc. There are two reasons for the UNRESOLVED queues: (1) to recover vertices; (2) to recover single-encroached features, as described below.

As in the point-set case, the algorithm will now iteratively perform *break* and *clean* moves until it reaches a quality conformal mesh. These are encoded in the work queues: a

d	The ambient dimension.
$\mathcal{P}$	The input piecewise linear complex.
$\mathcal{P}_i$	The i-dimensional facets of $\mathcal{P}$.
Ω	The input domain (a convex subset of $\mathbb{E}^d$).
F	A feature: a facet of $\mathcal{P}$ (vertex, edge, ..., Ω).
F^+	A feature that contains F.
M_F	The current refined mesh of F.
c	A Voronoi node (the circumcenter of a simplex).
V	A set of Voronoi cells.
v	A single Voronoi cell.
b	A protective ball represented as a triple $\langle (c, r), F\rangle$. Geometric tests performed on b are performed on $B(c, r)$; F is the feature that b is helping protect.
S	The set of sub-balls that intersect a region.
Y	The set of sub-balls to which an insertion must *yield*.
C	The set of Cells that will be modified by the insertion of a point.
C'	The set of Cells that were modified by the insertion of a point.

Table 1: List of variable names used in the SVR algorithm.

break move draws from an ENCROACHED or UNRESOLVED queue; a clean move draws from a SKINNY queue. However, we must sequence the moves correctly across dimensions. In particular, we maintain the invariant that every mesh is always of good quality. We first refine for quality (clean moves) from the bottom (lowest dimension) up. Next, we refine for input (break moves) from top down. This approach is critical to establishing the runtime bound, though it is irrelevant to the correctness proof: a buggy order will slowly produce the correct result.

When we consider adding a Steiner point in SPLIT, as before, we may have to yield to an input vertex and FORCEINSERT it instead of the Steiner point. However, we may also have to yield to lower-dimensional input features: otherwise a Steiner point could be added very close to a feature, defeating the sizing proofs. Furthermore, the ball b being SPLIT will still exist after yielding: inserting new vertices into a lower-dimensional mesh does not affect F. This explains the goto in line 15 of SPLIT. The reinvocation of SPLIT is likely to yield to a new vertex inserted by one of the lower-dimensional facets, or to instead find itself inserting the original circumcenter.

The set of lower-dimensional balls discovered in SPLIT may be very large: of linear size. Therefore, each iteration may take linear time. We cannot afford to discover the encroached subballs one by one; instead we must either deal with *all* encroached subballs simultaneously or perform additional bookkeeping.

Unlike in standard Delaunay mesh refinement techniques, for runtime reasons we are required to occasionally allow input protective balls to be encroached by mesh vertices; however, we only allow a single encroachment, and only by vertices of an equal- or lower-dimensional feature (including by input points). The need for this is based on the fact that

SVR($\mathcal{P}$: a piecewise linear complex)

Initialize($\mathcal{P}$)
while workqueues are not empty **do**
    b = DEQUEUE()
    SPLIT(b)
end while
return M_Ω

Figure 3: The mainline of the SVR algorithm. We simply process events, denoted by the protective ball that needs to be split.

SPLIT(b: a ball $\langle (c, r), F\rangle$)

1: **if** c is no longer a Voronoi node of M_F **then return**
2: compute V: the set of Voronoi cells that intersect $B(c, r)$
3: compute $S = \cup_{v \in V}$BALLS(v)
4: **if** $\exists b' \in S$ such that b' is a vertex-ball and $b' \in B(c, r)$ **then**
5:     b' represents a point p
6:     FORCEINSERT(b, p)
7: **else**
8:     compute Y, the set of balls in S encroached by c
9:     **if** Y is empty **then**
10:         FORCEINSERT(b, c)
11:     **else**
12:         **for** each $b' \in Y$ **do**
13:             SPLIT(b')
14:         **end for**
15:         **goto** 2
16:     **end if**
17: **end if**

Figure 4: How to split a protective ball. Clearly, if the ball is no longer being protected, we no longer need to split it (and, indeed, we must not). Otherwise, look for a nearby vertex if there is one; if so, yield to it and insert the vertex (line 6). If not, look for nearby lower-dimensional protective balls that we encroach. If there are some, split them all (line 13). Splitting them will create new vertices we may want to yield to, or new protective balls we encroach, so try splitting b again (line 15).

ForceInsert(b: a ball $\langle (c, r), F \rangle$, p: a point)

Starting at c in M_F, compute the set of cells C that p will change.
Create C': the reshaped cells, plus the new cell for p.
Update M_F, replacing C by C'
Compute $S = \cup_{v \in C}$Balls(v)
$\forall v \in C$, Unlink(v, Balls(v))
for all $v' \in C'$ **do**
    compute $S' = \{b \in S | b \cap v' \neq 0\}$
    AddLink(v', S')
end for
for all $v' \in C'$ **do**
    if v' is a skinny Voronoi cell **then**
        let (c', r') be the farthest Voronoi node of v'
        Enqueue($\langle (c', r'), F \rangle$, skinny, $\dim(F)$)
    end if
end for
for all dimensions $i > \dim(F)$ **do**
    ReplaceBalls(C, C', i)
end for

Figure 5: The ForceInsert routine actually inserts a point p into a mesh for a feature F. First, we update the mesh M_F; then we update the auxiliary data structures for F. Next, we add any new Skinny events (AddLink handles the other types of events). Finally, we add the information to all the higher-dimensional meshes that contain F.

INITIALIZE($\mathcal{P}$: a piecewise linear complex)

let $\mathcal{P}_i$ be the i-dimensional features
for $i = d$ **downto** 0 **do**
    for each feature $F \in \mathcal{P}_i$ **do**
        create the Voronoi tesselation $M_F = \text{Voronoi}(\partial F)$
    end for
end for
for each feature $F \in \mathcal{P}$ **do**
    for each Voronoi node $(c, r) \in M_F$ **do**
        create ball $b = \langle (c, r), F \rangle$
        for each feature F^+ that contains F **do**
            ADDBALL(F^+, b)
            if b is encroached by a vertex of M_{F^+} **then**
                ENQUEUE(b, ENCROACHED, $\dim(F^+)$)
            end if
            if b is not resolved by M_{F^+} **then**
                ENQUEUE(b, UNRESOLVED, $\dim(F^+)$)
            end if
            $\forall v \in M_{F^+}$ that intersect b, ADDLINK(v, b)
        end for
    end for
end for

Figure 6: Initialization proceeds by building the Voronoi diagram of the convex hull of each feature, then building the intermediate structures in bottom-up fashion. Because of the assumption that the convex hull has constant size, we can loop over all the cells of any feature's mesh in constant time.

```
ReplaceBalls(B, B', i)
cells ← ∪_{b∈B}(Cells_i(b))
for all b ∈ B do
   for all v ∈ cells do
      RemoveLink(v, i, b)
   end for
end for
for all b' ∈ B' do
   for all v ∈ cells do
      if V(v) intersects b', then AddLink(v, i, b')
   end for
   if b is not resolved by cells, Enqueue(b, unresolved, i)
end for
```

Figure 7: After a mesh inserts a point, all higher-dimensional meshes need to be informed of the new point and the new protective balls; this may add items to the work queues.

```
AddLink(v, i, b)
   Add v to Cells_i(b)
   Add b to Balls_i(v)
   if v encroaches b then
      if v is a Steiner point then
         Increment SteinerEncroachers(b)
      else
         Increment InputEncroachers(b)
      end if
      if SteinerEncroachers_i(b) > 0 or InputEncroachers_i(b) > 1 then
         Enqueue(b, encroached, i)
      end if
   end if
```

Figure 8: AddLink and RemoveLink maintain the mapping between protective balls and cells, along with testing whether the protective ball is encroached and needs to be fixed. RemoveLink is symmetric to AddLink and thus we leave it as an exercise to the reader.

```
DEQUEUE
  for i = 0 to d do
    pop and return if workqueue(SKINNY, i) is non-empty
  end for
  for i = d downto 0 do
    pop and return if workqueue(ENCROACHED, i) is non-empty
  end for
  for i = d downto 0 do
    if workqueue(UNRESOLVED, i) is non-empty then
      b ← pop one element off the queue
      using CELLS_i(b), check if b is now resolved
      if not, return b
    end if
  end for
```

Figure 9: Dequeue accesses the work queues in the order described in the text. Elements never become non-skinny unless the farthest Voronoi node was removed, in which case it is no longer in the mesh. Similarly, encroached balls never become unencroached. However, unresolved elements may have become resolved. We check here so that calling code need not reason about why any ball is being processed.

while we can charge according to the spread (L/s) of the input for the point location of features the user input, we want the charge on sub-features the algorithm creates to be only linear in the size of the output. Allowing singly-encroached input features allows the algorithm to ensure it never performs point location on any non-input protective balls until the gap size around the protective ball is related to its diameter.

Encroachment is normally disallowed so that the encroachment test becomes a cheap test for ensuring that input features appear in the output. However, if the input feature does not appear in the output, it will be on the UNRESOLVED queue. Therefore there are no worries that allowing single encroachment will cause the output to misconform.

Without allowing single-encroachment, the $\lg(L/s)$ term is applied to the output size, rather than only the input, which would cause the algorithm to run in $O(m \lg L/s)$. It is unclear whether the distinction matters in practice.

6 Structural Properties of Quality Voronoi Diagrams

In this section we present several structural lemmas about good radius-edge meshes. These lemmas are crucial for our analysis and may be well suited for timing analysis of future algorithms.

6.1 Feature size in gap balls

Let M be a τ-quality Voronoi diagram in $\mathbb{E}^d$. Our first main goal in this section is to show that the cfs of any point in a gap-ball is bounded below by a constant times the ball's radius.

Lemma 6.1 *Suppose that M is a τ-quality Voronoi diagram, and suppose that B is a gap-ball of M with center c and radius r. If $x \in B \cap CC(M)$ then*

$$\mathrm{cfs}(x) \geq c_{6.1} r$$

where $c_{6.1}$ depends only on τ and is independent of dimension.

To prove 6.1, we first need a technical fact that is most likely not new.

Lemma 6.2 *If $q \in CC(M)$ and $q \in V_p$ for some point $p \in M$ then there exists a Voronoi point v on V_p such that $|qp| \leq |vp|$*

Proof: Let $q \in CC(M) \cap V_p$ as in the hypothesis. We may assume that the affine dimension $CC(M)$ is d for otherwise we may project the problem into a lower dimensions space. WLOG we may assume that p is the origin. Consider the linear programming problem with constraints taken as the facets of V_p and objective function $\max q^T x$. We know that the problem is feasible. Thus, there are two possibilities for the solution: (1) it is a finite zero-dimensional cell, or (2) there is an unbounded sequence of solutions. In the first case the solution is a Voronoi point of V_p which must have a distance to p of at least $|qp|$. We show the second case is not possible by way of a contradiction. Suppose that u is a vector such that (a) $q^T u > 0$ and (b) all point on the array determined by u are in V_p. By condition (b) the open half-space $\{x \mid x^T u > 0\}$ is disjoint from $CC(M)$. But q is in this open half-space, contradicting the fact that $q \in CC(M)$. Thus case (2) is not possible. ∎

As a corollary we get the following fact that we will use repeatedly:

Lemma 6.3 *Consider a point $x \in V_M(v) \cap CC(M)$ for an arbitrary mesh M and an arbitrary vertex $v \in M$. Then:*

$$r_M(v) \leq \underset{M}{\mathrm{cfs}}(x) \leq 2R_M(v)$$

Proof: The nearest neighbour of x is v by definition. The closest x can be to a vertex v' other than v is when x is on the boundary of $V_M(v)$, since that point is equidistant from v and some v'; then $\mathrm{cfs}_M(x) = |xv|$. To minimize $|xv|$ while still being on the boundary, we choose the point of closest approach of the boundary to v, which by definition is at $|xv| = r_M(v)$.

For the upper bound, walk from x to its nearest Voronoi node; this is a distance of at most $R_M(v)$ from Lemma 6.2. This node is at distance at most $R_M(v)$ from at least $d+1 > 2$ vertices, thus we have a path of length at most $2R_M(v)$ from x to more than two vertices. ∎

We will proceed toward proving Lemma 6.1 by starting with a few simple observations. First, suppose that q is in a gap-ball B of radius r and center c. It follows that $\mathrm{cfs}(q) \geq r - |qc|$.

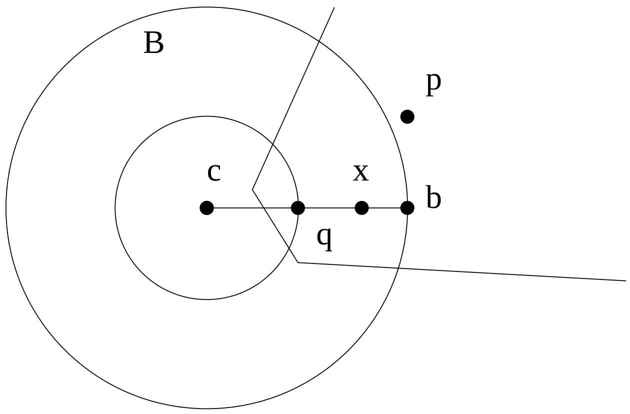

Figure 10: A Ball B such at V_p contains q and b.

Second, suppose that q in the the Voronoi cell V_p. It follows that $\mathrm{cfs}(q) \geq |pq|$ and $\mathrm{cfs}(q) \geq r_p$. To see the second inequality we may assume that $|pq| \leq r_p$. Consider any other $p' \in V(M)$ be given with $p' \neq p$. We know that $|pp'| \geq 2r_p$ and by the triangle inequality $|pq| + |qp'| \geq |pp'|$. Thus $|qp'| \geq r_p$.

We now use these two observations for the proof of Lemma 6.1:

Proof: Let B be a ball and M a Voronoi diagram as in the hypothesis of the lemma. By scaling it will suffice to prove the lemma for $r = 1$ and show that the local feature size is bounded below by a constant.

Further, let $x \in B \cap CC(M)$. If $|xc| < 1/2$ then $\mathrm{cfs}(x) \geq 1/2$ by the first observation above. Thus we may assume that $|xc| \geq 1/2$.

Consider the ray $\overrightarrow{cx}$. Let q and b be the points on $\overrightarrow{cx}$ at a distance 1/2 and 1 from c. It will suffice to bound from below the local feature size of all points on $\overrightarrow{cx}$ between q and b in $CC(M)$. Let V_p be a Voronoi region containing q. Let R_p and r_p be the inradius and outradius of V_p. See Figure 6.1.

Since $x \in CC(M)$ and $c \in CC(M)$, it follows by convexity that $q \in CC(M)$. Thus by Lemma 6.2 we know that $|qp| \leq R_p$. There are two cases depending on whether or not $b \in V_p$.

Case 1: Suppose that $b \in V_p$. Since both q and b are in V_p it follows that $x \in V_p$ and thus $\mathrm{cfs}(x) \geq r_p$. Since $p \notin B$, we know that $1/2 \leq |qp|$. using the fact that V_p has bounded aspect ratio we get:

$$\mathrm{cfs}(x) \geq r_p \geq R_p/\tau \geq \frac{1}{2\tau}$$

Case 2: Suppose that $b \notin V_p$. Thus $|bp| > r_p$. We next will prove a lower bound on $|qp|$. WLOG we can write c, q, b, and p in Cartesian coordinate. The proof will hold any dimension but for the simplicity of the exposition we will only consider the plane containing the points c, q, b, and p. Thus we may assume that $c = (0, 0)$, $q = (1/2, 0)$, $b = (1, 0)$, and $p = (x, y)$.

Since we are lower-bounding $|qp|$, WLOG we may assume that $|cp| = 1$, i.e., $x^2 + y^2 = 1$. We get the following chain of inequalities:

$$|qp| \leq R_p \leq \tau r_p \leq \tau |bp|$$

Define $d := |qp|$. Consider the calculation of d^2

$$d^2 = (x - 1/2)^2 + y^2 = 5/4 + x \tag{1}$$

Here we used the fact that $x^2 + y^2 = 1$. By a similar calculation we get that $(|bp|)^2 = 2 - 2x$ Thus $(5/4 - x) \leq \tau^2(2 - 2x)$, i.e.

$$x \leq 1 - \frac{1}{4(2\tau^2 - 1)} \tag{2}$$

Using Equations 1 and 2 we get:

$$d^2 \geq 5/4 - (1 - \frac{1}{4(2\tau^2 - 1)}) = 1/4(1 + \frac{1}{2\tau^2 - 1}) \tag{3}$$

If we write $d = 1/2 + \delta$ then $\text{cfs}(x) \geq \delta$. Using Equation 3 and the fact that $\delta^2 + \delta + 1/4 = d^2$ we get that

$$\delta^2 + \delta \geq \frac{1}{4(2\tau^2 - 1)} \tag{4}$$

Thus by the quadratic formula we get that

$$\delta \geq \frac{-1 + \sqrt{1 + \frac{1}{2\tau^2 - 1}}}{2} \tag{5}$$

Thus for $\tau > 1/\sqrt{2}$ the local feature size at x is bounded away from zero and the bound is only a function of τ. $\blacksquare$

We now a state a lemma that is functionally equivalent to Lemma 6.1, but allows the gap-ball to have one vertex encroaching it.

Lemma 6.4 *Suppose M is a τ-quality Voronoi diagram and p is a Voronoi vertex. Define M' as the Voronoi diagram of $V(M)\backslash\{p\}$. Suppose B is a gap-ball of radius r in M', then for all $x \in B$:*

$$\underset{M}{\text{cfs}}(x) \geq c_{6.4} r$$

Proof: For the proof, we will simply remove p, show that we can still apply Lemma 6.1, and then put p back in, and show that the result is not much changed.

Claim: M' is a $((2 + \tau)\tau)$-quality Voronoi diagram. Consider any vertex v in M' whose cell V_v is different from it's cell in M. Then v must have been adjacent to p. We argue that V_v still has relatively good aspect ratio.

First we note that $r_{M'}(v) \geq r_M(v)$.

Second, we have by adjacency that $R_{M'}(v) \leq 2R_M(v) + R_M(p)$.
Since v and p were neighbors, we have $R_M(p) \leq \tau R_M(v)$.
From which it follows $R_{M'}(v) \leq (2+\tau)R_M(v)$.
Since $R_M(v)/r_M(v) \leq \tau$ by assumption, we then combine inequalities to obtain:

$$\frac{R_{M'}(v)}{r_{M'}(v)} \leq (2+\tau)\tau$$

So the aspect ratio of V_v is still quality.

Since we have shown that M' is quality, we can then apply Lemma 6.1 to a given $x \in B$, and obtain:

$$\operatorname*{cfs}_{M'}(x) \geq c_{6.1} r$$

Where $c_{6.1}$ depends only on $(2+\tau)\tau$.

We now claim that $\mathrm{cfs}_{M'}(x) \leq 2\tau\, \mathrm{cfs}_M(x)$.
Let q be the Voronoi vertex of M such that $x \in V_q$. Let w be the nearest Voronoi *node* to x. By construction, w must be on V_q, so $|qw| \leq R_M(q)$.
We use the fact that the cfs of a Voronoi node is not changed by the single deletion of a Voronoi vertex, since it is equidistant to $d+1$ vertices.
i.e. $\mathrm{cfs}_{M'}(w) = \mathrm{cfs}_M(w) = R_M(q)$. We have:

$$\begin{aligned}\operatorname*{cfs}_{M'}(x) &\leq \operatorname*{cfs}_{M'}(w) + |xw| \leq \operatorname*{cfs}_{M}(w) + R_M(q)\\ &\leq 2R_M(q) \leq 2\tau r_M(q) \leq 2\tau \operatorname*{cfs}_{M}(x)\end{aligned}$$

Summing up, we combine these inequalities to find that:

$$\operatorname*{cfs}_{M}(x) \geq \frac{c_{6.1}}{2\tau} r = c_{6.4} r$$

for a suitably defined constant $c_{6.4}$, where $c_{6.1}$ in this equation depends on $(\tau+2)\tau$. ■

6.2 Grading guarantees

We now state two lemmas that relate the grading Γ of a Voronoi diagram to the quality τ. These lemmas are not really new, but we include more simple formulations than prior work.

Lemma 6.5 (Quality Gives Bounded Grading) *Suppose M is a τ-quality Voronoi diagram. Then there exists a constant $c_{6.5}$ depending only on τ such that $\Gamma_M(x) \leq c_{6.5}$ for any $x \in CC(M)$.*

Proof: Let B be any gap-ball with x on the surface. By Lemma 6.1, so we obtain that:

$$\operatorname*{cfs}_{M}(x) \geq c_{6.1}(\tau) G_M(x)$$

But then clearly,

$$\Gamma_M(x) \leq \frac{1}{c_{6.1}(\tau)}$$

So we define $c_{6.5}(\tau) = 1/c_{6.1}(\tau)$ and we are finished. ∎

Lemma 6.6 (Bounded Grading Gives Quality) *Suppose we have a Voronoi diagram M, and suppose that $\Gamma_M(p) \leq \rho$ at every vertex $p \in M$. Suppose further that every Voronoi node of M is contained in $CC(M)$. Then M is a 2ρ-quality Voronoi diagram.*

Proof: Let $p \in M$ be given. R_p is the distance from p to v, the circumcenter of some Delaunay simplex. Consider the ball centered at p of radius R_p. Given our assumptions, this is a gap-ball, so we have:

$$R_p \leq G_M(p)$$

Note further that $2r_p = NN_M(p)$, so it follows that:

$$\frac{R_p}{r_p} \leq \frac{2G_M(p)}{NN_M(p)} = 2\Gamma_M(p) \leq 2\rho$$

Hence V_p has aspect ratio at most 2ρ. ∎

We note that the hypothesis for Lemma 6.6 is satisfied when we have a τ-quality mesh such that the diametral ball of every mesh simplex on the convex hull is unencroached.

6.3 Packing lemma

We can prove a Lemma that relates to packing a set of some vertices around a given center. This general notion will be employed in several of the proofs about quality meshes and Voronoi diagrams. Intuitively, the distance between these vertices as we move away from the center increases geometrically, like in a quality mesh. If we take a set of annuli whose radii incrementally double, Lemma 6.7 shows that there will be a constant number of vertices in each annulus.

Everywhere that a term related to $\log L/s$ appears in the analysis of SVR is a result of this lemma. In particular, the radii of the annuli generally range from L down to s and decrease by a factor of two each time, so there will be $\log L/s$ total annuli, each containing a constant number of vertices.

Lemma 6.7 *Consider an annulus of radii (R_1, R_2) in d dimensions. Consider a family of balls where each ball has its center within the annulus; each ball has radius at least r; and no ball includes any other ball's center. The number of such balls that fit is bounded by:*

$$\left(2\frac{R_2}{r}+1\right)^d - \left(2\frac{R_1}{r}-1\right)^d \quad \textit{For } R_1 > r/2$$

$$\left(2\frac{R_2}{r}+1\right)^d \quad \textit{For } R_1 \leq r/2$$

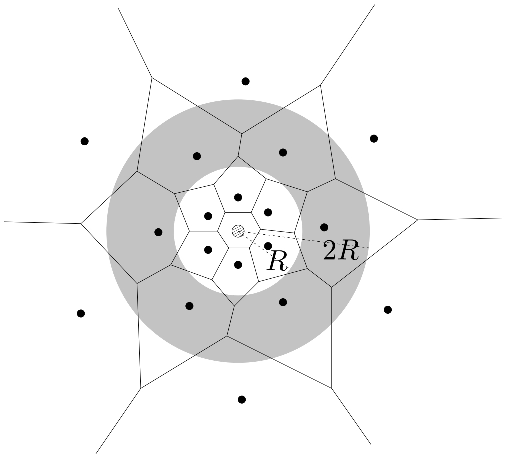

Figure 11: An illustration showing the intuition behind Lemma 6.7 and its later uses. A quality Voronoi diagram packs vertices around the small central feature. There are only a constant number of vertices in each radius-doubling annulus (shown in gray).

Proof: First, reduce the radius of every ball to exactly r. Clearly this can only reduce the volume of intersection between each ball and the annulus, increasing the number of balls that can fit.

Next, halve the radius of each ball. This yields a packing of the balls: no two balls intersect except at a point. Again, this can only reduce the volume of each ball.

Let V_d [1] be the volume of the unit sphere in d dimensions. A shrunken ball therefore has volume $a = V_d(r/2)^d$.

One of these shrunken balls on the boundary of the annulus can only enclose space within distance $r/2$ of the annulus boundaries; therefore, if we grow the annulus by $r/2$ in both directions, we will have an object that completely includes all the balls. The enlarged annulus has volume $A = V_d(R_2 - r/2)^d - V_d(\max(R_1 - r/2, 0)^d$.

Because the shrunken balls are non-overlapping and are completely enclosed in the enlarged annulus, we have at most A/a balls that can fit, which, after some algebra, yields the result. ∎

6.4 Degree bound

The lemmas just presented allow a generalization of the bounded-ply theorem of Miller *et al.* [MTTW95].

Lemma 6.8 (Quality gives bounded degree) *Suppose we have a mesh M of quality τ. Then there is a constant Δ depending only on τ and d such that any empty ball centered in $CC(M)$ intersects at most Δ cells of M.*

[1] $V_d = \frac{2\pi^{d/2}}{d\ \Gamma(d/2)}$, where Γ is the recursive function $\Gamma(d+1) = d\ \Gamma(d)$. This is not the same Γ as we have been using for the grading of a mesh. This is the *real* Γ that all those mathematicians use.

Proof: Let us name the empty ball: $b = B(c, r)$.

A Voronoi cell $V_M(v)$ that intersects b has some point $x \in b \cap V_M(v)$. By Lemma 6.3 we know that $\text{cfs}_M(x) \leq 2R_M(v)$. By Lemma 6.1 we also know that $\text{cfs}_M(x) \geq c_{6.1}r$.

In other words, any Voronoi cell that intersects b is defined by a vertex at distance $O(r)$ from c, and has in-radius $\Omega(r)$. By the Packing Lemma, there can only be $O(1)$ such objects. ■

This implies that the Delaunay graph has bounded ply; the Voronoi diagram has bounded degree; and, the reason for our generalization, any lower-dimensional protective ball is only watched by a bounded number of higher-dimensional Voronoi cells.

The bound holds if we allow $O(1)$ points to encroach on a ball also: removing $O(1)$ points from a τ-quality mesh clearly will give a bounded-quality Voronoi diagram M' (of quality exponentially worsening with the number of removed points). The ball will at worse encroach every cell it encroaches in M', plus the cells of the $O(1)$ points, which is still a constant.

7 Spacing

The key lemma that Ruppert used in his paper stated that the algorithm will never insert two points more than a constant factor closer to each other than what is dictated by the lfs function over the input – in fact, it will never even *consider* inserting a point too close to a neighbour. By controlling the spacing of output vertices, Ruppert could then prove that his algorithm terminates with a mesh within a constant factor of the optimal size. We will do the same; we also need to control the spacing in order to achieve our time bounds.

The statement of the theorem is as follows:

Theorem 7.1 (Spacing Theorem) *For any point p considered for insertion into a mesh M, whether or not p is eventually inserted, we have that:* $\text{lfs}(p) \leq C NN(p)$

The proof follows in the next two subsections.

From Theorem 7.1 we get a corollary which allows us to bound the nearest neighbour of a mesh vertex at any time – in particular, at the end of the algorithm:

Corollary 7.2 *In any mesh M produced during the run of the algorithm, for any mesh vertex $v \in M$, we have that:* $\text{lfs}(v) \leq (1 + C)NN_M(v)$

Proof: At the time we inserted v, we had (by the theorem) that $\text{lfs}(v) \leq C NN_{M_v}(v)$. If it is the case that $NN_{M_v}(v) = NN_M(v)$ then the statement is proven. Otherwise, consider v' the nearest neighbour of v in M. When v' was inserted, $\text{lfs}(v') \leq c NN_{M_{v'}}(v')$ again by the induction hypothesis. Clearly, $NN_{M_{v'}}(v') \leq |vv'|$ since v existed in $M_{v'}$. Ergo, $\text{lfs}(v') \leq C NN_M(v)$. Plugging into the Lipschitz condition:

$$\begin{aligned} \text{lfs}(v) &\leq \text{lfs}(v') + |vv'| \\ &\leq C|vv'| + |vv'| \\ &\leq (1 + C)|vv'| \end{aligned}$$

And $|vv'| = NN_M(v)$ by definition of v'. ■

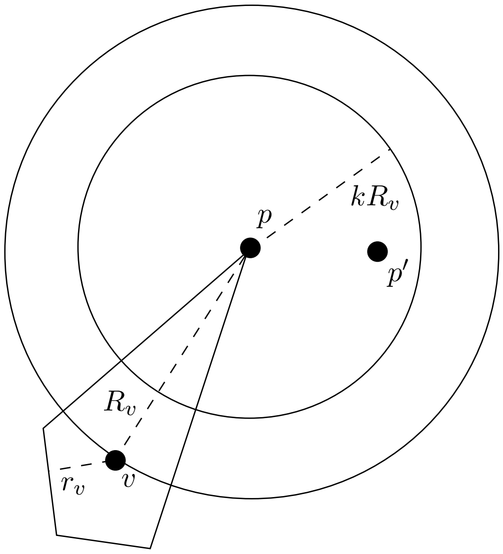

Figure 12: Destroying the skinny Voronoi cell of v in the clean phase. SVR tries to add the farthest Voronoi node p, but it might yield to some input point p' that is nearby p.

7.1 PPS Model

The simplest case is for the periodic point set algorithm of Section 4.

The algorithm can be seen as iteratively considering a sequence of points for insertion, inserting some and discarding others; this produces a sequence of meshes. The proof of the theorem will be by induction on that sequence. We denote by M_p the most recent mesh, M or earlier, in which point p was considered for insertion.

The proof of the theorem proceeds by cases. We consider two constants C_0 and C_d, the first relating to input vertices, the second to Steiner points. We shall constrain these constants (in boxes for ease of reading), and show that the set of constrains admits a solution. The cases are according to the following decision tree:

- If p is a Steiner point, then:
  - If the algorithm is executing a break move, then

$$\mathrm{lfs}(p) \leq 2NN_M(p)$$

Proof: If the algorithm is attempting to insert a Voronoi node during a break move, then the cell being refined using p must be the Voronoi cell of an input vertex; and the cell must contain a second input vertex. Therefore, there are two input vertices within the diameter of the cell, which can be used to defined the lfs(p). But $NN_M(p)$ is no larger than half the cell diameter. ■

This case constrains $\boxed{C_d \geq 2}$

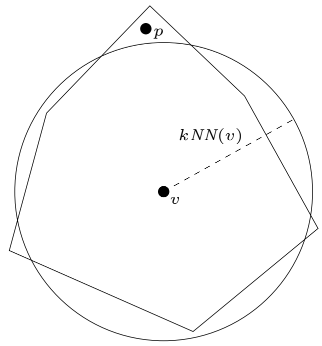

Figure 13: Inserting an input vertex close to an input.

- Otherwise, the algorithm is executing a clean move, and

$$\text{lfs}(p) \leq (2\tau^{-1}(1+C)+1)NN_M(p)$$

Proof: Corollary 7.2 tells us that by induction, $\text{lfs}(v) \leq (1+C)NN_M(v)$. Meanwhile, since the cell is ill-shaped, we know that $R(v)/r(v) > \tau$, thus $NN_M(v) < R(v)/\tau$. Furthermore, p is the farthest Voronoi node, so $R(v) = |vp| = NN_M(p)$. Using the Lipschitz condition, we know

$$\text{lfs}(p) \leq \text{lfs}(v) + |vp|$$

Substitutions lead to the desired result. This is exactly analogous to Ruppert's proof. ■

This case constrains $\boxed{C_d \geq 2\tau^{-1}(1+C)+1}$

- Otherwise, p is an input vertex. Let v be the vertex in whose cell p lies (that is, v is the nearest neighbour of p).

  - If v is an input vertex,

$$\text{lfs}(p) \leq NN_M(p)$$

Proof: This is immediate by definition of $\text{lfs}(p)$ and of v. ■

This case constrains $\boxed{C_0 \geq 1}$

  - Otherwise v is a Steiner point, and

$$\text{lfs}(p) \leq (1+\frac{C_d}{k})NN_M(p)$$

Proof: We know $\text{lfs}(v) \leq C_d NN_{M_v}(v)$ from the prior cases. Furthermore, we know that v did not yield to input vertex p, which means that $|vp| \geq kNN_{M_v}(v)$. Finally, we use the Lipschitz condition and some algebra to prove the result. ■

This case constrains $\boxed{C_0 \geq 1 + \frac{C_d}{k}}$

The only thing that remains to be proven is that C_0, and C_d are, in fact, strictly positive constants as claimed.

From $C_d \geq 2$ we get the constraint $C_0 \geq 1 + \frac{2}{k}$.

From $C_d \geq 2\tau^{-1}(1+C)+1$ then some algebra shows that $C_0 \geq \frac{(1+k)\tau+2}{k\tau-2}$. To have positive non-zero constants, this adds the additional constraint that $k\tau > 2$.

To finish the theorem, then, we say that for $k\tau > 2$, there exists a constant (namely C_0) that satisfies the condition $\mathrm{lfs}(p) \leq CNN(p)$.

7.2 Full algorithm

For the fully-general case, we need to add a small amount of complication. Instead of just two constants C_0 and C_d we need one for every dimension: C_i is the spacing for a Steiner vertex added as the result of splitting an i-dimensional facet.

The proof for C_d is identical to the point set proof, yielding $\boxed{C_d \geq 2}$ and $\boxed{C_d \geq 2\tau^{-1}(1+C)+1}$. The proof for C_0 is almost identical, except that now C_0 depends on C_1 rather than C_d. Thus we have: $\boxed{C_0 \geq 1 + \frac{C_1}{k}}$.

For C_i, we are considering inserting a Steiner point p which is the circumcenter of a ball B. The reason we want to insert p is that some point q we considered for insertion encroached on B. We use the following lemma:

Lemma 7.3 *Let q be a Voronoi node in a mesh M that encroaches on a lower-dimensional protective ball B. Then*

$$NN_M(q) \leq \frac{2}{k} r(B)$$

Proof: Let M_{-} be the lower-dimensional mesh of which B is a protective ball. The protective ball is defined by some vertices, including s. If $s \in M$, then clearly $NN_M(q) \leq |qs|$. Otherwise, $kNN_M(q) \leq |qs|$ or else q would have yielded to s. Meanwhile, we also know that $|qs| \leq 2r(B)$ since both are in or on the boundary of the ball. Substituting the prior bounds gives us the result. ∎

Intuitively, this lemma says that the algorithm never processes an encroachment on a protective ball until the protective ball is large with respect to the current feature size. Note that we can tighten the result for the case that B is in dimension $d = 1$: when q is encroaching on the diametral ball of a segment. The maximum distance from q to s then is just $\sqrt{2}r(B)$ rather than $2r(B)$. The lemma can be tightened for higher dimension also, but only by a few percent, unless we modify the algorithm.

Comparing to prior work, this is precisely the same bound as has been proven before, except that because of the vertex yielding rule we lose a factor $1/k$.

We can now state the main lemma of this section:

Lemma 7.4 *When the algorithm considers refining a mesh M of dimension i by inserting p, then:*

$$\mathrm{lfs}(p) \leq C_i NN_M(p)$$

Decision tree:

- If q is an input vertex, then $\text{lfs}(p) \leq NN_M(p)$. **Proof:** B is already encroached by another input vertex w, or else the algorithm wouldn't split B. We know that w is the nearest neighbour of p since w is in the mesh and only one vertex can encroach on B: $NN(p) = |pw|$. Let u be the closer of q and w; then $|pw| \geq |pu|$. Meanwhile, by the assumption that input features meet only at orthogonal angles, we know that u must lie on a different input feature than p, so $\text{lfs}(p) \leq |pu|$.

  In other words, this requires that $\boxed{C_i \geq 1}$ ■

- Otherwise, q is a Steiner point. **Proof:** Let M^+ be the mesh attempting to insert q. Let $j = \text{CD}(q) = \dim(M^+) > i$. We know we considered q for insertion in M, therefore $\text{lfs}(q) \leq C_j NN(q)$. Furthermore, we know $NN(p)$ is the radius of B, implying that since q encroached on B, $|pq| \leq NN(p)$. Finally, from Lemma 7.3, $NN_{M^+}(q) \leq \frac{2}{k} NN_M(p)$. Algebra:

$$\begin{aligned} \text{lfs}(p) &\leq \text{lfs}(q) + |pq| \\ \text{lfs}(p) &\leq C_j NN(q) + NN(p) \\ \text{lfs}(p) &\leq (\frac{2}{k} C_j + 1) NN(p) \end{aligned}$$

  Therefore, this requires that $C_i \geq \frac{2}{k} C_j + 1$ for all $j > i$; in particular, it requires $\boxed{C_i \geq \frac{2}{k} C_{i+1} + 1}$ for $i \geq 2$ and $\boxed{C_1 \geq \frac{\sqrt{2}}{k} C_2 + 1}$ ■

All in all, this means that we have:

$$\begin{aligned} C_0 &\geq 1 + k^{-1} C_1 \\ C_1 &\geq 1 + \frac{\sqrt{2}}{k} C_2 \\ C_i &\geq 1 + \frac{2}{k} C_{i+1} \\ C_d &\geq \max(2, 1 + 2\frac{1 + C_0}{\tau}) \end{aligned}$$

Solving for C_0 shows that the Spacing Theorem holds whenever $\boxed{\tau k^d > 2^{d-1/2}}$.

We now prove a spacing theorem that will be useful for later timing analysis. This theorem essentially states that the size of lower dimensional meshes does not shrink beyond a constant factor of the highest dimensional mesh. The proof is similar to the previous spacing results.

Lemma 7.5 *Suppose p is considered for insertion in M_i $(1 \leq i < d)$ at some time, then at that time there exists a constant α such that $\alpha\, \text{cfs}_i(p) > \text{cfs}_{i+1}(p)$.*

Proof: Suppose in the first case that p was considered as a split move (the center of a ball b) that was encroached by q. By Lemma 7.3, we have

$$NN_{M_{i+1}}(q) \leq \frac{2}{k} R(b) = \frac{2}{k} \underset{i}{\mathrm{cfs}}(p)$$

q was some Voronoi Node (or split point) considered for insertion in M_{i+1}, so in either case

$$NN_{M_{i+1}}(q) = \underset{i+1}{\mathrm{cfs}}(q)$$

Since p was the center of ball b, we also have $\mathrm{cfs}_i(p) = R(b)$. Using Lipshitz, we can combine this to obtain:

$$\begin{aligned}
\underset{i+1}{\mathrm{cfs}}(p) &\leq \underset{i+1}{\mathrm{cfs}}(q) + |pq| \\
&= NN_{M_{i+1}}(q) + R(b) \\
&\leq (\frac{2}{k} + 1) R(b) = (\frac{2}{k} + 1) \underset{i}{\mathrm{cfs}}(p)
\end{aligned}$$

So as long as there exist $\boxed{\alpha \geq 1 + \frac{2}{k}}$, then the lemma will be satisfied in this case.

Suppose in the second case that p is a Voronoi node of a skinny Voronoi cell belonging to q.

As in previous lemmas, we can apply the hypothesis inductively on q or its nearest neighbor of M_i, and we find that $(1 + \alpha) \mathrm{cfs}_i(q) \geq \mathrm{cfs}_{i+1}(q)$. We have:

$$\underset{i}{\mathrm{cfs}}(p) = |pq| = R_q \geq \tau r_q = \frac{\tau}{2} \underset{i}{\mathrm{cfs}}(q) \geq \frac{\tau}{2(1+\alpha)} \underset{i+1}{\mathrm{cfs}}(q)$$

Then:

$$\begin{aligned}
\alpha \underset{i}{\mathrm{cfs}}(p) &\geq \frac{\tau}{2} \underset{i+1}{\mathrm{cfs}}(q) - \underset{i}{\mathrm{cfs}}(p) \\
&\geq \frac{\tau}{2} (\underset{i+1}{\mathrm{cfs}}(p) - |pq|) - \underset{i}{\mathrm{cfs}}(p) \\
&= \frac{\tau}{2} \underset{i+1}{\mathrm{cfs}}(p) - (1 + \frac{\tau}{2}) \underset{i}{\mathrm{cfs}}(p)
\end{aligned}$$

Thus:

$$\frac{2}{\tau} (\alpha + 1 + \frac{\tau}{2}) \underset{i}{\mathrm{cfs}}(p) \geq \underset{i+1}{\mathrm{cfs}}(p)$$

Or simply:

$$(\frac{2\alpha}{\tau} + \frac{2}{\tau} + 1) \underset{i}{\mathrm{cfs}}(p) \geq \underset{i+1}{\mathrm{cfs}}(p)$$

■ So we must find an α such that $\alpha \geq \frac{2\alpha}{\tau} + \frac{2}{\tau} + 1$, but this simply requires that $\boxed{\alpha \geq (\tau + 2)/(\tau - 2)}$

Clearly, we can always take an α so large when $\tau > 2$.

Combining Lemma 7.5 over all dimensions, we can see that there exist a constant (exponential in dimension) so that $\mathrm{cfs}_M(p) \in \Omega(\mathrm{cfs}_{M_d}(p))$ for any feature mesh M and any point $p \in M$. In particular, this allows us to lower bound the size of any protective balls which are queued.

Lemma 7.6 *There exists a constant β, such that for any protective ball b with center p in mesh M: from a lower dimensional mesh,*

$$R(b) \geq \beta \underset{d}{\mathrm{cfs}}(p)$$

The lemma follows trivially, since $R(b) = \mathrm{cfs}_M(p) \in \Omega(\mathrm{cfs}_d(p))$ by earlier arguments.

8 Quality maintenance

The key property on which all our bounds depend is that at every stage of the algorithm, the mesh is always a quality mesh. This determines our runtime and space bounds. The quality is not the τ quality bound that the user calls for – we cannot guarantee that the quality won't degrade by some amount – but is only smaller by a constant function of parameters τ and k, and of the dimension d.

To prove the result we first need some technical lemmas:

Lemma 8.1 (A Break Move Only Marginally Decreases *NN*) *For any vertex u in a mesh M, executing a break move to obtain M' decreases the nearest neighbor of u by at most a factor of $(1-k)/2$, that is:*

$$NN_{M'}(u) \geq \frac{1-k}{2} NN_M(u)$$

Proof:

Suppose we are breaking the Voronoi cell of p. Suppose its farthest Voronoi vertex is v at radius R_p.

There are two cases, either we insert v, or we yield to a point v'. For any $u \in M$, $NN_M(u) = NN_{M'}(u)$ unless v (or v') is the new nearest neighbor of u.

Case 1: We inserted v. If v is the new nearest neighbor of u, then the nearest neighbor of u is at most cut in half:

$$NN_M(u) \leq |up| \leq |uv| + R_p \leq 2|uv| \leq 2NN_{M'}(u)$$

Case 2: We yielded to v'. If v' is now the nearest neighbor of u, then the argument is the same with a worst constant. First we have:

$$|uv| \leq \frac{1}{1-k}|uv'|$$

From which it follows:

$$NN_M(u) \leq |up| \leq |uv| + |vp| \leq 2|uv| \leq \frac{2}{(1-k)}|uv'| = \frac{2}{(1-k)} NN_{M'}(u)$$

■

Lemma 8.2 (Cleaning Preserves Current Feature Size) *If we begin with a mesh M', and we execute a series of moves consisting only of clean moves to obtain M'', then for every vertex $v \in M''$, we have:*

$$\operatorname*{cfs}_{M'}(v) \leq c_{8.2} NN_{M''}(v)$$

Proof:

We first claim that at the time of insertion of any vertex v,

$$\operatorname*{cfs}_{M'}(v) \leq (c_{8.2} - 1) NN_{M''_v}(v)$$

This fact has the obvious corollary that for any mesh M'',

$$\operatorname*{cfs}_{M'}(v) \leq (c_{8.2}) NN_{M''}(v)$$

(the proof is identical to Corollary 7.2)

We proceed then to prove the fact as above. Suppose we wish to break some skinny Voronoi cell of p. There are two cases, either we insert the center v or we yield to some input point v'.

Suppose first we insert the center, v. Then we have by induction:

$$\begin{aligned} \operatorname*{cfs}_{M'}(v) \leq |vp| + \operatorname*{cfs}_{M'}(p) &\leq R_p + c_{8.2} NN_{M''}(p) \\ &\leq R_p + 2c_{8.2} r_p \end{aligned}$$

Since p was skinny:

$$\leq R_p \frac{2c_{8.2}}{\tau} = \leq (1 + \frac{2c_{8.2}}{\tau}) R_p = (1 + \frac{2c_{8.2}}{\tau}) NN_{M'}(v)$$

The second case is when we yield to some input point v'.
Proof is similar:

$$\begin{aligned} \operatorname*{cfs}_{M'}(v') \leq |v'p| + \operatorname*{cfs}_{M'}(p) &\leq (1+k) R_p + c_{8.2} NN_{M''}(p) \\ &\leq (1+k) R_p + 2c_{8.2} r_p \\ &\leq (1 + k + \frac{2c_{8.2}}{\tau} R_p \leq (1 + k + \frac{2c_{8.2}}{\tau} \frac{1}{1-k} NN_{M''}(v') \end{aligned}$$

It can be seen that if we set $c_{8.2} > (1+k)\tau/(\tau(1-k)-2)$ then the induction hypotheses hold properly. Note here that we now require $\boxed{\tau \cdot (1-k) > 2}$. ■

We will now show Lemma 8.3, which roughly states that the quality of the mesh is only marginally decreased by a single break move. This lemma is fairly intuitive since a break move inserts one vertex that is relatively far from its neighbors, and so it does not significantly degrade the mesh quality.

Lemma 8.3 (Break Quality) *There exists τ' depending only on τ and k such that M' is a τ'-quality mesh.*

Proof: Suppose v is a vertex inserted by a break move into M to obtain to M'.

We note that for vertices of a mesh, $\mathrm{cfs}_M(u) = NN_M(u)$, so Γ_M can be written with NN_M in the denominator for vertices of M.

For every point x in M', clearly $G_{M'}(x) \leq G_M(x)$, since refinement can only decrease G. And by Lemma 8.1, for every vertex $u \neq v$ in M', $NN_{M'}(u) \geq \frac{1-k}{2} NN_M(u)$.

Consider also the new vertex v. We have that $\mathrm{cfs}_M(v) \leq 2R_p$, where p was Voronoi site that was broken to create M'. We also have that $NN_{M'}(v) \geq (1-k)R_p$, from which it follows that $NN_{M'}(v) \geq \frac{1-k}{2}\,\mathrm{cfs}_M(v)$.

Since M was τ-quality, by Lemma 6.5 we then obtain that for every vertex $u \in M'$:

$$\Gamma'_M(u) \leq \frac{2}{(1-k)}\Gamma_M(u) \leq \frac{2c_{6.5}(\tau)}{(1-k)}$$

Since $\Gamma_{M'}$ is bounded at every vertex, it then follows from Lemma 6.6 that M' is a τ'-quality mesh where $\tau' = 4c_{6.5}(\tau)/(1-k)$. ■

We now show Lemma 8.4, which roughly that the cleaning process is approximately monotone with respect to quality. While cleaning to improve the mesh quality from τ' back to τ, we will never reach an intermediate stage where the quality of the mesh is more than marginally worse.

Lemma 8.4 (Intermediate Quality) *There exists τ'' depending only on τ and k such that every M'' is a τ''-quality mesh.*

Proof: Assume that we have M'', a τ''-quality mesh obtained in the process of cleaning M'. First note as before that for every point x, $G_{M''}(x) \leq G_{M'}(x)$.
Note that by Lemma 8.2, we have that for every vertex v,

$$NN_{M''}(v) \geq c_{8.2} \operatorname*{cfs}_{M'}(v)$$

And since M' was τ'-quality, we can combine with Lemma 6.5 to obtain that:

$$\Gamma_{M''}(v) \leq \frac{1}{c_{8.2}}\Gamma_{M'}(v) \leq \frac{c_{6.5}(\tau')}{c_{8.2}}$$

Since $\Gamma_{M''}$ is bounded at every vertex, it then follows from Lemma 6.6 that M'' is a τ''-quality mesh where $\tau'' = 2c_{6.5}(\tau')/c_{8.2}$. ■

Corollary 8.5 is the obvious corollary following from Lemmas 8.3 and 8.4.

Theorem 8.5 (Always Quality) *At any point during Sparse Voronoi Refinement, the intermediate mesh is a τ''-quality mesh.*

Corollary 8.6 (Sparse Mesh) *Any intermediate mesh during the lifetime of Sparse Voronoi Refinement is sparse, i.e. there is a constant depending only on τ and k that bounds the degree of every vertex.*

Proof: This follows directly from Corollary 8.5 and Theorem 3.15 in [MTTW95]. A Delaunay mesh with bounded constant radius-edge ratio is of bounded constant degree depending only on the quality (τ''). ■

9 Runtime analysis

When we look at the algorithm, the astute reader will intuit that there are two kinds of operations occurring. One kind is trivial to bound to run in $O(1)$ time thanks to the sparsity result of the previous section. The other kind is non-trivial: how big is BALLS_v in FORCEINSERT, for instance?

To handle this, we will amortize the cost of handling a protective ball over the lifetime of the ball. We can think about Point Location Events (PLEs), as a pair consisting of a SPLIT or FORCEINSERT along with a protective ball being affected. We will claim that the total runtime is just the number of PLEs. We will bound this number of events with an amortized argument, claiming a bound on the number of PLEs per protective ball.

The main tool of this section will be to show that in some sense, PLEs are packed around a protective ball. Whenever we have a PLE, the FORCEINSERT occuring is largely disjoint from other possible events, relative to the distance at which a protective ball is affected.

We now formalize these notions, including later definitions for a PLE, in a rigorous proof.

In all the cases, we will be considering some PLE involving the FORCEINSERT of a point q' affecting a protective ball with center p, and the goal will be to bound the proximity of p and q' through intermediaries. We will frequently refer to the scenarios as depicted in Figure 14 and Figure 15.

9.1 Currency exchange lemmas:

Lemma 9.1 *In the* SPLIT *scenario, $NN(q) \leq O(NN(q'))$.*

Proof: We know that $|qq'| \leq kNN(q)$. Therefore, there is an annulus of width $(1-k)NN(q)$ separating q' from a region that may have points in it: $NN(q') \geq (1-k)NN(q)$. ■

Lemma 9.2 *In the* SPLIT *case, $R(v) \leq O(NN(q))$.*

Proof: Consider a point $x \in B(q, NN(q)) \cap V(v)$. Such a point exists because we know we SPLIT only searches Voronoi cells that intersect the circumball centered at q.

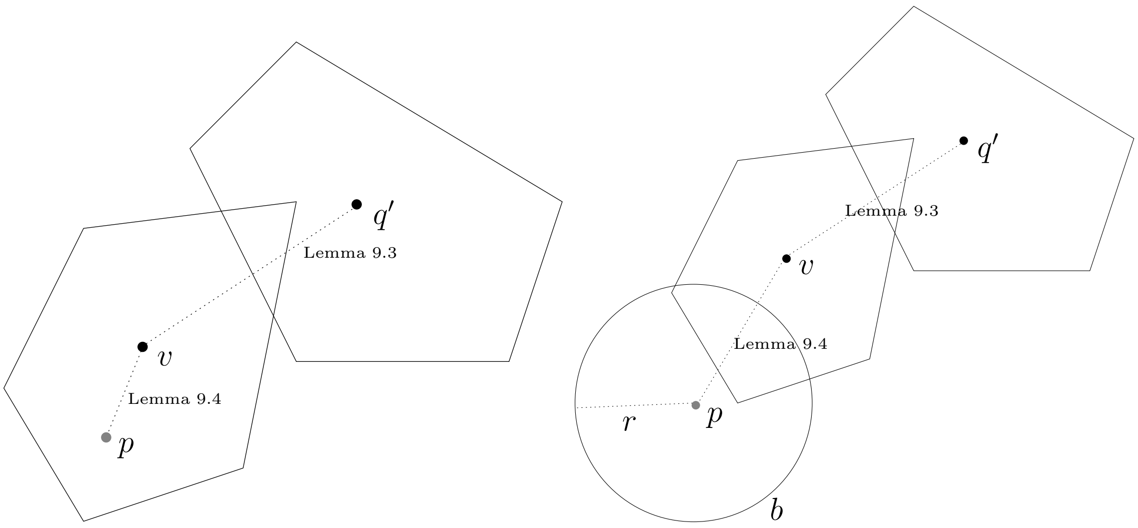

Figure 14: The FORCEINSERT Scenario. *Left:* The Voronoi cell of v is modified by the FORCEINSERT of q', and so every point p in BALLS(v) must be checked for relocation. *Right:* When we consider the presence of subfacets, any protective ball b (center p) in BALLS(v) must be checked for relocation. In either case, we will claim that p and q' are not relatively too far apart, so we will charge the FORCEINSERT PLEto their proximity.

By virtue of x being within $V(v)$, the current feature size $\mathrm{cfs}(x)$ must be at least $r(v)$, since at the least, x is equidistant from v and a second point, and the closest equidistant point is by definition exactly $r(v)$. Furthermore, $r(v) \geq R(v)/\tau$ because the cell is of bounded aspect ratio.

Meanwhile, by virtue of x being within $B(q, NN(q))$, where q is a circumcenter, $\mathrm{cfs}(x)$ is no further than twice the radius $NN(q)$ from at least $d+1$ points.

Putting these together yields the result: $R(v) \leq \tau\,\mathrm{cfs}(x) \leq 2\tau NN(q)$. ■

Lemma 9.3 *In the* FORCEINSERT *scenario, let M be the mesh before q' is inserted, and let M' be the mesh after q' is inserted. Then: $R'_M(q) \leq O(NN_M(q'))$.*

Proof: Because $V_{M'}(q')$ is a quality cell, $R_{M'}(q') \leq \tau r_{M'}(q') = \frac{\tau}{2} NN_{M'}(q')$. But q' is the only difference in spacing between M and M', so $NN_{M'}(q') = NN_M(q')$.

Thus $R_{M'}(q') \leq \frac{\tau}{2} NN_M(q')$ ■

Lemma 9.4 *In the* FORCEINSERT *scenario, let M be the mesh before q' is inserted, and let M' be the mesh after q' is inserted. Then: $R_M(v) \leq O(NN_M(q'))$.*

Proof: Consider a point $x \in V_{M'}(q') \cap V_M(v)$.

As before, we know that $\mathrm{cfs}_M(x) \geq r_M(v) \geq R_M(v)/\tau$.

By the Lipschitz condition, $\mathrm{cfs}_M(x) \leq \mathrm{cfs}_M(q') + |xq'|$. The farthest Voronoi node from q' in M' (after q' is inserted) is at distance $R_{M'}(q')$. A Voronoi node has $d+1$ equidistant

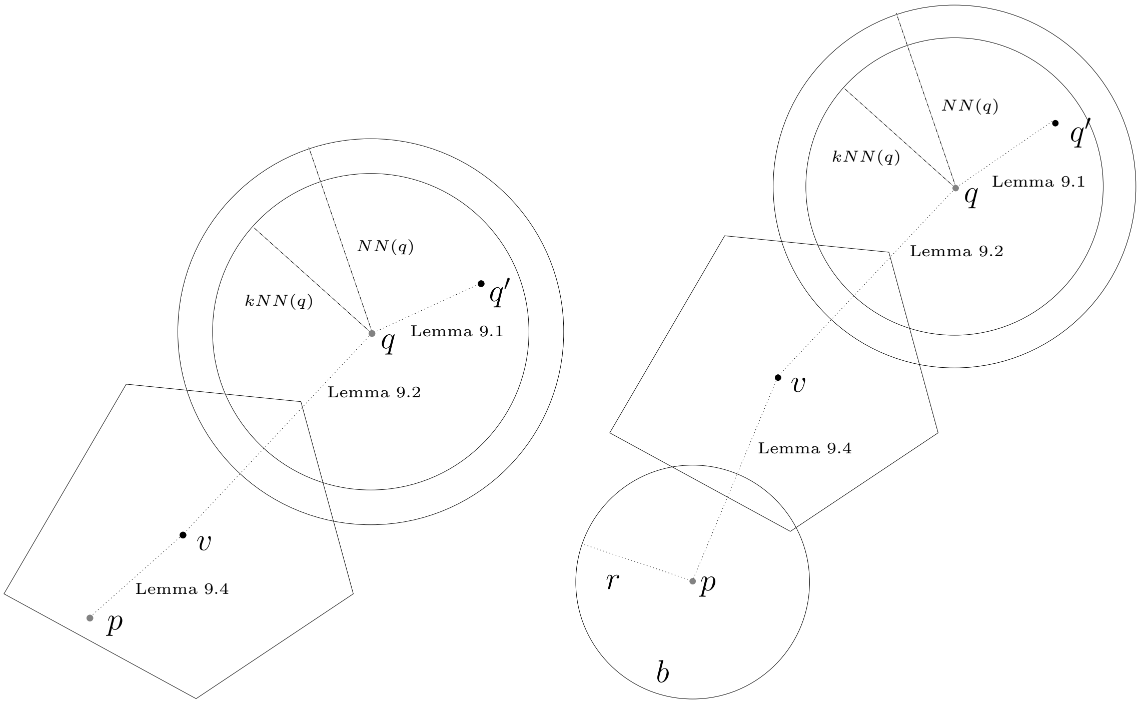

Figure 15: The SPLIT Scenario. *Left:* When performing a SPLIT of q, we query BALLS(v) to see if it contains any point p to which we could yield. Eventually, we FORCEINSERT some q'. (Perhaps q is q', or perhaps we yielded to some q'). We claim that p and q' are not relatively too far apart, so we will charge the SPLIT PLE to their proximity. *Right:* In the case where subfacets are present, p is the center of some ball b who is in BALLS(v).

mesh vertices; only one of them is q', so $\text{cfs}_M(q') \leq 2R_{M'}(q')$. More simply, $|xq'| \leq R_{M'}(q')$, leading to $\text{cfs}_M(x) \leq 3R_{M'}(q')$. Finally, Lemma 9.3 leaves us with: $R_M(v) \leq \frac{3\tau^2}{2} NN_M(q')$. ■

Lemma 9.5 *In both* SPLIT *and* FORCEINSERT *scenarios,* $r \leq O(R(v))$.

Proof: Consider a point $x \in B(p, r) \cap V(v)$.

By virtue of x being within B, we know from Lemma 6.1 that $\text{cfs}(x) \geq c_{6.1}\,\text{cfs}(p)$. Furthermore, B is singly-encroached, so $\text{cfs}(p) \geq r$.

Meanwhile, by virtue of x being within $V(v)$, we know that $\text{cfs}(x) \leq 2R(v)$.

Putting these together yields the result: $r \leq \text{cfs}(p) \leq \text{cfs}(x)/c_{6.1} \leq \frac{2}{c_{6.1}} R(v)$ ■

Lemma 9.6 *In the* SPLIT *case,* $|pq'| \leq O(NN(q'))$.

Proof: Consider $x \in B(p, r) \cap V(v)$ and $x' \in B(q, NN(q)) \cap V(v)$.

By the triangle inequality, $|pq'| \leq |px| + |xv| + |vx'| + |x'q| + |qq'|$. We can fill each term in trivially: $|px| \leq r$ and $|xv| \leq R(v)$ by the definition of x. $|vx'| \leq R(v)$ and $|x'q| \leq NN(q)$ by the definition of x'. Finally, $|qq'| \leq kNN(q)$ by construction.

Computing the sum and converting (via the currency-conversion lemmas) to units of $NN(q')$ yields the result:

$$|pq'| \leq \frac{4\tau(1 + \frac{1}{c_{6.1}}) + (1 + k)}{1 - k} NN(q')$$

■

Lemma 9.7 *In the* FORCEINSERT *case,* $|pq'| \leq O(NN(q'))$.

Proof: Consider $x \in B(p, r) \cap V_M(v)$ and $x' \in V_{M'}(q') \cap V_M(v)$.

By the triangle inequality, $|pq'| \leq |px| + |xv| + |vx'| + |x'q'|$. As in the SPLIT case, $|px| \leq r$ and $|xv| \leq R(v)$ follow by the definition of x; similarly, $|vx'| \leq R(v)$ and $|x'q'| \leq R_{M'}(q')$. We can convert these using the currency-conversion lemmas to reach the conclusion $|pq'| \leq (3\tau^2(1 + \frac{1}{c_{6.1}}) + \tau/2) NN(q')$. ■

Definition 9.8 *Consider an event in an i-dimensional mesh when the algorithm needs to do $O(1)$ work and charges it to a subfacet $B = B(p, r)$; subsequently, point q' is inserted.*

Such an event is dubbed a Packed Point Location Event *if $|pq'| \leq NN(q')$. We tag the event with the dimension of the mesh that caused it, and with the (abbreviated) name of the function, either S for* SPLIT*, or F for* FORCEINSERT*.*

For instance, $\text{PLE}^i_S(B, q')$ means a SPLIT into an i-dimensional mesh caused a PLE on the protective ball B, and subsequently actually inserted q'.

Lemma 9.9 (Annulus Proof) *Consider a point p in space. At most $O(1)$* PLE*s occur around p that insert points in an annulus $A(p, l, 2l) = B(p, 2l) - B(p, l)$ for any l.*

Proof: Consider any two PLEs a and b. The newer one has an empty ball of radius $NN(a) \in \Omega(|pa|) = \Omega(l)$ by the definition of a PLE. Therefore, $|ab| \in \Omega(l)$. Because this is true of every pair of PLEs in the annulus around p, every PLE has an ball of radius $\Omega(l)$ around it that is empty of any other PLE. Obviously, only a constant number of such empty balls can fit into the annulus. ■

Lemma 9.9 and the definition of a PLE together directly imply the following:

Corollary 9.10 *Consider a protective ball $b = B(p, r)$. Let $\mathrm{cfs}_0(p)$ be the feature size at p when the ball was created (either in the call to* ForceInsert*, or at initialization). Let $\mathrm{cfs}_1(p)$ be the feature size at p when the ball is last affected (either by being destroyed in* ForceInsert*, or because no further vertices are inserted nearby). Then at most $O(\lg(\mathrm{cfs}_0(p)/\mathrm{cfs}_1()))$* PLE*s of any type can occur that affect b.*

9.2 Full cost accounting

Not all the cost of Split and ForceInsert can directly be charged to PLEs. Here, we account for all costs.

Lemma 9.11 *All but $O(1)$ of the work of* ForceInsert *can be charged to* PLE*s.*

Proof: The costs of ForceInsert are:

- Inserting the point and updating the mesh. The cavities C and C' have size $O(1)$ thanks to Lemma 6.8. Thus, this takes $O(1)$ time.
- Gathering all the protective balls and reassigning them. Each ball b' in S incurs $O(1)$ work as we iterate over S a constant number of times to reassign all the lower-dimensional protective balls. Furthermore, we know from Lemma 9.7 that the geometry of the situation matches the requirement for calling this a PLE on b'.
- Marking skinny cells. C' has constant size, so this takes only $O(1)$ work.
- Calls to ReplaceBalls. Again, C and C' have constant size; furthermore, there are only d dimensions.

■

Lemma 9.12 *All but $O(1)$ of the work of each invocation of* Split *can be charged to* PLE*s.*

Proof: A similar analysis to the above almost works. There remains one complication: making sure that the recursive invocation in line 15 happens only $O(1)$ times before we finally insert a point. This is required to prove that each ball $b \in S$ is only queried $O(1)$ times per insertion, as is needed for the definition of a PLE.

By Lemma 7.6, every protective ball we might yield to has size $\Omega(r)$. If we split a ball, then the new balls it creates necessarily have smaller radius. If we needed to split balls $\omega(1)$ times in a row, then the new ball would have been split to size $o(r)$, a contradiction. Therefore, every ball inspected in Split is inspected at most $O(1)$ times before a point is finally inserted. ■

Theorem 9.13 *Total algorithm runtime charges sum to $O(n \lg L/s + m)$.*

Proof: Every time we insert a vertex, we pay for $O(1)$ overhead in SPLIT and FORCEINSERT as we have just proved.

The PLE charges we account for as follows. For input protective balls (and input points), the smallest PLE has radius $\Omega(s)$ thanks to the spacing proof (Lemma 7.1). Meanwhile, no points are inserted outside the mesh, so the largest PLE has radius $O(L)$. Thus each protective ball present at initialization time will have associated with it $O(\lg L/s)$ PLEs. By definition, there are n initial balls, leading to the $O(n \lg L/s)$ term.

The remainder of the protective balls created during the algorithm were necessarily created by FORCEINSERT. Any such ball b has $\mathrm{cfs}_0(b) \in O(\mathrm{cfs}_1(b))$, so PLE charges are $O(1)$ each. Every such ball is created by a FORCEINSERT which adds a point to the output (m total), and each FORCEINSERT creates only $O(1)$ new balls according to Theorem 8.6. Thus the total PLE charges on non-input protective balls are $O(m)$. ■

10 Conclusions

We have shown how to produce, in near-optimal time, a conformal mesh of the input domain, in arbitrary dimension. While this is a first, there are many remaining questions.

Firstly, we allow the user to demand a value of $\tau \geq 2^{d-1/2}$, equivalent to a radius-edge ratio of $\rho \geq 2^{d-3/2}$. With some proof work (without changing the algorithm at all), we know how to improve this slightly in $d > 2$, but not substantially. In two dimensions, our bound matches Ruppert's original bound of 20.7°, or $\rho \geq 2\sqrt{2}$ – and indeed it should, since our proof is based on the same precepts. In the decade since the publication of Ruppert's result, his proof has been improved to allow the user to demand angles of more than 25°; and it is believed that the correct answer is that the user should be able to demand almost 30° on any input (of course, on some inputs, the user can demand even larger angles). It is less clear how the bound changes according to dimension, but we believe our stated bound is much too conservative.

Most egregiously, we have demanded that the user must give us a PLC with all angles orthogonal or obtuse. Several recent papers have begun addressing the issue of removing the 90° angle restriction [CP03, PW04]. These techniques seek to create, at initialization, a protective region around small angles, requiring an oracle that will provide the algorithm with the lfs at many points. This requirement leads to very poor runtime bounds with any naive analysis. To incorporate these methods into SVR, the natural method would be to create some protective region that could adapt as SVR recovers more input features. Indeed, the recent work of Pav and Walkington [PW04] appears to be moving in this direction, as they attempt to reduce the oracular requirements. We believe that as algorithms for three dimensional meshing with arbitrary domains continue to mature, they can be incorporated into SVR to achieve better runtime guarantees.

Certainly, future work will include the parallelization of SVR, which is important for any modern large-scale meshing algorithm. All of the mesh modifications in SVR are local, so basic shared memory algorithmic techniques involving a conflict graph will most likely suffice. This analysis is in progress, and we do not foresee any major difficulties.

The last possibility is the reduction of the point location costs from $O(n \log(L/s))$ down to $O(n \log n)$ for inputs with pathological spread. This is mainly of theoretical concern. One possibility would be to cluster together several small input features and point locate them as a conglomerate, until the mesh is refined down to a relatively polynomial spread. It is quite unclear how to properly define such a clustering strategy, but vague intuitions suggest that something akin to well-separated pair decompositions [CK92] might suffice.

References

[Bar02] Jernej Barbic. Quadratic example for delaunay refinement. Private Communication, 2002.

[BEG90] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. In *31st Annual Symposium on Foundations of Computer Science*, pages 231–241. IEEE Computer Society Press, 1990.

[BEG94] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[BET99] Marshall W. Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, 9(6):517–532, 1999.

[CD03] Siu-Wing Cheng and Tamal K. Dey. Quality meshing with weighted delaunay refinement. *SIAM J. Comput.*, 33(1):69–93, 2003.

[Che89] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[Che97] L. Paul Chew. Guaranteed-Quality Delaunay Meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393, Nice, France, June 1997. Association for Computing Machinery.

[CK92] P. Callahan and S. Kosaraju. A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields, 1992.

[CP03] Siu-Wing Cheng and Sheung-Hung Poon. Graded Conforming Delaunay Tetrahedralization with Bounded Radius-Edge Ratio. In *Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms*, pages 295–304, Baltimore, Maryland, January 2003. Society for Industrial and Applied Mathematics.

[ELM+00] Herbert Edelsbrunner, Xiang-Yang Li, Gary L. Miller, Andreas Stathopoulos, Dafna Talmor, Shang-Hua Teng, Alper Üngör, and Noel Walkington. Smoothing and cleaning up slivers. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, pages 273–277, Portland, Oregon, 2000.

[Eri01] Jeff Erickson. Nice point sets can have nasty delaunay triangulations. In *Symposium on Computational Geometry*, pages 96–105, 2001.

[HMP06] Benoît Hudson, Gary Miller, and Todd Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, Birmingham, Alabama, 2006.

[HPU05] Sariel Har-Peled and Alper Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. In *Symposium on Computational Geometry*, 2005.

[Li03] Xiang-Yang Li. Generating well-shaped d-dimensional Delaunay meshes. *Theor. Comput. Sci.*, 296(1):145–165, 2003.

[LT01] Xiang-Yang Li and Shang-Hua Teng. Generating well-shaped Delaunay meshed in 3D. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–37. ACM Press, 2001.

[Mil04] Gary L. Miller. A time-efficient Delaunay refinement algorithm. In *Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 400–409, New Orleans, 2004.

[Mit93] Scott A. Mitchell. Refining a Triangulation of a Planar Straight-Line Graph to Eliminate Large Angles. In *34th Annual Symposium on Foundations of Computer Science*, pages 583–591. IEEE Computer Society Press, 1993.

[MPW02] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. Fully Incremental 3D Delaunay Refinement Mesh Generation. In *Eleventh International Meshing Roundtable*, pages 75–86, Ithaca, New York, September 2002. Sandia National Laboratories.

[MTT+96] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. Control Volume Meshes Using Sphere Packing: Generation, Refinement and Coarsening. In *Fifth International Meshing Roundtable*, pages 47–61, Pittsburgh, Pennsylvania, October 1996.

[MTT99] Gary L. Miller, Dafna Talmor, and Shang-Hua Teng. Optimal coarsening of unstructured meshes. *Journal of Algorithms*, 31(1):29–65, Apr 1999.

[MTTW95] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 683–692, Las Vegas, May 1995. ACM.

[MTTW99] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. On the radius–edge condition in the control volume method. *SIAM J. Numer. Anal.*, 36(6):1690–1708, 1999.

[MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370 (electronic), 2000.

[PW04] Steven E. Pav and Noel J. Walkington. Robust Three Dimensional Delaunay Refinement. In *Thirteenth International Meshing Roundtable*, pages 145–156, Williamsburg, Virginia, September 2004. Sandia National Laboratories.

[Rup95] Jim 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).

[She98] Jonathan Richard Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota)*, pages 86–95. ACM, June 1998.

[She02] Jonathan Richard Shewchuk. Constrained delaunay tetrahedralizations and provably good boundary recovery. In *Eleventh International Meshing Roundtable*, pages 193–204, Ithaca, New York, September 2002. Sandia National Laboratories.

[STÜ02] Daniel Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings, 11th International Meshing Roundtable*, pages 205–218. Sandia National Laboratories, September 15-18 2002. http://www.arxiv.org/abs/cs.CG/0207063.

[Tal97] Dafna Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997. CMU CS Tech Report CMU-CS-97-164.