Sparse Voronoi Refinement [*]

Benoît Hudson, Gary Miller, and Todd Phillips

Computer Science Department
Carnegie Mellon University

Summary. 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.

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 quadtrees 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.

Size Guarantee: We would like to say that our algorithm generates a mesh whose size is at most a constant fraction larger than the smallest radius-edge quality mesh. There are two problems with this goal. First, finding such a mesh seems difficult. Two, it is not obvious that this is the correct goal for meshing applications. To see the second concern consider an input of just two edges of the same length forming a cross. Assume the edges do not intersect but come arbitrarily close. An optimal radius-edge mesh would simply form a sliver out of these two edges. While, a good aspect-ratio mesh would introduce very small tetrahedra near where the two edges are close. Thus, it seems that we would like to return a mesh such that the local feature size at every point is bound both from above and below by a constant times the diameter of the simplex containing it. In this paper we will only show that our algorithm will return a mesh such that no tetrahedra is too small. That the distance between any two vertices is bounded below by the local feature size of the input.

Time and Space Usage: Given a Piecewise Linear Complex (PLC) as input [MTTW95], 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 ([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 $O(m)$.

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. 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 [MTTW95, 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 8.1), plus bookkeeping of $O(n \log(L/s))$ term arising from point location costs (Section 8.2).

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 9.

Section 2 discusses some related work, along with some of the design decisions of SVR.

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

Section 4 presents 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.

Section 5 presents 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]. For brevity we refer to earlier results for most of the proofs and only highlight the

novel portions; a technical report version of this paper presents the proofs in full detail [HMP06].

The results of Section 6 in particular provide fairly strong structural results about quality Voronoi diagrams that may be applicable to other meshing algorithm analyses; Sections 7 and 8 use the structural results to prove our algorithm is correct, size-guaranteed, and fast.

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 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.

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**. 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.

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. Chew added the circumcenter of poor quality triangles(Steiner nodes).

Ruppert [Rup95] extended this idea of adding circum-centers 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.

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. Har-Peled and Ungor removed the $\lg m$ factor using a quadtree data structure [HPU05].

Many algorithms for Delaunay refinement in 3D have been proposed [She02, CP03, PW04, CD03], but so far have eluded nontrivial runtime analysis. 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.

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 *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 [MTTW95].

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 9 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 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 will be addressed in later work 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 x 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 $CD(p)$.

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 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. *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 $G_M(x) \equiv G_{\mathrm{vertices}(M)}(x)$.

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

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

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].

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

SIMPLIFIED-SVR(P: d-dim point set, τ: mesh quality constant, k: $0 < k \leq 1$)
 1: Assume an initial Voronoi mesh M exists.
 2: Let U be the set of uninserted input vertices: $U = P - V(M)$
 3: **while** U is non-empty **do**
 4:    Perform a *break* move
 5:    **while** M has aspect ratio worse than τ **do**
 6:       Perform a *clean*
 7:    **end while**
 8: **end while**

TRYINSERT(p: d-dim point, M d-dim mesh, U: set of uninserted d-dim points)
 9: **if** $\exists u \in U$ s.t. $|pu| \leq kNN_M(p)$ **then**
10:    *Yield*: Insert u into M, updating U and the Voronoi diagram.
11: **else**
12:    Insert p into M, updating the Voronoi diagram.
13: **end if**

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

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 one of its farthest Voronoi nodes, 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 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.
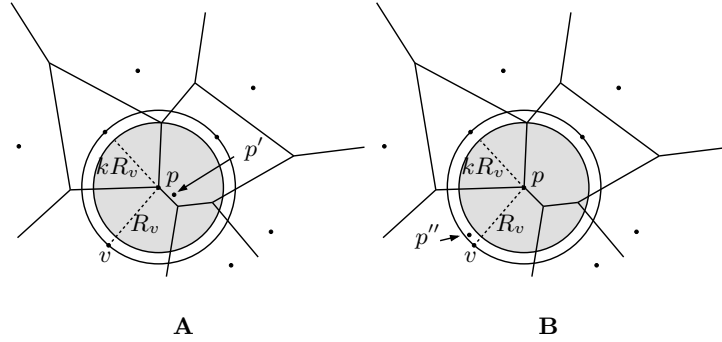
Fig. 2. The cell of v is being destroyed by SVR (either because it is skinny, or because p'' is inside); SVR tries to add the farthest Voronoi node p. The outer circle around p is of radius R_v; the inner disc, in grey, is of radius kR_v. **(A)** Because the uninserted input point p' is near p, SVR must 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.

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, $k\tau > 2\sqrt{2}$), 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 [MTTW95, 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 $\log(L/s)$. Precise statements of the lemmas are found in Section 8.2, while rigorous proofs appear in the tech report [HMP06].

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

TryAdd(p: i-dim Voronoi point, r: radius, M: i-dim mesh)
1: **if** $\exists q \in U(M)$ such that $|pq| \leq kr$ **then**
2:     ForceAdd(q,M)
3: **else if** $\exists B \in Balls(M)$ such that $p \in B$ **then**
4:     Split(B) for all such B
5:     TryAdd(p, r, M)
6: **else**
7:     ForceAdd(p, M)
8: **end if**

Split(B: protecting ball from a mesh M)
1: TryAdd(center(B), radius(B), M)
2: **while** M has a skinny cell $V(v)$ **do**
3:     TryAdd(far-node(v), $R_M(v)$, M)
4: **end while**

ForceAdd(p: i-dim point,M: i-dim mesh)
1: Run Bowyer-Watson or Edelsbrunner-Shah to insert p into the mesh M.
2: Reassign vertices in $U(M)$ and protecting balls in $Balls(M)$ to their new Voronoi cells as needed.
3: **for** every higher-dimensional mesh M^+ that contains M as a sub-feature **do**
4:     Remove from $Balls(M^+)$ all the balls associated with Voronoi nodes that were destroyed.
5:     Add to $Balls(M^+)$ a ball for every Voronoi node that was created.
6:     Add p to $U(M^+)$ if not already present.
7: **end for**
8: {The following only occurs if p was a vertex in $P(M)$.}
9: If $cd(p) < i$ then remove p from $U(M)$
10: **while** $\exists B \in F(M)$ such that $p \in B$ **do**
11:     If p is the second encroachment on B, Split(B)
12: **end while**

Fig. 3. The subroutines used in the SVR algorithm.

The input is described as a PLC, and must meet the criteria described in Section 3.

As scratch data structures, we maintain, *for every input feature* including the complete domain:

- A mesh M. We will prove that each mesh always has good quality (good Voronoi aspect ratio).
- A mapping U from each Voronoi cell in M to the list of uninserted vertices in that cell of containment dimension $< i$.
- A mapping $Balls$ from each Voronoi cell in M to the list of balls intersect the cell(these balls are used to protect lower-dimensional features).

A protecting ball is a ball $B(p, NN_{M^-}(p))$ centered at a Voronoi node of a lower-dimensional mesh M^-. This corresponds exactly to the circumball of a lower-dimensional subfacet.

In the bootstrapping phase of the algorithm, we initialize the data structures for each feature in order of increasing dimension. We generate the Voronoi diagram of the convex hull of each feature F, then partition each sub-feature's protecting balls among the cells of $M(F)$ to create the S mapping. We also partition the vertex set similarly.

As in the point-set case, the algorithm will now iteratively perform *break* and *clean* moves until it reaches a quality conformal mesh. The *clean* move is identical to before (except for an updated version of TryInsert). The *break* move is now expanded to try to break encroached balls using the new Split routine. Encroached balls correspond to lower-dimensional input features that has not yet been recovered.

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.

When we consider adding a Steiner point, as before, we may have to yield to an input vertex. However, we may also have to yield to lower-dimensional input features; this is a common technique in mesh refinement. 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 that have containment dimension less than $\dim(M)$. The need for this is based on the fact that 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.

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. For brevity, proofs are omitted, but can be found in [HMP06].

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

$$\mathrm{cfs}(x) \geq c_1 r$$

where c_1 depends only on τ and is independent of dimension.

We now a state a lemma that is functionally equivalent to Lemma 1, but allows the gap-ball to have one vertex encroaching it.

Lemma 2. *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 $x \in B$:*

$$\mathrm{cfs}_M(x) \geq c_2 r$$

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 3. *(**Quality Gives Bounded Grading***)*
Suppose M is a τ-quality Voronoi diagram. Then there exists a constant c_3 depending only on τ such that $\Gamma_M(x) \leq c_3$ for any $x \in CC(M)$.

Lemma 4. *(**Bounded Grading Gives Quality***)*
Suppose we have a Voronoi diagram M, and suppose that $\Gamma_M(p) \leq \tau$ at every vertex $p \in M$. Suppose further that every Voronoi node of M is contained in $CC(M)$. Then M is a 2τ-quality Voronoi diagram.

We note that the hypothesis for Lemma 4 is satisfied when we have a τ-quality mesh such that the diametral ball of every mesh simplex on the convex hull is unencroached.

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 neighbor. By controlling the spacing of output vertices, Ruppert could then prove that his algorithm terminates with a mesh of guaranteed size. We will do the same; and we also need to control the spacing in order to achieve our time bounds.

The statement of the theorem is as follows:

Theorem 1. *For any point p considered for insertion into a mesh M, whether or not p is eventually inserted, we have that $\mathrm{lfs}(p) \leq C NN(p)$*

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

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

The proofs herein are somewhat simplified from those presented by many prior authors, even as they are generalized to higher dimension. As in most such analyses, we split the single constant c into a constant C_i for every dimension $1 \dots d$. In our algorithm, unlike in most, we need an additional constant C_0 for the spacing of input vertices when they are inserted.

Lemma 5. *Let p be an input vertex being inserted by SVR. Then:*

$$\mathrm{lfs}(p) \le (1 + \frac{C_1}{k})NN_M(p)$$

Proof. Let v be the vertex in whose cell p lies. If v has containment dimension 0, then the lemma is obvious. Otherwise, we know $\mathrm{lfs}(v) \le C_1 NN_{M_v}(v)$ by the usual inductive argument. Furthermore, we know that v did not yield to input vertex p when v was inserted, which means that $|vp| \ge kNN_{M_v}(v)$. Finally, we use the Lipschitz condition and some algebra to prove the result. This shows that $\boxed{C_0 \ge 1 + C_1/k}$.

The bounds on C_i for $i > 0$ are exactly analogous to prior work but with a $1/k$ factor. This gives a linear program which we can solve for C_0, which shows that the sizing theorem holds whenever $\boxed{\tau k^d > 2^{d-1/2}}$. In other words, for any $\tau > 2^{d-1/2}$, there is a k close enough to 1 that allows the proof to go through. Setting $k = 1$ gives a correct algorithm, but we will see that the algorithm will run in time proportional to $(1 - k)^{-1}$ which suggests a tradeoff between solution quality and runtime.

7.1 Size Optimality

Mitchell and Vavasis [Mit93] shows that for $d = 3$, so long as the sizing condition is guaranteed – as it is for SVR, according to Theorem 1 –, then on a point set input, the mesh is within a constant factor of the smallest mesh that respects the input vertices. As shown by Shewchuk [She98], this proof fails in the presence of input features (even just segments) in dimension at least 3: by using slivers, on certain inputs, we can produce an arbitrarily smaller mesh than what is dictated by the local feature size. Given that slivers are undesirable, it is unclear exactly what the correct definition of size optimality is, when in the presence of features.

8 Timing Analysis Overview

8.1 Mesh update work

We claim that the entire refinement process modifies only $O(m)$ Voronoi cell boundaries over the life of the algorithm. Hence, the cost of maintaining the mesh is $O(m)$.

Proof (see [HMP06]) proceeds by first showing that the mesh is always a quality mesh at any point during the algorithm. It is then shown that the mesh is always sparse. Finally, a charging argument to count the total number of edges.

Lemma 6 (Always Quality). *At any point during Sparse Voronoi Refinement, the intermediate mesh is a τ'-quality mesh.*

Theorem 2 (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.*

Corollary 2 (Mesh Update Charge). *The number of Voronoi facets that are ever created is $O(m)$, so that the overall time spent updating the mesh is $O(m)$.*

Proof. Any time a Voronoi facet f is created, it is adjacent to a freshly inserted vertex v. We will charge the cost of creating f to the insertion of v. Consider the intermediate mesh immediately after the insertion of v. By Theorem 2, v is of constant degree, hence the number of facets charged to v is constant, so the total number of Voronoi facets that are ever created is linear in the total number of vertices ever inserted, hence $O(m)$.

8.2 Accounting for point location and relocation

We sketch a basic overview of the timing analysis for point location work. A rigorous analysis is omitted for brevity, but can be found in [HMP06].

Consider a feature and the sequence of meshes of that feature that the algorithm produces: $M_0, M_1, \ldots M_i, \ldots M_j, \ldots M_m$. We claim that through this sequence, no more than $O(1)$ point location events affect any ball B with center c before $\mathrm{cfs}(c)$ falls by half, i.e.:

Lemma 7. *If $\mathrm{cfs}_{M_j}(c) \geq \mathrm{cfs}_{M_i}(c)/2$ then B was relocated at most $O(1)$ times during the insertion of points p_i through p_j.*

Lemma 8. *If $\mathrm{cfs}_{M_j}(c) \geq \mathrm{cfs}_{M_i}(c)/2$ then B was searched or tested for encroachment at most $O(1)$ times during the insertion of points p_i through p_j.*

Together, these lemmas tell us that point location events must be in some sense "packed" around the points being located. It follows that the number of point location events around a point (or ball) to be inserted is bounded by the log of the ratio between the farthest and the nearest event.

For input points and features, this bounds the number of events to $\log(L/s)$. For created balls and queued points, it can be shown that the number of events is only a constant. Together this gives a bound on point location costs of $O(n \log(L/s) + m)$.

9 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

[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.

[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]   Benoit Hudson, Gary Miller, and Todd Phillips. Sparse Voronoi Refinement. Technical Report CMU-CS-06-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 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.

[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.

[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.