Runtime-Efficient Meshing for Piecewise-Linear Complexes*

Gary L. Miller Todd Phillips
Carnegie Mellon University

August 23, 2010

*This work was partially supported by the National Science Foundation under grant number CCF-0635257.

Abstract

We present a new meshing algorithm to mesh an arbitrary piecewise-linear complex in three dimensions. The algorithm achieves an $O(n \log \Delta + m)$ runtime where n, m, and Δ are the input size, the output size, and spread respectively. This runtime represents the first non-trivial runtime guarantee for this class of input. The new algorithm extends prior work on runtime-efficient meshing by allowing the input to have acute input angles (called creases). Features meeting at creases are handled with protective collars. A new procedure is given for creating these collars in an unstructured fashion, without the need for expensive size precomputations as in prior work. Output tetrahedra have quality radius-edge ratios in a region away from the "creases". Adjacent to creases, output tetrahedra have no large dihedral angles. The collar surface dividing these two regions is represented implicitly using surface reconstruction techniques. This new approach allows the algorithm to run in a single pass.

1 Introduction

One of the main goals of mesh generation is to represent functions efficiently on a computer. These functions may be the surface or body of a animation character, the temperature and pressure of a thermal reaction, or the control space of robotic arm. Critical to all these applications is an algorithm for finding this mesh. At first hand, one might think that the problem is so easy that it should not need such intense study. Unfortunately, meshing algorithms with strong guarantees seem to be more elusive than one might hope. There are many ways one can abstract the meshing problem we will use a linear set of features.

The input to a meshing algorithm is a set of input features. In our case these are a set of points, segments, and facets in a 3D domain. More formally this set is known as a piecewise linear complex (PLC), [MTT+96]. The meshing algorithm should decompose the space into tetrahedra, a mesh. The three important aspects of a mesh are that the mesh resolves all the input features (conforming), that mesh elements be well-shaped (quality), and that the number of elements be small (size-guarantee). One of the main missing components in current tetrahedral meshing algorithms research is the existence of refinement algorithms with good runtime analysis. Runtime analysis for some methods (notably those involving quad-trees or octrees) has been straightforward [MV00, HPÜ05] due to the very structured spatial decomposition. However, most practical meshing algorithms have very poor runtime guarantees. Meshing for inputs with no acute input angles and guaranteeing no small output angles has been achieved with an $O(n \log \Delta + m)$ runtime where n, m, and Δ are the input size, the output size, and spread (the ratio between the largest to smallest feature) [HMP06]. The goal of this paper is to extend this algorithm (Sparse Voronoi Refinement: SVR) to input domains that may include small input angles.

Ruppert [Rup95] was the first to address the issue of small input angles in refinement-based algorithms. His idea for 2D meshing was to add "small" protective balls around vertices with small input angles, thus, partitioning the domain into balls containing small input angles and the remaining area free of small input angles. In the interior of these balls he would use small-angled triangles while using no-small angle triangles outside the balls. In a series of papers, this idea has been extended to 3D by adding a set of balls (collars) around input points and edges that are common to a small input angles [MMG00, CSECY02, PW04, CP06, RW08]. Their constructions are nontrivial. Unfortunately, none of these algorithms have sub-quadratic runtimes even for inputs with polynomially bounded spread. In this paper we will show how to efficiently generate this partition into collars and to mesh the domain with only small angle elements interior to the collars and with no angles going to $180°$. The size of the mesh we return, m, will be no larger than other known collar based algorithms and often much smaller. The algorithm will have the further advantage that it generates the mesh in a single pass while previous algorithms first find the collars, adding the collars to the input features, and then start over to generate the mesh, possibly causing the mesh to be substantially larger.

1.1 Algorithm Overview

We follow a standard iterative Delaunay (Voronoi) refinement paradigm, wherein we continually add new vertices to a mesh with the two goals of increasing element quality and conforming to the input PLC. The algorithmic difficulties for handling acute angles in the PLCs arise due to a conflict between these two goals. It is impossible to fit quality tetrahedra inside creases of the PLC. We circumvented this conflict by giving two different guarantees on element quality.

Adjacent to the creases of the input, we generate tetrahedra with no large dihedral angles. In regions disjoint from the creases, we generate tetrahedra with a good ratio of circumradius to shortest edge. Almost all of the tetrahedra

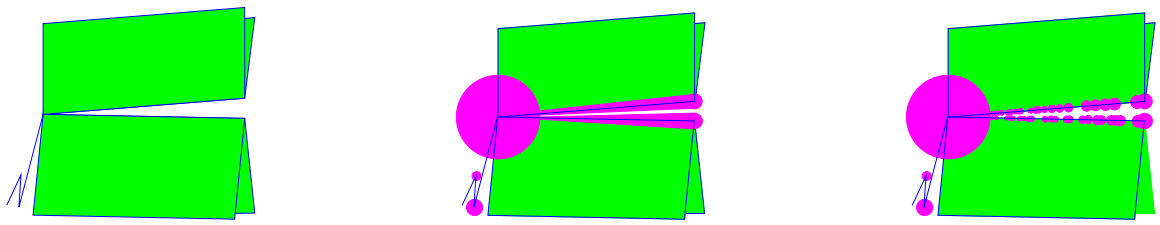

Figure 1: **Left:** An input consisting of four planes and several segments. **Center:** Proper collar regions are highlighted around the creases. **Right:** The collar region is approximated by a union of balls. Proper collar size is determined by the **gap size** (Definition 2).

have good aspect ratios, with the exception of slivers. (The sliver removal post-process of [LT01] can be modified easily to improve output aspect ratios without additional runtime cost.)

Unfortunately, a proper choice for these two regions is not known in advance, and the problem of distinguishing these regions is fundamentally the same as generating the entire mesh. We observe that the regions needing special treatment are precisely those where a traditional meshing algorithm will not terminate, refining forever based on the conflicting goals of quality and conforming.

As with most generic refinement algorithms, SVR does not terminate when the input features contain small angles. Even though such algorithms do not terminate, they do converge to a fixed mesh, though possibly of infinite size.

That is, if we pick any point p in the input domain other than a vertex at a small input angle, SVR will eventually include the point in a fixed, finite tetrahedron or Voronoi cell of the output. Our idea is to simulate the refinement algorithm that would generate an infinite mesh and use this mesh to specify our final finite mesh. Of course we will not let the mesh become of unbounded size in process.

In keeping with traditional approaches, we too create a *collar region* around the creases, and the mesh refinement within it is handled as a special case to ensure termination. We achieve better runtime by creating this special region dynamically. Our on-the-fly calculations allow the region to be computed in a single pass with the rest of the algorithm, avoiding any expensive preprocesses.

When the mesh elements in the neighborhood of a crease are approximately small enough, a proper size is computed and the collar region is augmented. Previous approaches to collar regions have either required expensive $O(n^2)$ pre-computations[CDR07] or assumed a lower-bound on size and created constant-sized collar regions [CDL07]. Newer approaches use an adaptive scheme similar to ours but without guarantees on runtime [CDR10]. We use careful predicates and show that the collar region is always augmented before refinement progresses too far, and that the collar-sizing calculations can always be performed efficiently. In three dimensions, the creases may be corners or whole segments. The collar region will consist of a union of balls that are centered on these creases and will eventually cover all of the creases.

Pseudo-code for the algorithm (with some implementation details omitted) is shown in Procedures 1-5.

1.2 Mesh Quality

One of the difficulties in meshing acute PLCs is to make guarantees on the quality of the output tetrahedra. Because good tetrahedra cannot be fit inside creases, a relaxed guarantee must be made on the quality of these tetrahedra. Our algorithm generates tetrahedra adjacent to the crease with no large dihedral angles. Any obtuse (down to $90° + \varepsilon$) angle guarantee can be specified by the user, with a possible tradeoff of a larger output size to achieve this angle.

Away from the creases, our algorithm generates tetrahedra with bounded radius-edge ratio, the ratio of circumradius to shortest edge. Our algorithm works for any bound on the ratio above $2\sqrt{2}$. The best bound achieved by Delaunay refinement algorithms is 2 in general. It may be possible to show that our algorithm achieves this bound with a stronger analysis.

Since our algorithm also generates good aspect-ratio Voronoi cells dual to the tetrahedra, the output achieves a stronger result than a radius-edge ratio bound. Considering a fixed vertex, the algorithm actually bounds the ratio of the largest circumradius of any adjacent tetrahedron to the shortest length of any adjacent edge.

In the regions away from creases, our algorithm may output *slivers*, tetrahedra with good radius-edge ratio but poor aspect ratio. Although we only claim the weak guarantee herein, we believe these sliver tetrahedra can be cleaned up

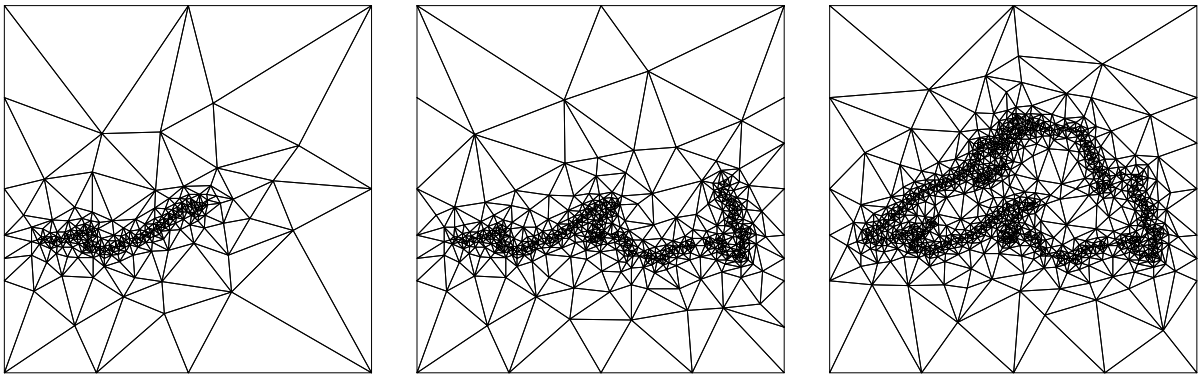

Figure 2: The generic Sparse Voronoi Refinement algorithm gradually resolves an input while iteratively maintaining a quality mesh, avoiding the computation of the Delaunay diagram of the input.

as a linear-time post-process similar to [LT01], but slightly modified to handle the collar regions. Such post-processes generally give only extremely slim guarantees on aspect ratio, but perform better in practice than in theory.

1.3 Runtime

There are essentially two failures of most Delaunay refinement algorithms with regards to provably good runtime. The first is due to pathological Delaunay diagrams, and the second is poor point location strategies. Our algorithm makes improvements in both cases.

A great many algorithms begin by constructing the Delaunay diagram of the input, which can have $O(N^2)$ edges in the worst case. Our algorithm, based on Sparse Voronoi Refinement, avoids this construction by inserting some Steiner points before resolving some parts of the input, to ensure that the complexity of the mesh is always linear in the number vertices.

The worst-case inefficiency in other algorithms is due to poor point location strategies. When the output size m is dominating the runtime cost (as is often the case in practice), it is important that the algorithm perform linearly in m. Although it is often left unanalyzed, many algorithms in the literature will only run in $O(m \log \Delta)$, particularly algorithms that generate quality meshes for each feature before filling space. After performing such "pre-meshing," all the vertices on these feature meshes must be located relative to one another, costing $O(\log \Delta)$ each in most schemes. Unfortunately, the total number of volume-filling vertices m is most often dominated by these surface vertices ([HMPS09]), and so pre-meshing ends up incurring $O(m \log \Delta)$ work.

2 Definitions

A 2D piecewise-linear complex (embedded in 3D) is a set of features, where each feature is a vertex, a segment, or a polygon with polygonal holes. The set should be closed under taking intersections or boundaries. For many PLCs, very basic meshing algorithms will suffice. The main difficulty for meshing arbitrary PLCs comes when features meet at acute angles. In such situations, it can become impossible to fit quality tetrahedra in the interior of these angles, and so more complicated algorithms must arise.

Definition 1 (Angle Between Features) *Consider a 2D PLC. Suppose two features F and G intersect at a feature H. Define $rays(H, F)$ as the set of all rays emanating from a point in H, orthogonal to H, and heading into F. Then the* **angle between features** *F and G is the minimum angle subtended by any pair from $rays(H, F)$ and $rays(H, G)$.*

This leads to the definition of a **crease** as a feature, where any two features meet at an acute angle. Meshing algorithms for PLCs in three dimensions easily handle PLCs without creases [She98, PW04]. The first volume-filling algorithm for arbitrary PLCs was given by Cheng and Poon [CP06]. Other conforming algorithms have followed [PW05, RW08]. All of the algorithms for acute PLCs in two and three dimensions have a similar flavor; creases are

protected by a system of **collars** (sometimes "intestines"), and special procedures are explicitly utilized for meshing within and around the collars. The sizing for these collars is typically determined by some variant of the **gap size**:.

Definition 2 (Gap Size [CP06]: g) *Given a PLC $\mathcal{P}$, the gap-size $g_{\mathcal{P}} : (\mathbb{R}^3 \to \mathbb{R})$ is defined at x as the shortest distance to two features,* one of which does not contain x*:*

$$g_{\mathcal{P}}(x) := \operatorname*{argmin}_{r \in (0,\infty)} \Big[\exists\, F, G \in \mathcal{P} \mid F \neq G \text{ and } B(x,r) \cap F \neq \emptyset \text{ and } B(x,r) \cap G \neq \emptyset \text{ and } x \notin F \Big]$$

Generally, we will supress the subscript $\mathcal{P}$ when the PLC in question is the input. The function g may be very discontinuous, but it is 1-Lipschitz along the interior of any feature [CDR07]. The algorithm will construct protective collars at and along creases as a system of balls that are centered on the creases. The radius of a ball centered at c will be given by $g(c)/3$, and enough balls will be taken to cover the creases (see Figure 1.1). Note that outside this collar region, $g(x)$ is bounded away from zero. The reason for sizing collars according to gap size is to maximize the size of the collar region without causing non-local intersections. output vertices. The collar region is maximized to help diminish the number of output vertices. If the collar regions were extremely small, the final mesh would fill the region between two acutely-meeting features with a huge number of tiny tetrahedra before stopping at the collar boundary.

To create such collars, a meshing algorithm must know values of g along creases. Previous algorithms took these functions as given, naively requiring brute-force computations taking $\Omega(n^2)$ [CDR07]. Later algorithms calculate approximations to sizing procedurally[DL09]. This works well in practice. The methods used, however, do not have good runtime guarantees. Our main contribution is a work-efficient procedure for approximating the collar sizing. This procedure could be run standalone, but is easily inlined into the SVR algorithm, allowing time-efficient meshing for abitrary PLCs.

In the interior of a feature, outside the collar region, the gap-size is closely related to the **local feature size**, which governs the spacing of vertices in an optimal no-small-angle mesh of a non-acute PLC.

Definition 3 (f: Local Feature Size [Rup95]) *Given a PLC $\mathcal{P}$, the local feature size $f_{\mathcal{P}} : (\mathbb{R}^3 \to \mathbb{R})$ is given by the shortest distance to two* disjoint *features of $\mathcal{P}$:*

$$f_{\mathcal{P}}(x) := \operatorname*{argmin}_{r \in (0,\infty)} \Big[\exists\, F, G \in \mathcal{P} \mid B(x,r) \cap F \neq \emptyset \text{ and } B(x,r) \cap G \neq \emptyset \text{ and } F \cap G = \emptyset \Big]$$

If $\mathcal{P}$ is a point set, then this is simply the distance to the second-nearest neighbor.

2.1 Sizing and the Infinite Mesh

Suppose we ran the generic SVR on an arbitrary PLC. The algorithm would run forever, but it is useful to abstractly consider its output, which we will call the infinite mesh. The output is a quality mesh, containing countably many points. Using standard refinement analysis, it is easily seen that the sizing of elements in this mesh is bounded below by the following function $\bar{g}$:

Definition 4 (Infinite Sizing $\bar{g}$) *Given a set of features $\mathcal{P}$, define the function $\bar{g}_{\mathcal{P}}$ as the largest function such that $\bar{g}_{\mathcal{P}} \leq g_{\mathcal{P}}$ and $\bar{g}_{\mathcal{P}}$ is 1-Lipschitz.*

Note that $\bar{g}$ goes to zero at all the creases, corresponding to the accumulation points of the infinite mesh. However, note that $\bar{g}$ is bounded away from zero in regions away from all the creases, implying that generic meshing will converge to some final triangulation in these exterior regions with size at least $\bar{g}$. Note also that $\bar{g}$ and g are essentially the same function in the exterior regions, with the exception where features meet obtusely. To simplify the discussion, consider putting collars around all feature intersections, and not just the creases. This simplification is removed trivially in the actual algorithm and analysis.

Suppose we have a collar system covering the creases. Let Ω_E be the portion of the domain Ω exterior to these creases. Then consider the following:

Definition 5 (Exterior Sizing $\hat{g}$) *Given a set of features $\mathcal{P}$ and an exterior region Ω_E, let $\hat{g}_{\mathcal{P}}$ be the largest function such that $\hat{g}_{\mathcal{P}} = g_{\mathcal{P}}$ when restricted to Ω_E and $\hat{g}$ is 1-Lipschitz.*

A quality mesh for an arbitrary PLC will have elements sized according to $\hat{g}$ everywhere.

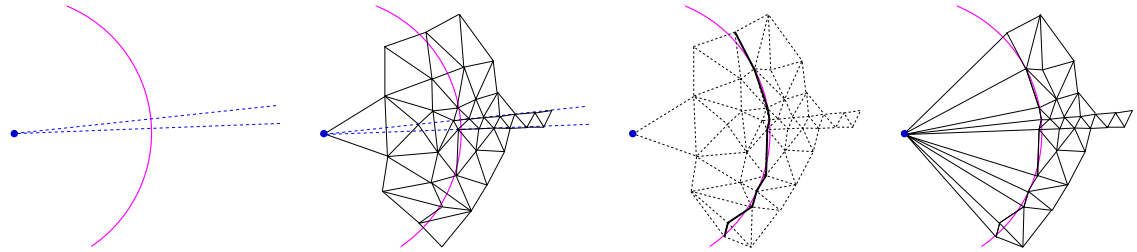

Figure 3: **From left to right:** An input with two lines meeting acutely and a collar region at the crease. The algorithm begins constructing an infinite quality conforming mesh, but stops early by ignoring the input within the region. A submesh to represent the collar surface is selected, and further refinement ensures proper surface approximation. Lastly, extraneous vertices within the collar region are discarded and replaced by a star emanating from the crease. The final mesh conforms to the two original input segments.

3 Modifications to SVR

3.1 Basic Algorithm

We recall a basic overview of the SVR Meshing Algorithm. Figure 1.3 gives snapshots of the algorithm running. The SVR algorithm iteratively maintains a Voronoi diagram V of inserted points. Additionally, it maintains queue of uninserted points and protective circumballs around unresolved input features. We make use of two structural properties from the SVR algorithm:

First, SVR always inserts new points that are θ**-medial** with respect to the current V for some $0 < \theta < 1$, meaning that the ratio of the nearest to second-nearest neighbor of a new point is bounded below by θ.

Second, $\mathcal{V}$ is always a τ**-well-spaced** Voronoi diagram for some $\tau > 1$. If V is a Voronoi cell of vertex v in $\mathcal{V}$, let R_v be the radius of the smallest v-centered ball containing V, and let r_v be the largest ball contained in V. V is τ-well-spaced iff $R_v/r_v < \tau$.

3.2 Guard Collars

The algorithm adds spheres to the collar region dynamically, with the goal of eventually covering the entirety of the creases. Once a sphere is added, the algorithm no longer attempts to conform to the PLC interior to the sphere. Attempting to define a collar region too soon in the execution would result in a burdensome calculation. Defining a collar too late will result in excess work due to over-refinement. The goal of this subsection is to prove that this procedure in fact works correctly and quickly.

It is safe to calculate the gap-size at a point x once the mesh has refined down to a size near $g(x)$, so that the computation is mostly local. But since $g(x)$ is what we wish to calculate to begin with, we must use a different criteria which is testable, and then show that this is equivalent. The algorithm employs the following definition:

Definition 6 (Hub Feature and Isolated Point) *Consider a mesh M any Voronoi cell V centered at $v \in M$. Let $\mathcal{P}|_V$ be a PLC restricted to V. Consider any nontrivial feature $F|_V \in \mathcal{P}|_V$. F is a* **hub feature** *with respect to V if $F|_V \subset G|_V$ for any $G \in \mathcal{P}$. In other words, F is the unique minimal feature of $\mathcal{P}_V$. We say that a point $x \in F|_V$ is* **isolated** *with respect to M.*

A similar notion of a hub feature appears in [ES97].

The algorithm acts in a just-in-time fashion, adding a collar around a vertex on a crease as soon as it is isolated. The following lemma guarantees that this is not too soon by relating it back to the gap-sizing (proof is in the appendix):

Lemma 1 (Isolated is Equivalent to g) *Let x, $\mathcal{P}$, M, and v as in Definition 6, then $R_v \leq g(x)/2$ iff x is isolated.*

The other direction is to prove that the algorithm puts up a collar in time, before the mesh has been refined more than a constant factor (ε_0) smaller than the collar that will be placed.

Procedure 1 Main method for the algorithm. Initialization with bounding boxes is not shown for simplicity. The main loop repeatedly DESTROYS gap balls to achieve conforming and quality goals. Balls are typed in a priority order to ensure good runtime. Maintenance of work queues is not shown for simplicity. Finally, the call to RIPSTITCHCOLLAR removes whatever cruft is inside the collar system and replaces it with a no-large-angle conforming tetrahedralization.

SVRCOLLARS($\mathcal{P}$: a piecewise linear complex, τ, θ, σ)
1: $B :=$ DEQUEUE()
2: **while** WorkQueue is non-empty **do**
3: **if** possible DESTROY(any large ball adjacent to a skinny Voronoi cell)
4: **else** if possible DESTROY(any encroached protective ball)
5: **else** if possible DESTROY(any ball containing unresolved input)
6: **else** if possible DESTROY(any poor quality collar representation ball)
7: **return** RIPSTITCHCOLLAR()

Lemma 2 (Crease Recovery) *Every vertex v created on a crease of $\mathcal{P}$ is eventually sent as an argument to the function* CREATECOLLAR. *Furthermore, there exists some constant $\varepsilon_0 > 0$ such that the inequality dist$(v, M) > \varepsilon_0 g(v)$ holds for the mesh M at the time when* CREATECOLLAR(v) *is called.*

Proof is in the appendix. This lemma guarantees that the *vertices* on creases are resolved early enough. A technicality is to make sure that the rest of the crease (along segments in particular) is protected as well. We claim this region is protected for some constant ε_2:

Lemma 3 (Crease Protection) *There exists a constant ε_2, such that no* UNRESOLVED *work event is ever enqueued within $\varepsilon_2 g(x)$ of any point x on a crease.*

Proof is in the appendix.

3.3 Runtime for Calculating g

When a vertex becomes isolated, the algorithm must then calculate g at the vertex. This calculation is done naively; the algorithm walks around neighboring Voronoi cells in a breadth-first fashion, and collects all the unresolved and resolved input that intersects these cells (see Procedure 4). The claim is that the additional work to implement this calculation of g is asymptotically no more than SVR. A full proof is given in the appendix. We sketch a proof here:

Lemma 4 (Sizing Exploration Runtime) *The time to calculate $g(v)$ for every v inserted on a crease totals at most $O(n \log \Delta + m)$.*

Proof: Consider the mesh M when a vertex v is inserted on a crease. The procedure for calculating $g(v)$ is straightforward. Witnesses to $g(v)$ may be unresolved input points (linked to cells of M through the existing point location structure) or they may be vertices in a mesh already. The procedure is to explore the mesh M in a breadth-first fashion, checking distance ("interviewing") all potential witnesses until $g(v)$ is found.

There are two parts to the runtime complexity, the first is to bound the number of cells explored, and the second is to bound the number of potential witnesses interviewed.

Consider the new Voronoi cell around v with inradius r_v; $B(v, r_v)$ does not intersect M. By Lemma 2, $r_v \in \Omega(g(v))$, which means the algorithm need only explore enough cells to cover some $B(v, O(r_v))$. But since the algorithm is processing smallest cells first, the explored cells have size $\Omega(r)$, and so by simple packing, only a constant number of cells are explored. This constant can be charged to the insertion of v to give total work $O(m)$.

This $O(m)$ also bounds the number of potential witnesses that are already resolved. The number of potential witnesses that are unresolved input points is, in fact, unbounded for any given insertion. Instead, we will count the total number of interviews in an amortized fashion. Consider a witness w, and consider every insertion v' that interviews w. For each possible v', there is an empty ball of radius $r_{v'}$ at interview time because v' is isolated. But w and v are near each other; $dist(w, v') \in O(r_{v'})$ as before. A packing argument shows the number of these empty balls per annulus around w is a constant, so that the total number of insertions interviewing a given w is $O(\log \Delta)$, giving $O(n \log \Delta)$ total work. □

Procedure 2 The central operation of our algorithm is to continually DESTROY large gap balls by adding a point within them. If there is an unresolved vertex p within θr of c, then c should warp to p conforming purposes(line 4). In line 3, the algorithm carefully inserts the candidate of lowest containment dimension. This ensures that points on creases are added before too many points are added near a crease, so that collars can be created to stop refinement. Otherwise, DESTROY then looks for any nearby lower-dimensional protective balls encroached by c. If there aren't any encroached balls, it is safe to insert c (line 10). If there is some encroached ball B', this call to DESTROY is a witness that B' must be destroyed, so yield to destroying B'. The destruction of B' may have created some new points to warp to so go back and check everything again (line 8). The runtime proofs of SVR bound this repeated work.

DESTROY($B = \langle (c, r), F \rangle$: a BALL)

1: **if** B is no longer an empty ball in F **then return**
2: **if** There exists a point p in a lower dimensional mesh within θr of c **then**
3: Choose p minimizing $\mathrm{CD}(p)$
4: FORCEINSERT(F, p)
5: **else**
6: **if** c is interior to a lower dimensional ball B' **then**
7: DESTROY(B')
8: **goto** 2
9: **else**
10: FORCEINSERT(F, c)

Procedure 3 The FORCEINSERT routine always inserts a point p into a mesh of a feature F. If any crease vertex is now isolated (Definition 6), augment the collar system to protect it. Lastly, enqueue any new skinny cells for destruction. Updates to the point location data structures are omitted for simplicity.

FORCEINSERT(F: feature to add to, p: a point)

1: Add p to the Voronoi diagram of F
2: **for** any vertex v on a crease(possibly unresolved) that is now **isolated do**
3: **if** $dim(v) < dim(F)$ **then**
4: CREATECOLLAR(v, F)
5: Enqueue any new skinny Voronoi Cells

4 Smooth Collar Surface

Once the algorithm has terminated, the area inside the collars is gutted and filled with a new tetrahedra. However, care must be taken at the edge of these regions to ensure two properties. First, that a proper tetrahedralization of the whole domain is generated. Second, that the tetrahedra adjacent to the creases have some quality guarantees.

Past algorithms take one of two approaches. The first is to use the weighted Delaunay triangulation to ensure proper mesh topology. But this gives no guarantees on the tetrahedra adjacent to the crease. The second is to maintain some explicit template triangulation of the collar surface throughout an algorithm. Our new approach is to use methods from surface reconstruction to select a sub-triangulation of the mesh that approximates the collar surface well. To do this we use the **restricted Delaunay** triangulation of the collar surface. The novelty of our approach is in its laziness: new vertices are only added as necessary. When the collar is first created, the existing nearby mesh vertices might imply a boundary of the collar region with sufficient quality, and in this case no new vertices would be added.

We will add two new types of work items to the work queue for SVR. The first is for collar topology. Whenever the implicit surface representation would imply incorrect connectivity in the output, the algorithm will add another vertex on the border of the collar region. The second is to achieve no large angles on the tetrahedra adjacent to the crease. If the collar surface triangulation approximates the normals of the collar surface well, then putting in stars of tetrahedra around a collar ball centers will result in a triangulation with no large angles. Accordingly, we will add a work event for whenever there is a triangle of the collar surface that poorly approximates the normal. A genericity constraint will ensure that the algorithm never attempts to measure the normal at the intersection of two spheres, so that it is well-defined.

Procedure 4 CREATECOLLAR computes the size of a proper collar ball around v. A neighborhood of v is explored. g keeps track of the nearest feature that has been found. Loop until there cannot be a feature nearer to v. When the loop terminates, $g = g(v)$. The actually procedure is rather brute force, with the cleverness in its analysis (Section 3.2).

CREATECOLLAR(v: vertex on crease, F: feature needing collar)
1: Initialize $\mathcal{F} = \emptyset$, the set of features near v
2: Initialize $g = \infty$, the size of the neighborhood needing to be explored
3: Initialize $X = \emptyset$, the region that has been explored
4: **while** $B(v, g) \not\subset X$ iterate on $v' \in M_F$ in breadth-first order beginning with v **do**
5: Update $X := X \cup V_{M_F}(v')$
6: Update $\mathcal{F} := \mathcal{F} \cup \{F' \mid \exists \langle (c, r), F' \rangle \in \text{CONFBALLS}(v') \text{ and } F' \neq F\}$
7: Update $g := \min\{|vF'| \mid F' \in \mathcal{F}\}$
8: Add the ball $B(p, g/3)$ to the collar system, update the collar surface and restricted Delaunay

Procedure 5 RIPSTITCHCOLLAR performs a post process on the volume mesh M to replace the triangulation inside the collar system. First compute the collar region, the crease vertices $\mathcal{V}$, and the vertices internal to the collar region $\mathcal{I}$. Rip out the collar regions in line 4. The first loop then pairs up crease points with collar spheres, and the inner loop pairs crease edges with collar circles. The second loop goes over all the pairs and fills in $\mathcal{T}$ by stitching in a star around each crease point and stitching a book around each crease edge.

RIPSTITCHCOLLAR() $\rightarrow$ A Triangulation of Ω
1: Set $\mathcal{T}$:= the Delaunay diagram dual to M
2: Compute $\mathcal{V}$, the set of all vertices on creases
3: Compute $\mathcal{I}$, the set of all vertices whose Voronoi cells are entirely interior to the collar region
4: Remove from $\mathcal{T}$ all simplices adjacent to $\mathcal{I}$
5: Initialize a set of pairs $\mathcal{P} = \emptyset$, each pair will be a vertex and sphere or edge and circle
6: **for all** $v \in \mathcal{V}$ **do**
7: **if** there is a collar sphere S centered at v **then**
8: Add $\langle \{v\}, S \rangle$ to $\mathcal{P}$
9: **if** v is on a crease edge **then**
10: **for all** Circles C formed by two interesecting balls, one of which is centered at v and the other at some v' (there are 2) **do**
11: Add $\langle \{v, v'\}, C \rangle$ to $\mathcal{P}$
12: **for all** $\langle T, C \rangle \in \mathcal{P}$ **do**
13: Compute $\mathcal{R}$, the restricted Delaunay simplices of C
14: **for all** $T' \in calR$ **do**
15: Add $\text{Sim}(T' \cup T)$ and its subsimplices to $\mathcal{T}$
16: **return** $\mathcal{T}$

4.1 Restricted Delaunay and Voronoi

Definition 7 (Collar System) *A* **collar system** $\mathcal{B}$ *is a set of* **collar balls** $B(c, r) \in \mathcal{B}$. *Define the* **collar surface** $\mathcal{S}$ *as the boundary of* $\bigcup \mathcal{B}$. $\mathcal{S}$ *must have the following requirement: If a point* $s \in \mathcal{S}$ *is contained in more than one ball, then* s *must be contained in some circle* $C \subset \mathcal{S}$ *formed by the intersection of balls in* $\mathcal{B}$. *This disjointness property guarantees that* $\mathcal{S}$ *can be described as a collection of circles* $\mathcal{C}$ *and partial spheres* $\mathcal{A}$. *Call this pair* $\langle \mathcal{C}, \mathcal{A} \rangle$, *the* **collar description** *of a collar surface.*

Lemma 5 (g-sized balls form a system) *Suppose* $\mathcal{P}$ *is a PLC and* $\mathcal{B}$ *is a set of balls centered on the creases of* $\mathcal{P}$ *satisfying* $r < g(c)/2$ *for every* $B(c, r) \in \mathcal{B}$. *Then* $\mathcal{B}$ *is a collar system.*

A proof is straightforward by noticing that any intersecting balls lie on a common segment and sweeping a plane along the segments reveals the proper topology.

A hallmark definition for surface reconstruction theory is the restricted Delaunay diagram of a set of vertices restricted to a surface. For simplicity, we will only consider collar surfaces, but the definition is general:

Definition 8 (Restricted Delaunay and Voronoi of a Collar Surface) *Let a point set* M *and a collar surface* $\mathcal{S}$ *be given. Define the* **Restricted Voronoi diagram** *of* M *restricted to* $\mathcal{S}$, *denoted* $\text{Vor}(M)|_{\mathcal{S}} \subset \text{Vor}(M)$, *as the subset of*

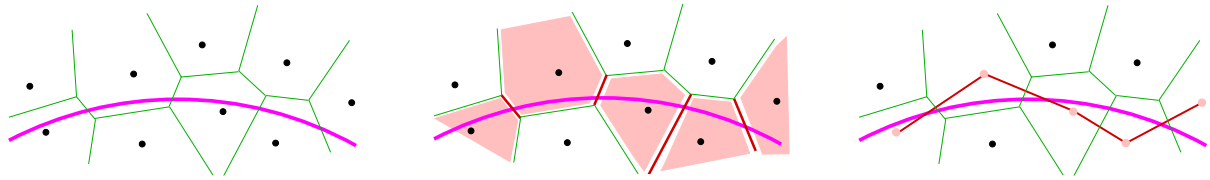

Figure 4: *Left:* A Voronoi diagram $\mathrm{Vor}(M)$ and a surface $\mathcal{S}$ cutting through. *Center:* The restricted Voronoi cells $\mathrm{Vor}(M)|_{\mathcal{S}}$. *Right:* The restricted Delaunay triangulation $\mathrm{Del}(M)|_{\mathcal{S}}$ approximates $\mathcal{S}$.

Voronoi polytopes that intersect $\mathcal{S}$:

$$\mathrm{Vor}(M)|_{\mathcal{S}} := \{V \in \mathrm{Vor}(M) | V \cap \mathcal{S} \neq \emptyset\}$$

Define the **Restricted Delaunay Diagram** *($\mathrm{Del}(M)|_{\mathcal{S}} \subset \mathrm{Del}(M)$) as the dual simplices:*

$$\mathrm{Del}(M)|_{\mathcal{S}} := \{\mathrm{Dual}(V) \mid V \in \mathrm{Vor}(M)|_{\mathcal{S}}\}$$

See Figure 4.1.

The restricted Delaunay definition also applies for collar surfaces in lower dimension. The restricted Delaunay definition implicitly assumes a generic intersection property on the collar description and the point-set: no Voronoi corner lies on a partial sphere; no Voronoi facets or edges are tangent to a sphere; no Voronoi corners or edges intersect a circle; and no Voronoi facets are tangent to a circle. As the exact radii of a collar ball is never critical in the algorithm, this can always be achieved by perturbation.

Observe that if any vertices of M are actually on $\mathcal{S}$, they will always be contained in the restricted Delaunay. The restricted Delaunay is closed under subsets: if $\mathcal{S}' \subset \mathcal{S}$, then $\mathrm{Del}(M)|_{\mathcal{S}} \subset \mathrm{Del}(M)|_{\mathcal{S}'}$. Thus, it makes sense to associate portions of the restricted Delaunay with pieces of the collar description. (Note that this does not yet necesarily partition the restricted Delaunay, some simplices may be associated with more than one sphere or circle.)

We will employ the following definitions regarding $\mathrm{Del}(M)|_{\mathcal{S}}$:

Definition 9 (Representation Sets) *Consider M, $\mathcal{S}$ and $\mathrm{Del}(M)|_{\mathcal{S}}$ as above. For a simplex $T \in \mathrm{Del}(M)|_{\mathcal{S}}$, define its* **representation set** $\mathrm{Rep}(T) := \mathrm{Dual}(T) \cap \mathcal{S}$. *For a triangle, the dual is a voronoi edge, so this yields a set of* **representation points**. *For an edge, call it a* **representation curve**. *For a vertex, call it a* **representation patch**. *(Note that these sets may not actually be a single curve or patch.)*

Representation sets are also defined in two dimensions. The restricted Delaunay does a good job approximating a surface. Consider that as more and more points are drawn from $\mathcal{S}$ and added to M, $\mathrm{Del}(M)|_{\mathcal{S}}$ converges to $\mathcal{S}$ in several norms. The main goal is to show is that $\mathrm{Del}(M)|_{\mathcal{S}}$ has the same topology as the collar surface. The chief technology employed in such results is based on the theorem of Edelsbrunner-Shah[ES97], which requires the "topological-ball property", that the intersection of each representation set with the surface is a topological ball of proper dimension. We will achieve this property by enforcing a set of topological constraints.

Definition 10 (Consistent Sample) *Given a point set M and a collar surface $\mathcal{S}$, a simplex $T \in \mathrm{Del}(M)|_{\mathcal{S}}$ is called* **locally consistent** *if $\mathrm{Rep}(T) \cap \mathcal{S}$ is a topological $(|T| - 1)$-ball. M is a* **consistent sample** *for $\mathcal{S}$ if every simplex of $\mathrm{Del}(M)|_{\mathcal{S}}$ is locally consistent.*

Concretely, this definition means that every set of representation points must be a singleton, every representation curve must be a topological 1-ball, and every representation patch must be topological 2-ball.

A restricted Delaunay simplex T can always verify its local-consistency by testing only the subset of collar balls that intersect $\mathrm{Dual}(T)$. Note that if M is a consistent sample of $\mathcal{S}$, this does not necessarily imply that M is a consistent sample of some subset of $\mathcal{S}$.

Definition 11 (Collar-Consistent Sample) *A consistent sample M of $\mathcal{S}$ is a* **collar-consistent sample** *if M is a consistent sample for the set of circles in the collar decomposition of $\mathcal{S}$.*

Lemma 6 (Collar Surface Topology) *If M is a collar-consistent sample for a collar surface $\mathcal{S}$, then $\mathrm{Del}(M)|_{\mathcal{S}}$ is a triangulation homeomorphic to $\mathcal{S}$. Furthermore, M agrees with the collar decomposition, so that the restricted Delaunay of M restricted to any partial sphere is a triangulation homeomorphic to the partial sphere, and the restricted Delaunay of M restricted to any circle forms a cycle.*

Proof: Since M is a consistent sample, it will meet "the extended topological ball property" of the Edelsbrunner-Shah [ES97], which will guarantee that $\mathrm{Del}(M)|_{\mathcal{S}}$ is homeomorphic to the collar surface. Since M is a collar-consistent sample, then for the set of circles $\mathcal{C}$ in the collar description, $\mathrm{Del}(M)|_{\mathcal{C}})$ is a set of cycles. By nesting of the restricted Delaunay, this is a subset of $\mathrm{Del}(M)|_{\mathcal{S}}$. These cycles partition $\mathrm{Del}(M)|_{\mathcal{S}}$ in to the restricted Delaunay of each partial sphere. □

The algorithm also seeks to approximate the geometry of the collar surface, motivating the following:

Definition 12 (Representation Angle) *Consider M, $\mathcal{S}$, and a triangle $T \in \mathrm{Del}(M)|_{\mathcal{S}}$. $\mathrm{Dual}(T)$ is a Voronoi edge E that pierces $\mathcal{S}$ at $\mathrm{Rep}(T)$. For a point $p \in \mathrm{Rep}(T)$, p is on some partial sphere S of the collar description. The* **representation angle** *$\sigma(T, M, \mathcal{S})$ is measured at p as the angle between E and the normal of S at p. If $\mathrm{Rep}(T)$ is not a single point, take the largest (worst) representation angle.*

The representation angle gives a measurement of how tilted a triangle T is relative to the local portion of the collar surface $\mathcal{S}$. Note that by perpendicularity, σ is also the angle between the plane containing T and the plane tangent to $\mathcal{S}$ at $\mathrm{Rep}(T)$ Furthermore, if all the vertices of T lie on the same partial sphere S, then $\sigma = 0$.

The algorithm will add points on the collar surfaces to guarantee collar consistency and to obtain a bound on representation angles. The bound on representation angles is used to guarantee the quality of simplices that are created adjacent to creases.

References

[CDL07] S. Cheng, Tamal K. Dey, and Joshua A. Levine. A practical delaunay meshing algorithm for a large class of domains. In *Proceedings of the 16th International Meshing Roundtable*, pages 477–494, 2007. 3

[CDR07] S. Cheng, Tamal K. Dey, and Edgar A. Ramos. Delaunay refinement for piecewise smooth complexes. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1096–1105. ACM Press, 2007. 3, 5

[CDR10] Siu-Wing Cheng, Tamal K. Dey, and Edgar A. Ramos. Delaunay refinement for piecewise smooth complexes. *Discrete and Computational Geometry*, 43:121–166, 2010. 3

[CP06] Siu-Wing Cheng and Sheung-Hung Poon. Three-dimensional delaunay mesh generation. *Discrete Comput. Geom*, 36:419–456, 2006. 2, 4, 5

[CSECY02] David Cohen-Steiner, Éric Colin de Verdière, and Mariette Yvinec. Conforming Delaunay Triangulations in 3D. In *18th Son Comp. Geom.*, pages 199–208, June 2002. 2

[DL09] T.K. Dey and J.A. Levine. Delaunay meshing of piecewise smooth complexes without expensive predicates. *Algorithms*, 2:1327–1349, 2009. 5

[ES97] Herbert Edelsbrunner and Nimish R. Shah. Triangulating topological spaces. *IJCGA*, 7:365–378, 1997. 6, 10, 11

[HMP06] Benoît Hudson, Gary Miller, and Todd Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, Birmingham, Alabama, 2006. Long version available as Carnegie Mellon University Technical Report CMU-CS-06-132. 2, 14

[HMPS09] Benoît Hudson, Gary L. Miller, Todd Phillips, and Donald R. Sheehy. Size complexity of volume meshes vs. surface meshes. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 2009. 4

[HPÜ05] Sariel Har-Peled and Alper Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. In *Symposium on Computational Geometry*, 2005. 2

[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. 3, 4

[MMG00] Michael Murphy, David M. Mount, and Carl W. Gable. A Point-Placement Strategy for Conforming Delaunay Tetrahedralization. In *Proceedings of the Eleventh Annual Symposium on Discrete Algorithms*, pages 67–74. Association for Computing Machinery, January 2000. 2

[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 *5th International Meshing Roundtable '96*, pages 47–61, Pittsburgh PA, October 1996. Sandia National Laboratories. 2

[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. 17

[MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370 (electronic), 2000. 2

[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. 2, 4

[PW05] Steven E. Pav and Noel J. Walkington. Delaunay refinement by corner lopping. In *Fourteenth International Meshing Roundtable*, pages 145–156. Sandia National Laboratories, September 2005. 4

[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). 2, 5

[RW08] Alexander Rand and Noel Walkington. 3d delaunay refinement of sharp domains without a local feature size oracle. In *17th Internation Meshing Roundtable.*, 2008. 2, 4

[She98] Jonathan Richard Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pages 86–95, Minneapolis, Minnesota, June 1998. Association for Computing Machinery. 4

A Guard Collars

This is a longer version of section 3.2

Recall that SVRC adds balls to the system dynamically, with the goal of eventually covering the entirety of the creases. The goal of this subsection is to prove that this procedure in fact works, an essential part of the termination of the algorithm. An argument will show that the current mesh is not too small in the neighborhood of a crease before it is covered by the collar system. The notion of "not too small" in this lemma will also be useful later in proving that there are not too many extraneous vertices later removed from the collar region during RIPSTITCHCOLLAR, essential to the overall runtime of the algorithm.

Recall the following definition from the algorithm:

Definition 13 (Isolated Point) *Consider a point $x \in \mathcal{P}$ and any mesh M. Let F be the parent feature of x, and v be the vertex of M such that $x \in V_M(v)$. The point x is* **isolated** *with respect to M and $\mathcal{P}$ if every feature of $\mathcal{P}$ that intersects $V_M(v)$ contains x.*

This definition is useful because it is easily testable during the run of SVRC. To relate this to the proof technology, the following simple lemma exposes a relation between isolation and the sizing g:

Lemma 7 (Isolated is Equivalent to g) *Let x, $\mathcal{P}$, M, and v as in Definition 6. If $R_v \leq g(x)/2$ iff x is isolated.*

Proof: Follows directly from the definition of g. Clearly $B(x, g(x))$ does not intersect any features except those containing x. Consider $x \in B(v, R_v) \in B(v, g(x)/2)$, but then $B(v, g(x)/2)$ must be a subset of $B(x, g(x))$, and so x is isolated. Prove the other direction by contrapositve. Let w be a witness to $g(x)$, i.e. $g(x) = |xw|$. If x is not isolated, then WLOG $w \in V(v)$, so $g(x) = |xw| \leq |xv| + |vw| \leq 2R_v$. □

Now I prove the main lemma:

Lemma 8 (Crease Recovery) *Every vertex v created on a crease of $\mathcal{P}$ is eventually sent as an argument to the function* CREATECOLLAR. *Furthermore, there exists some constant $\varepsilon_0 > 0$ such that the inequality $n_M(v) > \varepsilon_0 g(v)$ holds for the mesh M at the time when* CREATECOLLAR(v) *is called.*

Proof: Let v be given. Recall from the algorithm that v is marshalled to CREATECOLLAR if ever it is the case that v is θ_0-medial and is isolated, or if v is inserted.
Recall further that $\theta_0 := \frac{2\theta}{2\theta+1} \in (0, 1)$, where SVRC insertions warp to points within θ times the radius when inserting a circumcenter.

Take

$$\varepsilon_1 := \frac{1}{4\tau_2}$$

, where τ_2 is the global bound on mesh quality throughout the algorithm.

Let x be the first vertex that is inserted into any mesh M such that $|xv| < \varepsilon_1 g(v)$. Consider the mesh M^- and M^+ immediately before and after the insertion of x.

If $x = v$, then the collar size is called immediately before v is inserted, and since v must have been $(1-\theta)$-medial. Then by choice of x, it must be that $n_{M^-}(v) > \varepsilon_1 g(v) > \varepsilon_0 g(v)$, so the case is trivial. Assume $x \neq v$ for the remainder of the proof.

Clearly x must have higher containment dimension than v for it to be within $B(v, \varepsilon_1 g(v))$ and must be on some superfeature of v's parent feature. Thus, when x was created as the center of some gap-ball $B(x, 2r_x)$, it had a chance to warp to v, but did not. Thus

$$|xv| > 2\theta r_x \tag{A.1}$$

but then

$$\theta 2r_x < \varepsilon_1 g(v) \tag{A.2}$$

Since the mesh is always τ_2 quality, $R_x < \tau_2 r_x$, so then $R_x < 2\tau_2\varepsilon_1 g(v)$. But then since $v \in B(x, R_x)$ by design, the choice of VE_1 gives isolation in M^+. by Lemma 1.

Since v is isolated in M^+, I now claim that v is θ_0-medial wrt to M^+.

Since x was the center of a gap-ball, there was some vertex b of M^- on the surface of the ball. Then by (A.1):

$$|bv| \leq |xv| + r_x \leq (1 + \frac{1}{2\theta})|xv|$$

Since x and b are both in M^+, we have $n_{M^+}(v) = |vx|$ and $f_{M^+}(v) \leq |bv|$, but then this yields:

$$n(v) \geq \frac{2\theta}{2\theta + 1} f(v) = \theta_0 f(v)$$

So v is θ_0-medial.

Thus CREATECOLLAR will be invoked after the insertion of x (if not already at some previous iteration). It remains to show the lower bound on $n(v)$ at this iteration.

Take b as before, and take:

$$\varepsilon_0 := \frac{\theta\varepsilon_1}{\theta + 1}$$

Note since $\theta \in (0, 1)$, then $0 > \varepsilon_0 > \varepsilon_1$. Since x was the first vertex closer than $\varepsilon_1 g(v)$, then $|bv|$ must have been larger than this distance. Consider then, using (A.1) as well:

$$\varepsilon_1 g(v) < |bv| \leq |xv| + |xb| = |xv| + 2r_x < (1 + \frac{1}{\theta})|xv|$$

Bringing the constant over, then:

$$n_{M^+}(v) = |xv| > \frac{\theta\varepsilon_1}{\theta+1} g(v) = \varepsilon_0 g(v)$$

Over the life of the algorithm, the nearest neighbor distance n only decreases. So if CREATECOLLAR was called at some earlier iteration before the insertion of x, then the lower bound on n still holds. □

The previous lemma guaranteed that the vertices on creases are resolved early enough. A technicality is to make sure that the rest of the crease (along segments in particular) is protected as well. Define the ε-collar-region as the region given by the union of balls $B(x, \varepsilon g(x))$ taken over all x on the creases. I will show this region is protected for some constant ε_2.

Lemma 9 (Crease Protection) *There exists a constant ε_2, such that no* UNRESOLVED *work event is ever enqueued in the ε_2-collar-region for a vertex not on a crease.*

Proof: Proof is by contradiction, and will rely on the lower-bound on size at crease recovery from the Crease Recovery Lemma 2. Take ε_0 as in the proof of Lemma 2 Take $\varepsilon_2 := \varepsilon_0/3 < \varepsilon_0/(2+\varepsilon_0) < \varepsilon_0$.

Suppose some vertex v has an UNRESOLVED work event enqueued and v is in the ε_2-collar-region but not on a crease.

Suppose v near some created vertex on crease u, i.e. $v \in B(u, \varepsilon_2 g(u))$, but since $\varepsilon_2 < \varepsilon_0$, then by the Crease Recovery Lemma, CREATECOLLAR has been called on u. Then since $v \in B(u, g(u)/3)$, then v is interior to the collar system and would not have been enqueued.

So it must be the case that v is near some crease subsegment but far from its vertices. Based on its relative distance to the segment and the definition of g, it is clear that v must be on some superfeature of the segment. But then v would not be inserted if it encroached on this segment. Suppose a is an endvertex of this segment, let R and c be the center radius of the diametral protective ball. Without loss of generality, assume that v is close to c, so that $|vc| < \varepsilon_2 g(v)$. Since v does not encroach, and g is Lipschitz along the segment, then:

$$R < |vc| < \varepsilon_2 g(c) \leq \varepsilon_2 g(a) + \varepsilon_2 R$$

Since the other end of the segment is not too close to a by Lemma 2, it holds that $\varepsilon_0 g(a) < 2R$, so this gives:

$$R < \varepsilon_2 g(a) + \varepsilon_2 R < \frac{2\varepsilon_2}{\varepsilon_0} R + \varepsilon_2 R < \left(\frac{2\varepsilon_2}{\varepsilon_0} + \varepsilon_2\right) R$$

But then the choice of ε_2 makes the right side constant less than one, so this is a contradiction. □

B Packing Arguments

In this section, I describe a straightforward packing lemma that will be used several times. This lemma bounds the size of a sequence of disjoint events around a central origin, and will be main tool for proving bounds of the form $O(\log \Delta)$.

Definition 14 (Proximal Event Sequence) *Define an* **event** *as a pair consisting of a ball and a vertex, denoted $\langle B(u,r), v\rangle$. An* **event sequence** *U around v is an ordered set of events $\langle B(u_i, r_i), v\rangle \in U$ with the following interior disjointness property:*

$$\forall\, i, \forall\, j < i,\ u_j \notin B^\circ(u_i, r_i)$$

so that each event point u has a ball around it whose interior is empty of previous event points.
*An event sequence is a ν-***proximal event sequence** *if there exists $\nu \in (0, \infty)$ such that:*

$$\forall\, i,\ r_i \geq \nu |u_i v|$$

I now prove a bound on the size of a proximal event sequence around a vertex. Statements of this form appear in many works, including [HMP06]. The framework here is written to suit the needs of this thesis.

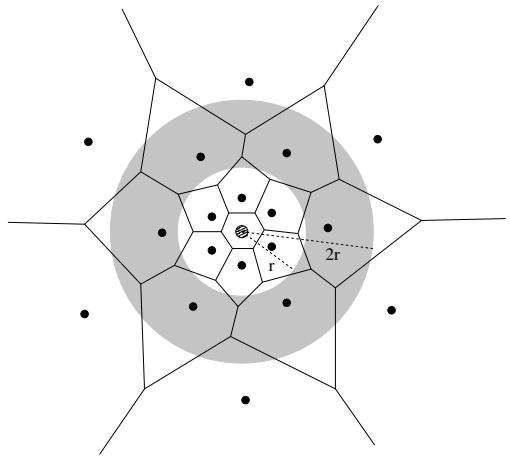

Figure 5: An illustration showing the intuition behind Lemma 10 and its later uses. A quality Voronoi diagram packs vertices around the small central feature. There are only a constant number of vertices in each radius-doubling annulus (shown in gray).

Lemma 10 (Proximal Event Packing) *Suppose U is a ν-proximal event sequence around v with distances from v bounded by $r \leq |u_i v| < R$, then:*

$$|U| \lesssim \log(R/r)$$

When $R/r \lesssim \Delta$, this gives:

$$|U| \lesssim \log \Delta$$

If all the events are at roughly the same distance d, i.e. $|u_i v| \sim d$, then $R/r \lesssim 1$ so that:

$$|U| \lesssim 1$$

Proof: The proof is by packing events into a sequence a of size-doubling annuli around v. Define the kth annulus around v as the difference of two balls given by $A_k := B(v, 2^{k+1}r) - B(v, 2^k r)$. Partition the events into sets $U_k := \{\langle B(u_i, r_i), v\rangle \mid u_i \in A_k\}$.

All of the U_k are empty when $k > \log(R/r)$. To finish the lemma, I claim that $U_k \lesssim 1$ for all the non-empty annuli.

Consider the set U_k for some fixed k, and consider any pair of events $i < j \in U_k$. By disjointness of the sequence, $u_i \notin B^\circ(u_j, r_j)$, so there are two smaller balls $B(u_i, r_j/2)$ and $B(u_j, r_j/2)$ that are disjoint. Letting r_k be the smallest radius of any event in U_k, there is a family of disjoint balls $\mathcal{B}_k := \cup_i B(u_i, r_k/2)$.

The small radius r_k was the radius of some event j so it follows directly from assumptions that $r_k \geq \nu|u_j v| \geq \nu 2^k r$. If $r_k > 2^k r$, then shrink the balls in $\mathcal{B}_k$ further by reducing $r_k := 2^k r$. This now guarantees that $r_k \sim 2^k r$. Then each ball in $\mathcal{B}_k$ has $Vol(B) \sim 2^{3k} r^3$. Since the balls are disjoint, the volume of $\mathcal{B}_k$ is found by summing to get $Vol(\mathcal{B}_k) \sim |U_k| 2^{3k} r^3$, with constant depending on ν.

This centers are in the annulus, but the entire balls may not be, so consider a fattening of the annulus A_k by an additive factor of r_k, to define:

$$\bar{A}_k := B(v, 2^{k+1}r + r_k) - B(v, \max(0, 2^{\mathrm{k}}\mathrm{r} - \mathrm{r_k}))$$

so that $\mathcal{B}_k \subset \bar{A}_k$. Since $r_k \sim r$, this fattening does not much change the size of the annulus, so it follows from simple geometry that $\bar{A}_k$ has volume $Vol(\bar{A}_k) \sim 2^{3k} r^3$.

All of the balls in B_k are completely contained in $\bar{A}_k$, so $Vol(B_k) \leq Vol(\bar{A}_k)$. Summarizing:

$$Vol(\bar{A}_k) \sim 2^{3k} r^3 \sim \frac{Vol(B_k)}{|U_k|} \leq \frac{Vol(\bar{A}_k)}{|U_k|}$$

thus $|U_k| \lesssim 1$, with constant depending only on ν. $\square$

In Section D this lemma is utilized to show that all of the nonlinear point-location work can be amortized as one proximal event sequences around each input feature. The analysis of SVRC also uses this lemma to bound the work used to calculate the size of the collar region.

C Collar Sizing Runtime

The calls to method FASTCOLLARSIZE generate work to perform an exploration of the local neighborhood around some vertex v in order to calculate $g(v)$.

The efficiency of this procedure is what causes the need for approximately largest first work queues. When the queues are processed largest first, the geometric sizing of the mesh will decrease in somewhat of a breadth-first fashion. If there are two disjoint regions that both require refining, then the largest-first order will roughly alternately refine them, rather than refining one first and then the other. This will be captured in the proofs by observing that in this case, the outstanding UNRESOLVED work events will always be proportional to each other, and all the other cells in the mesh will be larger than this. The following lemma encodes this notion:

Lemma 11 (Same Size Lemma) *Consider the mesh M at some point during a run of SVRC. Let R be the outradius of the largest* UNRESOLVED *work event. Then for every vertex $v \in M$, $R_v \gtrsim R$.*

Proof: First, I claim that every UNRESOLVED work event so far has had radius larger than R. Suppose v is the unresolved feature causing the current largest work event. Any previous cell containing v must have had a larger radius, and so whatever UNRESOLVED work events have been done must have been even larger.

Since there is an outstanding UNRESOLVED event, then no COLLAR moves have been performed, so then UNRESOLVED events represent the only base case for the Weak Spacing proof. Letting R' be the radius of the smallest unresolved event performed, a simple revisiting of the Weak Spacing Proof then gives $n_M \gtrsim R'$. Alwasy

But since $R' > R$, then $n_M \gtrsim R$. The proof is then finished since it is always the case that $R_v \gtrsim n_M(v)$. □

Returning to the collar exploration, the procedure explores a local patch of the mesh looking for a witness to the function g. All of the work looking at witnesses will be accounted for in the next section. It is necessary to account for all the cells explored, in case there are many cells but few witnesses.

Lemma 12 (Fast Collar Exploration) *Suppose* FASTCOLLARSIZE *is called at some vertex v. Then the exploration to find $g(v)$ explores at most constantly many Voronoi cells of M.*

Proof: Recall that v may or may not have just been inserted. Let M be the current mesh, unless if v was just inserted, then let M be the mesh just prior to inserting v.

Let $q \in M$ such that $v \in V(q)$. Since $q \neq v$, then $V(q)$ must be enqueued as an UNRESOLVED move. Let R be the outradious of the largest UNRESOLVED move, then $R \geq R_q$. But by the Same Size Lemma, every cell $V(u)$ in M has $R_u \gtrsim R \geq R_q$.

By the Crease Recovery Lemma, there is an empty ball around v of radius $\varepsilon_0 g(v)$, but this then implies that $|vq| \geq \varepsilon_0 g(v)$, so then $R_q \geq VE_0 g(v)$.

But this implies that every cell in the mesh is large with respect to $g(v)$, and since the cells are all τ_2-well-spaced, then only constantly many cells will intersect the ball $B(v, g(v))$. The breadth-first search will only try cells intersecting this ball before it finds a witness to $g(v)$, so the lemma is proved. □

D Amortized Work Operations

The operations in SVRC related to point-location will be accounted in this section. Recall that every call to DESTROY results in a mesh insertion near the center of some gap-ball. Before this vertex is inserted, SVRC makes a query to an intersection graph used for Point Location to determine if there are any nearby location objects (protective balls and collars), either for warping, or for yielding to first perform a lower dimensional insertion. After an insertion eventually occurs, the BLG must be updated to reflect the newly changed mesh. Additionally, the flags for isolation and θ-medial must be updated for any UNRESOLVED events. The BLG is also queried when CREATECOLLAR is called to determine the sizing $g(v)$ for a new collar centered at v. Call v a **target** when either v is being inserted or v is the center of a collar sizing query.

All of the work will occur in a relatively local neighborhood of the target, and so we account for all this work as a set of **location work events** (LW-Events), each of which is a pair $\langle v, O \rangle$, where v is a target, and O is a location object of the BLG. There may be more than one LW-Event for a given pair, but I claim that the number of repeated events is constant per pair.

Lemma 13 (Unique LW-Events) *The total number LW-Events is at most a constant times the number of unique pairs $\langle v, O\rangle$ for which there exists an LW-Event.*

Proof: The lemma follows by simple inspection of the algorithm. LW-Events occur in FORCEINSERT, UPDATELOCATION, PROPAGATELOCATION, CREATECOLLAR, and CREATECOLLAR. Each of these methods is only ever called once with a given target. Furthermore, any pair $\langle v, O\rangle$ will cause at most constantly many LW-Events in each method call. Thus, the total LW-Events for any given pair is constant, so it suffices to count the unique pairs. □

Unfortunately, the number of LW-Events associated with a single target can be very large. Consider the first few mesh insertions, which may scan the entire input to update the BLG. The key argument is an amortization to count LW-Events per object, instead of per target. Let W be the set of all unique LW-Events. Given W and a single location object O, define the set of all LW-Events associated with O as W_O; simply those pairs which include O.

Some location objects may cause many LW-Events. Consider an UNRESOLVED work event for some input vertex that is not resolved until the mesh reaches very fine resolution. This vertex will be checked for relocation many times as the mesh insertions decrease the size of whatever current Voronoi cell contains the vertex. In this case, I will show that the size of $|W_O| \lesssim \log \Delta$ if O is part of the input. On the other hand, consider a REPBALL around some subsegment that is created at some time during the algorithm. Either this ball is part of the output, or soon after, this ball will be encroached. If it is encroached, then its center is inserted, it is destroyed, and two new balls are created. Either way, the lifetime of this object will be short enough that I will bound $|W_O| \lesssim 1$ for any created objects. Putting these together will give the runtime bound desired:

$$\begin{aligned} |W| = \sum_O |W_O| &= \sum_{O \in \text{Input}} |W_O| + \sum_{O \in \text{Created}} |W_O| \\ &\lesssim \sum_{O \in \text{Input}} \log \Delta + \sum_{O \in \text{Created}} 1 \\ &\leq N \log \Delta + |M| \end{aligned}$$

D.1 Counting Location Work

This sections recalls structural properties of quality Voronoi diagrams. Recall that every mesh during SVR is τ_2-well-spaced for some τ_2. This section will separate LW-Events into two cases based on their target.

Consider first the LW-Events where the target is a vertex v being inserted. Vertex v was inserted because of some gap-ball $B := B(c, R)$. The location objects that v will pair with will be any object intersecting B, as well as any object intersecting a Voronoi cell that is modified by inserting v.

The first lemma states that the such LW-Event pairs are not relatively far apart:

Lemma 14 *Let v, B, c and R, as in the preceding paragraph. Let O be an object such that $\langle v, O\rangle$ is a LW-Event, then:*

$$|vO| \lesssim R$$

Proof: Let M^- and M^+ be the mesh immediately before and after the insertion of v Since c is a Voronoi corner of M^-, let q be a vertex whose corner it is, so that $R = R_q(M^-)$.

Suppose first that O intersects some cell $V(u)$ that happened to intersect the gap-ball B. Let U be the set of all such u. By the Bounded-Ply Theorem [MTTW95], since all these cells intersect a gap-ball, then $|U| \lesssim 1$. But then by Lipschitz arguments, neighbors in a well-spaced mesh are sized similarly, so $R_u \lesssim R_q = R$ and $|uq| \lesssim R$ for every u. But then using triangle inequality, and since O intersected $V(u)$:

$$|vO| \leq |vc| + |cq| + |qu| + |uO| \leq R + R + |qu| + R_u \lesssim R$$

Now suppose instead that O intersected some cell $V_{M^-}(u)$ which is then modified by inserting v. It must then that u and v are adjacent in M^+. Also, in the new mesh M^+, either O intersects $V_{M^+}(v)$ or $V_{M^+}(u)$. If it is the former, then using $r_v(M^+) \leq |vq|/2 \leq R$:

$$|vO| \leq R_v \leq \tau_2 r_v \leq \tau_2 R \lesssim R$$

If it is the latter, then since M^+ is well-spaced:

$$|vO| \leq |vu| + |uO| \leq 2R_v + R_u \leq 2R_v + \tau_2 R_v = (2+\tau_2)R_v \leq \tau_2(2+\tau_2)R \lesssim R$$

□

Since they are close together, then packing arguments developed in Section B will apply. Recalling Definition 14 for a Proximal Event Sequence, where each event (in this case a target) has an empty ball around it lower bounded by the distance to the center (in this case the location object):

Lemma 15 (Insertion LW-Events form a sequence) *Consider any location object O and consider the set W_O of all the LW-Events $\langle v, O\rangle$ for which the target v is a vertex being inserted. Then W_O forms a Proximal Event Sequence around the O.*

Proof: Let some $\langle v, O\rangle$ be given. Let M^- be the mesh immediately before inserting v, and let $B := B(c, R)$ be the gap-ball being destroyed by v. Then v must have been near c, i.e. $|vc| \leq \theta R$. But since B was a gap-ball, this means that $n_{M^-}(v) \geq (1-\theta)R$. Let $B_v := B(v, (1-\theta)R)$ be the associated empty ball centered at v.

This ball is empty of any previously inserted vertices, so the sequence of pairs $\langle B_v, v\rangle$ form an event sequence.

But then by the preceding Lemma 14, it follows:

$$|vO| \lesssim R \lesssim (1-\theta)R$$

So this must be a proximal event sequence around O. □

A similar pair of lemmas will establish the same result for LW-Events whose target is performing a collar sizing query.

Lemma 16 *Suppose $\langle v, O\rangle$ is an LW-Event where v is the argument to* CREATECOLLAR*, then $|vO| \lesssim g(v)$*

Proof: Recall from the code (Figure 4) that `SVRC` performs a breadth-first search around v until it finds a witness for $g(v)$.

Let q be the Voronoi cell containing v, i.e. $v \in V(q)$, noting that it may be that $v = q$ if v was just inserted. This code is only called when v is isolated, therefore $R_q \leq g(v)/2$ by Lemma 1.

Suppose some cell $V(u)$ is explored. If $u = q$, then simply $|vO| \leq |vq| + |qO| \leq 2R_q \leq g(v)$. Suppose $u \neq q$. Then it must be the case that $V(u) \cap B(v, g(v)) \neq \emptyset$ otherwise this would contradict breadth-first; `SVRC` would first have explored all the cells intersecting $B(v, g(v))$ and would have found a witness and stopped. Let $x \in V(u) \cap B(v, g(v))$, then since u is the nearest-neighbor of x, $V(u)$ cannot be too large, i.e.:

$$R_u \leq |ux| \leq |xq| \leq |xv| + |vq| \leq g(v) + R_q \leq 1.5g(v)$$

Continuing then:

$$|vO| \leq |vx| + |xu| + |uO| \leq g(v) + 2R_u \lesssim g(v)$$

□

Continuing as in the other target case:

Lemma 17 (Collar Sizing LW-Events form a sequence) *Consider any location object O and consider the set W_O of all the LW-Events $\langle v, O\rangle$ for which the target v is a vertex argument to* CREATECOLLAR*. Then W_O forms a Proximal Event Sequence around the O.*

Proof: Let some $\langle v, O\rangle$ be given. By the Crease Recovery Lemma 2, there is an empty ball around v with radius at least $\varepsilon_0 g(v)$ for some fixed ε_0 when CREATECOLLAR is called.

These balls are disjoint from any previously inserted vertices, so the pairs $\langle B(v, \varepsilon_0 g(v)), v\langle$ form an event sequence. But then by the preceding Lemma 16, it immediately follows that they form a proximal event sequence around O. □

E Counting the Proximal Event Sequences

The previous section established that the LW-Events form two proximal event sequences around the location objects, one for insertion targets and one for collar targets. From here on, it suffices to consider one proximal event sequence around each location object, with the understanding that the total number of LW-Events is then underestimated by at most a factor of two.

Lemma 18 (Bound on LW-Events) *Consider a location object O, and let M_O be the LW-Events associated with O. If O is a created object, then $|M_O| \lesssim 1$. If O is an input object, then $|M_O| \lesssim \log \Delta$.*

Proof: Let v be the target of the LW-Event which minimizes $|vO|$, with the exception that if O is a protective ball and v encroaches it, this should not be included. This excludes at most two LW-Events per object, since the ball would always be destroyed after it is doubly-encroached. If the location object is a ball $B(c, R)$, then this guarantees that $|vc| > R$.

Suppose O is a created object, then it is always the case that O is not too small, i.e. $R \gtrsim f_M(c)$ for every M containing O, which then implies that $|vc| \lesssim R$. But then all the LW-Events are at distance proportional to R, so the second statement of the Proximal Packing Lemma 10, so $|W_O| \lesssim 1$.

Suppose O is an input object. Then the farthest LW-Event cannot be larger than the diameter of the PLC. The nearest event will be no closer than the nearest neighbor in the final output. By termination arguments this is bigger than the output feature sizew which is bigger than the smallest feature of the clipped PLC. Thus the ratio for farthest to nearest LW-Events is at most a constant times Δ, so by the Proximal Packing Lemma, there are at most $|W_O| \lesssim \log \Delta$. □

Corollary 1 (Total Location Work) *The total number of Location Work Events W is bounded by:*

$$|W| \lesssim N + |M| \log \Delta$$