Eurographics Symposium on Geometry Processing 2011
Mario Botsch and Scott Schaefer
(Guest Editors)

Volume 30 (2011), Number 5

Localized Delaunay Refinement for Volumes

Tamal K Dey[†1] and Andrew G Slatton[‡1]

[1]The Ohio State University, Computer Science and Engineering

Abstract

Delaunay refinement, recognized as a versatile tool for meshing a variety of geometries, has the deficiency that it does not scale well with increasing mesh size. The bottleneck can be traced down to the memory usage of 3D Delaunay triangulations. Recently an approach has been suggested to tackle this problem for the specific case of smooth surfaces by subdividing the sample set in an octree and then refining each subset individually while ensuring termination *and* consistency. *We extend this to localized refinement of volumes, which brings about some new challenges. We show how these challenges can be met with simple steps while retaining provable guarantees, and that our algorithm scales many folds better than a state-of-the-art meshing tool provided by CGAL.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, Surface, Solid, and Object Representations

1. Introduction

Delaunay refinement is accomplished by iteratively sampling an input geometry at points locally farthest from the current sample set whenever a condition is violated [Che89, Rup95]. It has been recognized as an effective tool for sampling and meshing a variety of geometries including polyhedra [She98], smooth surfaces [BO05, CDRR07] and volumes [ORY05], piecewise smooth surfaces [BO06] and complexes [CDR08]. The popularity of the technique can be attributed to its ability to equip algorithms with provable guarantees and also to generate good quality meshes in conjunction with optimizations [ACSYD05, TWAD09].

One shortcoming of such an otherwise excellent tool is its lack of scalability. Since traditional Delaunay refinement maintains a complete three dimensional Delaunay triangulation of a growing sample, its time and memory usages are determined by those of the Delaunay triangulation algorithm being employed. Although considerable progress has been made to speed up Delaunay triangulations [ACR03,ILSS06], still the state-of-the art does not scale well for computing 3D meshes with simplices in the range of a few million. Parallel algorithms could be a solution to this problem, but to date they are limited only to Delaunay refinement of polyhedra [NCC04].

Recognizing the gap, very recently Dey, Levine, and Slatton [DLS10] proposed an algorithm that can sample and mesh a smooth *surface* using *localized* Delaunay refinement. The localization is obtained by maintaining the current sample in an octree and then refining each leaf node separately. The obvious problems this solution faces are how to maintain a lower bound on inter-point distances over the global set and how to fit individual meshes consistently. Dey et al. showed that these two concerns can be alleviated by careful point insertion and a re-processing technique, while ensuring quality guarantees. The algorithm improved the memory usage by an order of magnitude and therefore prevented memory thrashing–a culprit of scaling.

In this paper, we extend the technique of Dey et al. [DLS10] to volumes bounded by smooth surfaces. Since volume meshing with quality tetrahedra is an important requirement for many scientific simulations, extension of a successful technique for surface meshing to this domain is an important undertaking. It turns out that this extension from surface to volume is non-trivial, bringing new challenges and requiring additional observations and results. In this discourse, we reveal theoretical results which guarantee some qualities for the output, as well as practical results which show that our method surpasses the performance of a CGAL [cga] volume meshing tool by many folds; these are

[†] e-mail: dey.8@osu.edu
[‡] e-mail: slatton.2@buckeyemail.osu.edu

the two main contributions of this investigation. It should be noted that our method does not address sliver removal, a nontrivial problem commonly faced by Delaunay refinements in general.

1.1. Definitions and Notations

Our algorithm and the analysis thereof make use of Voronoi/Delaunay tessellations and their restrictions to a volume O and its boundary $\partial\mathsf{O}$, as well as the idea of "locally farthest points". For completeness, we recapitulate the definitions of these here.

For a set of points $P \subset \mathbb{R}^3$, we denote its Voronoi diagram and Delaunay triangulation as $\mathrm{Vor}\,P$ and $\mathrm{Del}\,P$ respectively. Each k-dimensional Delaunay simplex σ (vertex, edge, triangle, and tetrahedron) is dual to a $(3-k)$-dimensional Voronoi face V_σ (cell, facet, edge, vertex respectively).

The point set P in our case will be from a volume O bounded by a smooth boundary $\partial\mathsf{O}$ in $\mathbb{R}^3$ often approximated by a polygonal surface in practice. The set of Delaunay simplices whose dual Voronoi faces intersect O or $\partial\mathsf{O}$ will be of special interest to us. We use special subcomplexes of $\mathrm{Del}\,P$ called the *restricted Delaunay complexes* with respect to O and $\partial\mathsf{O}$. They are defined as:

$$\mathrm{Del}\,P|_{\mathsf{O}} = \{\sigma \in \mathrm{Del}\,P : V_\sigma \cap \mathsf{O} \neq \varnothing\}$$
$$\mathrm{Del}\,P|_{\partial\mathsf{O}} = \{\sigma \in \mathrm{Del}\,P : V_\sigma \cap \partial\mathsf{O} \neq \varnothing\}.$$

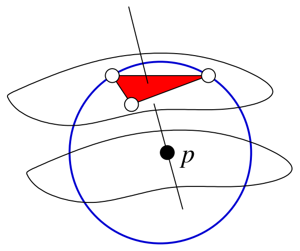

Figure 1: *A restricted triangle in $\mathrm{Del}\,P|_{\partial\mathsf{O}}$: its dual Voronoi edge intersects the surface. One such intersection point is p which is the center of a surface Delaunay ball of the triangle.*

The complex $\mathrm{Del}\,P|_{\mathsf{O}}$ consists of tetrahedra, triangles, edges, and vertices whose dual Voronoi faces intersect O. Similarly, $\mathrm{Del}\,P|_{\partial\mathsf{O}}$ consists of triangles, edges, and vertices whose duals intersect $\partial\mathsf{O}$. The dual Voronoi edge of a restricted Delaunay triangle f may intersect $\partial\mathsf{O}$ at multiple points. Each of these intersection points is the center of a three dimensional ball that does not contain any point of P inside, but contains the vertices of f on its boundary (Figure 1). They are called the *surface Delaunay balls* of f.

The locally farthest point in a Delaunay refinement is a point where a Voronoi face such as a Voronoi edge or a Voronoi vertex intersects the input geometry.

2. Overview

Usually, the Delaunay refinement is accomplished by maintaining the Delaunay triangulation of the entire sample set in memory because any refinement based on proper subsets of the sample incurs two questions, namely:

1. How can one ensure that the meshes of these subsets fit together consistently?
2. How can one guarantee termination of such an algorithm?

The first of these arises because, given two arbitrary subsets of a sample, some points in one may lie inside some Delaunay balls of the other; therefore, some simplices that are restricted with respect to the subset may then not be restricted (or may not exist at all) with respect to the entire point set. The solution to this problem lies in choosing subsets carefully, and appropriately selecting from the triangulations of these subsets those simplices which will comprise the final output.

The second problem arises because an added point that is locally farthest in a subsample may not remain so in the full sample. This means that there is no lower bound on inter-point distances within the sample, and therefore no upper bound on the number of point-insertions. This problem is solved by a point-insertion scheme that can guarantee a lower bound on inter-point distances.

In this paper, we draw upon the approach of Dey et al. [DLS10] to mesh the volume O bounded by a smooth 2-manifold input $\partial\mathsf{O}$, a problem for which the questions and general solutions mentioned above remain valid. However, there are additional difficulties that arise during volume meshing – one pertaining to termination and another to the topology of the output.

We prove termination via a packing argument; therefore we must show that we only insert points inside some bounded domain. This can be troublesome when we refine tetrahedra because, though O is bounded, the dual Voronoi vertex of an arbitrarily chosen Delaunay tetrahedron may not lie in O. So, we must take some measures to ensure that we refine only those tetrahedra whose dual Voronoi vertices lie in O. We discuss our solution to this problem at the end of our algorithm description and in our argument for termination.

We aim to guarantee the topological equivalence of the output mesh to the input volume. This assurance requires all restricted triangles to have all their vertices on $\partial\mathsf{O}$. To meet this condition, Oudot et al. [ORY05] force a triangle refinement, and we follow suit; however, this strategy faces the following problem in localized refinement: a Voronoi vertex may be inserted arbitrarily close to $\partial\mathsf{O}$ as it is not completely protected by surface Delaunay balls in the partial set;

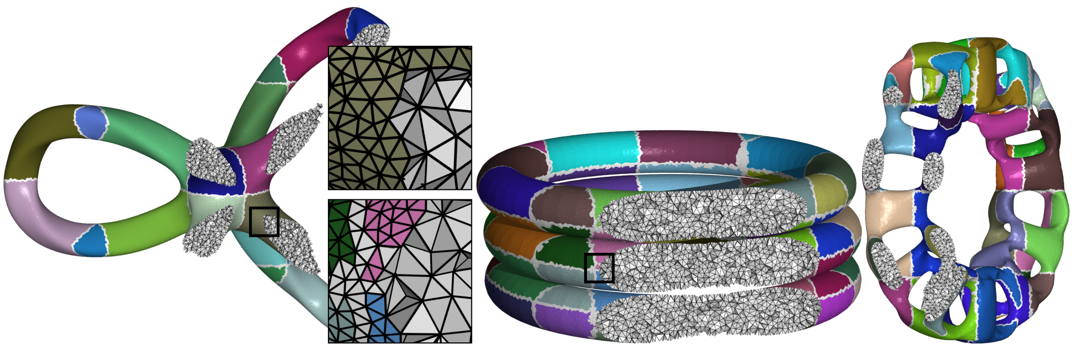

Figure 2: *Cut away views show some tetrahedra in meshes produced by our algorithm. Surface colors signify decomposition with octree leaf nodes.*

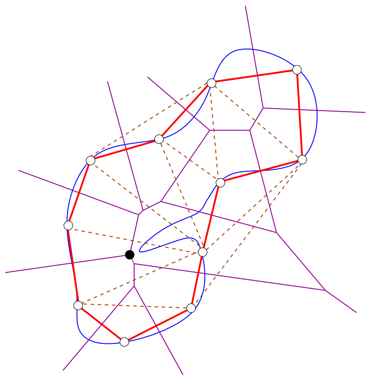

Figure 3: *The black Voronoi vertex, if inserted as a circumcenter, would be very close to the curve (surface). Even though the restricted Delaunay edges (triangles) with samples on the curve (surface) form a manifold, the curve (surface) near the vertex may not yet be meshed properly to prevent volume triangulation introducing the Voronoi vertex in a local refinement.*

see Figure 3 for an illustration. Then to get rid of restricted triangles incident to such an interior vertex the refinement could be arbitrarily dense, jeopardizing termination. Our solution: when a surface vertex is inserted, all volume vertices close to it are deleted. Our proof of termination leverages this action.

We prove that the algorithm terminates, outputting a 3D mesh which is always a manifold and is close to the input volume with respect to λ, a user supplied parameter. The guarantee remains valid no matter what value of the parameter λ is supplied. Furthermore, when λ is sufficiently small, the output becomes isotopic to the input volume.

3. Algorithm

Our algorithm computes and refines the surface and volume mesh simultaneously. It divides the sample set P using an octree, and processes only one leaf node of the octree at a time. During the processing of a node v, six conditions are checked. When one of these conditions is violated, the algorithm refines the node's local triangulation accordingly. When there are no more violations in v, it begins processing another node of the octree. When none of the refinement criteria are violated in any of the nodes, a final output is generated. Throughout the algorithm, we utilize a point insertion strategy that is necessary for our proof of termination, and we may select some nodes for *reprocessing* in order to maintain a global consistency across meshes. Details follow.

3.1. Node Processing

The nodes of the octree requiring processing are maintained in a queue Q, and each node is processed when it reaches the head of Q. A node may be processed by one of two actions: split or refine. Each node v of the octree maintains a set of points $P_v = P \cap v$. When the number of points in v exceeds a user-defined parameter κ, that is, $|P_v| > \kappa$, we invoke a split; if $|P_v| \leq \kappa$ when v reaches the head of Q, we invoke a refine.

In a split, v is divided into eight children of equal size, each of which is geometrically similar to v. The points of P_v are then divided among these children, with each child taking the points that lie within its volume, and then these children are enqueued in Q.

When a node v is refined, we begin by computing its local triangulation Del R_v, where R_v is a *superset* of P_v. Specifically, we initialize $R_v := N_v \bigcup P_v$, where $N_v \subseteq P$ contains the points of P that lie within a distance 2λ of the boundary of v (Figure 4); the value of 2λ is chosen in order to prove our claims regarding the output. While violations of our refinement criteria persist, we refine the local triangulation of v. Detailed descriptions of the refinement criteria are given

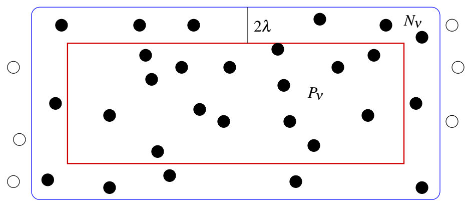

Figure 4: $R_v := N_v \bigcup P_v$. *The solid rectangle shows the boundary of node* v. *Solid points are in the initial* R_v*; hollow points are not.*

later. If $|P_v| > \kappa$ at any point during the refinement of the local triangulation, we invoke a split of v.

When a node v is not being processed we clear its local Delaunay triangulation (along with all the data structures associated with it) in order to save memory, maintaining only its $P_v \subseteq P$ and a list of restricted tetrahedra inside v. The list of tetrahedra can be stored on disk to reduce memory footprint, as it will not be required again until the output step; however, P_v should be kept in memory because it may be accessed often by our point-insertion method.

3.2. Localized Refinement

For each point p in a node v, we want the local triangulation around p be nice, that is, surface triangles around p form a topological disk, and tetrahedra around p form a topological ball. Our ultimate goal is to fit all these individual local triangulations seamlessly in a global one. Toward that goal, we define the *surface star* F_p of a point $p \in P_v$ as the set of triangles incident on p that are restricted to ∂O in the local triangulation, and the sub-simplices of all such triangles. Formally,

$$F_p = \{f \,|\, f \in \mathrm{Del}\, R_v|_{\partial O} \text{ is either a triangle incident to } p \text{ or a sub-simplex of such a triangle}\}.$$

Similarly, we define the *volume star* T_p of a point $p \in P_v$ as the set of tetrahedra incident on p that are restricted to O in the local triangulation, and the sub-simplices of all such tetrahedra. Formally,

$$T_p = \{t \,|\, t \in \mathrm{Del}\, R_v|_O \text{ is either a tetrahedron incident to } p \text{ or a sub-simplex of such a tetrahedron}\}.$$

3.2.1. Refinement

During the refinement phase we check six conditions in a *priority order* which gives priority to CI over CJ if I<J:

1. (C1) $\forall f \in F_p$, where f is a triangle and $p \in P_v$, the surface Delaunay ball of f has radius less than λ;
2. (C2) $\forall f \in F_p$, where f is a triangle and $p \in P_v$, all vertices of f lie on ∂O
3. (C3) $\forall p \in P_v$, if $p \bigcap \partial O = p$ then F_p is a topological disk with each edge $e \in F_p$ that is incident on p being also incident on exactly two triangles in F_p;
4. (C4) $\forall f \in F_p$, where f is a triangle and $p \in R_v$, the Voronoi edge dual to f intersects ∂O only once;
5. (C5) $\forall t \in T_p$, where t is a tetrahedron and $p \in P_v$, t has circumradius less than λ;
6. (C6) $\forall t \in T_p$, where t is a tetrahedron and $p \in P_v$, the radius-edge ratio is at most 2.

In the event of a violation, a tuple (p,q) is returned, where q is a candidate for insertion and p is the closest point to q in R_v. If C1, C2, or C4 is violated, q is the center of the largest surface Delaunay ball of the culprit f. In the case of C3, q is the center of the largest surface Delaunay ball of all triangles in F_p. For violations of C5, q is the circumcenter of the culprit t. For violations of C6, let c be the circumcenter of the culprit t. If c lies inside some surface Delaunay ball, then q is the center of that surface Delaunay ball; otherwise, $q := c$. When no violations remain, we return a null tuple.

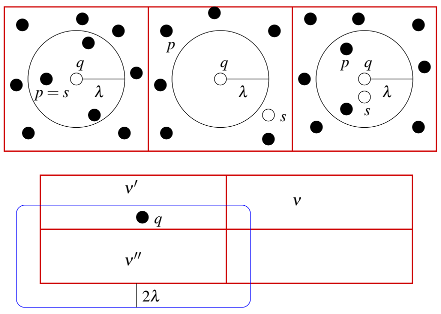

Figure 5: *(top) shows some examples of point insertion (solid points are in* R_v*; hollow points are not). In the rightmost image,* s *is added to* R_v*; in the other two images* q *is added to both* R_v *and* P*. (bottom) depicts an example of reprocessing: when* v *inserts* q*,* v' *and* v'' *are enqueued for reprocessing.*

3.3. Point Insertion and Deletion

Algorithm 1 InsertDelete(v,p,q,λ)

```
1: $s := \mathrm{argmin}_{u \in P}\, d(q,u)$
2: if $d(q,s) \le \lambda$ and $s \notin R_v$ then
3:     $R_v := R_v \cup \{s\}$; return $s$
4: else
5:     if $q \in \partial O$ then
6:         delete any $p \in P \setminus \partial O$ with $d(p,q) \le \lambda$
7:     end if
8:     $P := P \cup \{q\}$; update $R_v$; return $q$
9: end if
```

Let (p,q) be a tuple returned by some violation during refinement. Then q is a candidate for insertion, but it may lie arbitrarily close to some point in P despite being locally far in R_ν. In order to disallow arbitrarily close insertions in our sample, we find the closest point $s \in P$ to q. If s lies within distance λ of q, and $s \notin R_\nu$ (recall that p is the nearest point to q in R_ν), then we throw away q and insert s into R_ν; otherwise, we insert q into R_ν and add it to P. Note that R_ν is augmented in both cases, and P is augmented in only the latter case. Figure 5(top) illustrates different cases.

Sometimes we also require point deletions in response to the insertion of a *surface* point, that is, a point in ∂O. Deleted points are always *volume* points, that is, points that are in P but not in ∂O. If the new point q is a surface point, we delete all volume points that are within λ distance of q. We remark that deletions happen rarely in practice and only at the beginning when the surface ∂O is not yet completely covered with surface Delaunay balls.

Furthermore, when a new point is added to or deleted from P it is possible that some nodes are enqueued for reprocessing. At the onset, it may seem that insertion, deletion, and reprocessing may cause an endless cycle, preventing termination. We prove that this never happens.

3.4. Reprocessing Nodes

Algorithm 2 NodeEnqueue(ν, s, λ)

```
1: Compute $W := \{\nu' \neq \nu | d(s,\nu') \leq 2\lambda\}$
2: for each $\nu' \in W$ do
3:     Enqueue$(\nu', Q)$
4: end for
```

Each node in the octree represents a well-defined subset of P, specifically a node represents its $R_\nu = P_\nu \bigcup N_\nu$. It is possible that when a point $q \notin P$ is inserted during the refinement of node ν, it changes the content of some $R_{\nu'}$, $\nu' \neq \nu$ through insertion or deletion, and so may affect the part of the final output generated by ν'. Then in order to maintain consistency between the meshes of ν' and others we must reprocess ν'. Therefore, whenever some $q \notin P$ is inserted by node ν, we enqueue all nodes $\nu' \neq \nu$ within distance 2λ of q. See Figure 5(bottom).

3.5. Algorithm LocVol

Our algorithm LocVol first encloses the input surface ∂O into a bounding box which becomes the root of the octree subdivision. As in [DLS10], some points sampled from ∂O initialize P. The inner while loop (4-12) processes a node ν assuming a routine Violation that checks for respective violations of conditions in the priority order C1-C6.

When a node ν is not being processed, we maintain its sample set $P_\nu = P \bigcap \nu \subseteq P$ and a list of tetrahedra $\bigcup_{p \in P_\nu} T_p$. This list is maintained to avoid recomputing the mesh in a node while finalizing the output mesh. When ν is extracted from Q for processing, we compute Del R_ν which is also updated with each insertion and deletion of point(s). To check violations C1-C4, we compute and maintain the restricted surface triangulation Del $R_\nu|_{\partial O}$. To check violations C5-C6, we also need the restricted volume triangulation Del $R_\nu|_O$. We initially identify the tetrahedra of Del $R_\nu|_O$ by walking from a dummy tetrahedron in Del R_ν which is assumed to be at infinity, and marking tetrahedra appropriately by making a note each time we pass through a restricted triangle. After the initial marking, subsequent markings (due to point insertion/deletion) can be handled locally.

We prove that LocVol terminates. At termination we output $\bigcup_{p \in P} T_p$. Figures 2,6, and 7 show some examples.

Algorithm 3 LocVol$(\partial O, \kappa, \lambda)$

```
1: Initialize $P$ and Initialize $Q$ with a bounding box;
2: while $Q$ is not empty do
3:     $\nu :=$ Dequeue$(Q)$;
4:     while $(p,q) :=$ Violation$(\partial O, \nu, \lambda)$ is not null do
5:         $s :=$ InsertDelete$(\nu, p, q, \lambda)$
6:         if $s \notin P$ then
7:             NodeEnqueue$(\nu, s, \lambda)$
8:         end if
9:         if $|P_\nu| \geq \kappa$ then
10:            Split $\nu$ and enqueue its eight children to $Q$
11:        end if
12:    end while
13: end while
14: Return $P$ and $\cup_p T_p$.
```

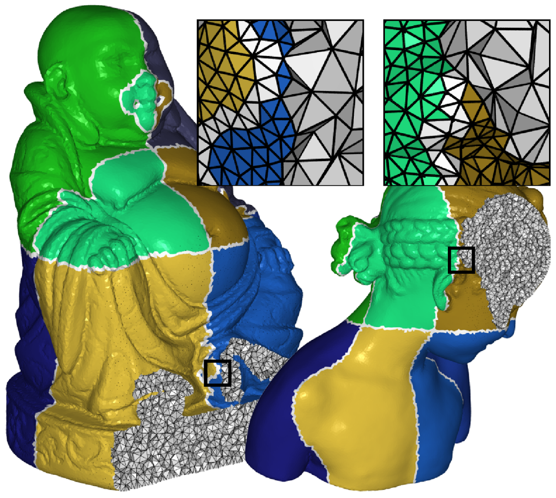
Figure 6: *Buddha and Bimba with 718K and 661K tetrahedra respectively.*

4. Guarantees

4.1. Termination

We use a standard packing argument that only finitely many points can be inserted in a *bounded* domain with a fixed lower bound on inter-point distances. The points inserted on the boundary ∂O satisfy the boundedness condition automatically. However, the circumcenters of tetrahedra that are inserted may lie outside the volume O, and hence potentially threaten to populate an unbounded domain. Condition C4 prevents this by recalling a result from [ORY05, Lemma 1]: If each Voronoi edge in $\mathrm{Vor} R_V$ either intersects ∂O in a single point (transversally) or does not intersect it at all, then $\partial(\mathrm{Del} R_V|_O) = \mathrm{Del} R_V|_{\partial O}$. The conclusion means that the surface triangulation $\mathrm{Del} R_V|_{\partial O}$ also bounds the restricted volume triangulation $\mathrm{Del} R_V|_O$ whose tetrahedra necessarily have their circumcenters in O by definition. Therefore, if we access only those tetrahedra that lie in a volume bounded by the restricted surface triangulation, we are ensured that their circumcenters are in O. The algorithm LocVol precisely finds these tetrahedra at a time when condition C4 is satisfied. Therefore, all points inserted by LocVol lie in a bounded domain, namely in O.

The points inserted by LocVol are distinguished into two classes, the *surface points* which lie on ∂O and the *volume points* which lie in the interior $O \setminus \partial O$. Now we argue that the algorithm terminates.

Assume that the algorithm does not terminate. Then we have either an infinite number of deletions or an infinite number of insertions; however, because we only delete when we insert a surface point, one must have an infinite number of surface insertions in order to have an infinite number of deletions. Because we never delete surface points, an infinite number of surface insertions implies that there is no fixed lower bound on the distance between pairs of surface points. We show that this is not the case. To facilitate discussion, let us divide point pairs into three main classes: surface-surface pairs (ss), surface-volume pairs (sv), and volume-volume pairs (vv). Each refinement criterion, directly or indirectly, implies a lower bound for the interpoint distances of one or more of the above pair types, and this is the premise of the proof of termination. Let d_{ss}, d_{sv}, and d_{vv} denote the smallest inter-point distances in the current pair classes. Furthermore, let $\bar{d}_{ss}$, $\bar{d}_{sv}$, and $\bar{d}_{vv}$ denote the new such distances after a point insertion or deletion in any of the violations C1-C6.

First, we argue the following implications:

- (C1) $\Rightarrow \bar{d}_{ss} \geq \min\{d_{ss}, \lambda\}$, $\bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$
- (C2) $\Rightarrow \bar{d}_{ss} \geq \min\{d_{ss}, d_{sv}/2, d_{vv}/2\}$ and $\bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$
- (C3) $\Rightarrow \bar{d}_{ss} \geq \min\{d_{ss}, \lambda_{\partial O}\}$ and $\bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$
- (C4) $\Rightarrow \bar{d}_{ss} \geq \min\{d_{ss}, \lambda^*\}$ and $\bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$
- (C5) $\Rightarrow \bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$ and $\bar{d}_{vv} \geq \min\{d_{vv}, \lambda\}$.
- (C6) $\Rightarrow \bar{d}_{sv}, \bar{d}_{vv} \geq \min\{2d_{ss}, d_{sv}, d_{vv}\}$ or $\bar{d}_{ss} \geq \min\{d_{ss}, d_{sv}, d_{vv}\}$.

Notice that during insertion InsertDelete either augments R_V with an existing point s from P, or inserts a new point q. When a new point q is inserted, its closest point p over the global point set P lies in R_V. Also, because insertion of a surface point causes the removal of all volume points within a distance of λ of the inserted point, we have $\bar{d}_{sv} \geq \min\{d_{sv}, \lambda\}$ for C1...C4. We argue for the other implications in the following.

(C1): In this case we consider q because t has circumradius more than λ. The distance $d(q,p) = d(q,P)$ is at least λ providing the implication.

(C2): Consider the case where C2 is violated by a restricted triangle having at least one surface sample as a vertex. Then the inserted point q must be at least $d_{sv}/2$ away from all other points. If C2 is violated by a triangle having no surface samples as vertices, then the inserted point must be at least $d_{vv}/2$ away from all other points. This leads to $\bar{d}_{ss} \geq \min\{d_{ss}, d_{sv}/2, d_{vv}/2\}$.

(C3): Here we can argue as in [DLS10] that $d(q,p) = d(q,P)$ is at least a surface dependent constant $\lambda_{\partial O}$.

(C4): When C4 is considered, condition C2 ensures that f has all vertices on ∂O. We can safely assume that f has a circumradius smaller than any fixed positive constant, say λ^*, since otherwise $d(q,p) = d(q,P) > \lambda^*$, and we are done. Therefore, assume in particular that $d(p,q) \leq \lambda^*$ for $\lambda^* = 0.06 \mathrm{lfs}_{min}(p)$ where $\mathrm{lfs}_{min}(p) = \min_p \mathrm{lfs}(p)$ is the minimum local feature size of ∂O at p [AB99], and thus a surface dependent constant. The furthest intersection point of the dual Voronoi edge of such a small triangle f with the surface must lie at least $\mathrm{lfs}(p)$ away from p; see [Dey06, Lemma 3.7]. It follows that $d(q,P) > \lambda^* > \mathrm{lfs}_{\min}$.

(C5): Since q in this case is the circumcenter of a tetrahedron with circumradius at least λ, $d(p,q) = d(q,P) \geq \lambda$.

(C6): Recall that for a violation of C6, we insert the Voronoi vertex q if it is not inside any surface Delaunay ball, and insert a surface point otherwise. In the former case, the circumradius r of the tetrahedron t being split is more than 2 times its smallest edge. Then, $r > 2\min\{d_{ss}, d_{sv}, d_{vv}\}$. Since $r = d(q,p) = d(q,P)$, we have $\{\bar{d}_{sv}, \bar{d}_{vv}\} \geq \min\{2d_{ss}, d_{sv}, d_{vv}\}$. In the other case, when a surface point is inserted, the surface Delaunay ball containing the circumcenter of t has a radius at least $r/2$. Then, we have $\bar{d}_{ss} \geq r/2 \geq 2\min\{d_{ss}, d_{sv}, d_{vv}\}/2 = \min\{d_{ss}, d_{sv}, d_{vv}\}$.

Theorem 1 LocVol terminates.

Proof. As we argued earlier, we need to show that finitely many surface and volume points are inserted, or d_{ss} and d_{vv} have a positive lower bound. Observe that all steps maintain a positive lower bound on d_{ss} except possibly C2 and C6. In C2, it may decrease to half of d_{sv} or d_{vv}. But, d_{sv} and

d_{vv} maintain a positive lower bound through all steps except possibly in C6 where it may decrease to $2d_{ss}$. But, then, a possible cycle through C2 and C6 cannot decrease d_{ss}. This implies a positive lower bound on d_{ss} through all steps. Since d_{vv} has a positive lower bound in every step except possibly in C6 where it can be $2d_{ss}$, we have the desired result.

4.2. Geometric and Topological Guarantees

Theorem 2 The output mesh of LocVol(∂O,κ,λ) satisfies the following two properties:

(i) The output mesh $T = \cup_p T_p$ is a subcomplex of the restricted Delaunay complex $\mathrm{Del}\, P|_O$ with ∂T a 2-manifold. Each point in the output is at a distance λ from O and each tetrahedron has radius-edge ratio at most 2.

(ii) There exists a $\lambda^* > 0$ so that if $\lambda < \lambda^*$, the output mesh becomes isotopic to O with a Hausdorff distance $O(\lambda^2)$.

Proof. (i) We show that each tetrahedron $t \in \cup_p T_p$ is a restricted Delaunay tetrahedron in the global triangulation $\mathrm{Del}\, P$. Let t belong to T_p, where p is in the node ν. When ν is processed for the last time, the circumscribing ball B of t is empty of points in R_ν since it is a Delaunay tetrahedron in $\mathrm{Del}\, R_\nu$. No points of P can lie inside B either. Since B has a radius less than λ (condition C5), any such point would be within 2λ distance of $p \in \nu$ and thus would be in R_ν by definition. We claim that no point of P can be inserted in B after the last time ν is processed because, assuming the contrary, the point inserted inside B has to be generated by processing another node ν' after ν. In that case, ν would have a point inserted within 2λ distance, as the B has radius at most λ. This would cause ν to be enqueued and reprocessed. But, this would contradict that ν had already been processed for the last time.

Now we argue that $S = \partial T$ is a 2-manifold. Notice that any triangle which is in ∂T is a restricted Delaunay triangle f in $\mathrm{Del}\, R_\nu|_{\partial O}$ for some node ν when ν is processed for the last time. The restricted triangles incident to a point $p \in \nu$ form a topological disk (condition C3). By an argument similar to the one used for tetrahedra, we can show that these triangles remain restricted in the global triangulation $\mathrm{Del}\, P$. Furthermore, they remain so when no volume point is considered. Now we can argue as in [DLS10] to establish that triangles in S are consistent, that is, if a triangle $f \in S$ has vertices a,b,c, it appears in each surface star F_a, F_b, and F_c. In sum, the triangles incident to each vertex in S form a topological disk (with each edge having exactly two triangles). By a standard result in PL topology S must be a 2-manifold.

Since all tetrahedra in $\cup_p T_p$ have circumcenters in O and have radii no larger than λ, all points in $\cup_p T_p$ lie within λ distance of O. Condition C6 ensures that all tetrahedra in this set have radius-edge ratio no more than 2. This completes the proof of (i).

(ii) We have already shown that S is a subcomplex of the global triangulation restricted to the surface, that is, $S \subset \mathrm{Del}\, P|_{\partial O}$. Let P' denote the vertex set of S. Since $P' \subseteq P$, we have $S \subset \mathrm{Del}\, P'|_{\partial O}$. Now consider a triangle $f \in \mathrm{Del}\, P'|_{\partial O} \setminus S$. We claim that no such f exists when $\lambda \leq 0.06 f_{min}$ proving that $S = \mathrm{Del}\, P'|_{\partial O}$. When $\lambda \leq 0.06\mathrm{lfs}_{min}$, the restricted triangles in $\mathrm{Del}\, P'|_{\partial O}$ around any point $p \in P'$ must make a topological disk [BO05, CDRR07]. This disk is already made by the triangles in S and there is no room for any extra triangle such as f proving the claim.

Since $S = \partial T$ coincides with a restricted surface triangulation, the underlying space of ∂T is isotopic to ∂O when $\lambda \leq 0.06\mathrm{lfs}_{min}$ [BO05]. It follows from standard results in piecewise linear topology that the underlying space of T is isotopic to the volume O bounded by ∂O. The bound of $O(\lambda^2)$ on Hausdorff distances between S and ∂O also follows from the result in [BO05]. This completes the proof.

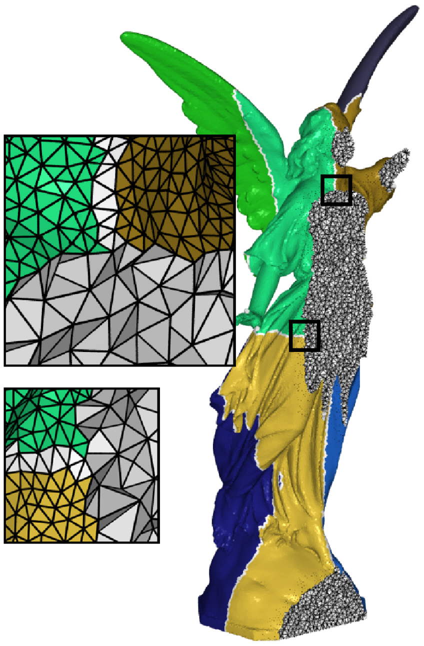
Figure 7: *Lucy with 729K tetrahedra.*

5. Experiments and Results

We have implemented LocVol using the Delaunay triangulation of CGAL [cga]. A number of experiments were conducted on a PC with 2.0GB of 667MHz RAM, 1.5GB swap space, and a 2.8GHz processor running with Ubuntu 9.04. In all tables, the parameter λ is expressed as a factor of the smallest dimension of the bounding box of O. Its effect on mesh size is illustrated in Figure 10.

It is worth noting that the times recorded herein are not CPU times, but rather the total time elapsed from the beginning of the experiment to the end. We have elected to

model	λ	κ	#tets	mem	time
Buddha	0.0075	2M	7.90M	2616	12:25
Buddha	0.0075	1M	7.90M	1936	3:30
Buddha	0.0075	500k	7.91M	1001	5:30
Buddha	0.0075	250k	7.91M	730	5:35
Buddha	0.007	2M	*Abort	*Abort	*Abort
Buddha	0.007	1M	9.72M	1945	7:00
Buddha	0.007	500k	9.72M	1127	8:10
Buddha	0.007	250k	9.72M	760	7:50
Bimba	0.0065	2M	7.75M	2616	11:05
Bimba	0.0065	1M	7.75M	1968	2:30
Bimba	0.0065	500k	7.75M	1330	4:55
Bimba	0.0065	250k	7.75M	766	3:50

Table 1: *Time(hr:min) and memory(MB) usage while varying κ. All experiments for $\kappa \leq$1M had 8 nodes except Buddha for $\lambda = 0.007$, $\kappa = 250$k, which had 36.*

present this timing instead of CPU timing because the main purpose of our algorithm is to scale better by avoiding memory thrashing.

5.1. Tuning κ

The parameter κ is the maximum number of points that any one node is permitted to contain, and as such the number of nodes required to maintain a given triangulation in LocVol is heavily dependent on its value. There is a time-overhead associated with the processing of each node; specifically, for each node v we must find its N_v, construct its Del($N_v \bigcup P_v$), and possibly check whether some simplices satisfy the refinement criteria that were already known to satisfy these criteria at the end of some previous visit to v or one of its ancestors.

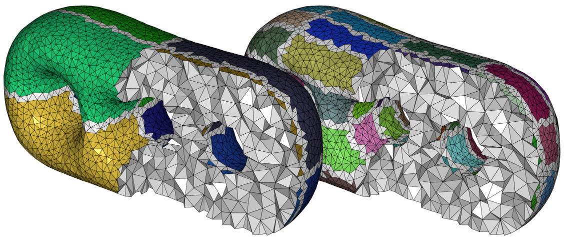

Figure 8: *3Holes: Mesh quality is mostly impervious to κ ($\kappa = 4000, 500$ from left to right).*

This overhead favors the single-node case where $\kappa \geq |P|$; however, as the size of the triangulation increases so does memory consumption, and when memory consumption significantly exceeds the available space in main memory, the single-node case begins to slow due to frequent page swapping. It is reasonable to hypothesize that LocVol performs optimally when κ is chosen maximally such that LocVol never significantly exceeds the capacity of main memory. Such a selection minimizes overhead while avoiding frequent page swaps. The experimental data in Table 1 supports this hypothesis, showing κ = 1M to be the optimum (from among the values tested) on our platform for various input models. We also see that a change in 2M tetrahedra (approx. 0.3M vertices) causes only a very small change in peak memory. This indicates that the optimal κ is 'nearly' independent of mesh size (which is governed by λ). But, of course, as huge meshes occupy more memory even to store only the vertex data structures, less memory becomes available for other purposes affecting the optimal κ value.

5.2. Scaling

Table 2 shows the experimental results of three comparable experiments (two using LocVol, one using CGAL's "mesh_polyhedral_domain" from CGAL 3.8, in release mode with -O3 optimization) run on several input models. One may notice that the LocVol experiments with κ = 2M have only a single node, and thus they do not leverage the localization techniques of our algorithm, whereas the κ = 1M shown here do make use of localization. A comparison of the results of these two types of experiments then reveals something about the effects of incorporating localization into Delaunay refinements in general: namely that localization of an implementation of Delaunay refinement conserves memory and thereby attains a speedup (at least for large refinements) over a non-localized implementation that is otherwise the same.

The memory recorded for each experiment is the peak memory footprint observed. We observe that the memory requirements for our localized algorithm are lower than those required by our non-localized implementation and the CGAL demo for comparable experiments.

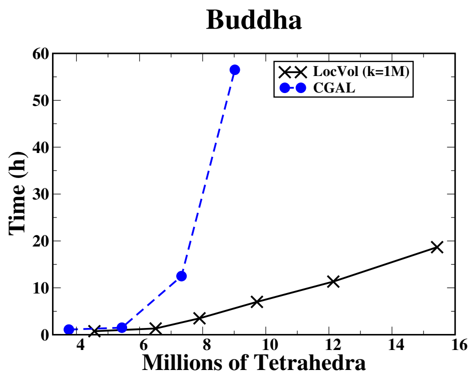

Figure 9: *Time plots to illustrate scaling.*

5.2.1. Comparison to CGAL volume meshing

We selected CGAL's "mesh_polyhedral_domain" demo for comparison to LocVol for two reasons. First, we wanted to

model	λ	**V**ersion	#leaf nodes	#tets	#verts	**m**em (MB)	**t**ime (hr:min)
9HandleTorus	0.028	LocVol: $\kappa = 2M$	1	7.65M	1.26M	2598	14:30
	0.028	LocVol: $\kappa = 1M$	8	7.65M	1.26M	1921	5:10
	0.028	CGAL	-	7.14M	1.18M	2240	13:55
OctaHandles	0.0032	LocVol: $\kappa = 2M$	1	7.32M	1.22M	2545	11:40
	0.0032	LocVol: $\kappa = 1M$	8	7.32M	1.22M	1943	2:30
	0.0032	CGAL	-	6.82M	1.14M	2185	8:10
Bracelet3	0.0105	LocVol: $\kappa = 2M$	1	7.46M	1.25M	2592	14:50
	0.0105	LocVol: $\kappa = 1M$	8	7.46M	1.25M	1923	3:10
	0.00105	CGAL	-	6.95M	1.17M	2213	12:00
HoledRing	0.0042	LocVol: $\kappa = 2M$	1	6.91M	1.19M	2483	10:15
	0.0042	LocVol: $\kappa = 1M$	8	6.90M	1.19M	1941	1:30
	0.0042	CGAL	-	6.43M	1.11M	2119	3:10
3Holes	0.0115	LocVol: $\kappa = 2M$	1	7.80M	1.26M	2574	11:40
	0.0115	LocVol: $\kappa = 1M$	8	7.80M	1.26M	1889	3:00
	0.00115	CGAL	-	7.26M	1.18M	2241	15:30
Homer	0.0085	LocVol: $\kappa = 2M$	1	8.02M	1.30M	2658	16:25
	0.0085	LocVol: $\kappa = 1M$	8	8.02M	1.30M	1914	4:25
	0.0085	CGAL	-	7.45M	1.22M	2315	20:00
Lucy	0.007	LocVol: $\kappa = 2M$	1	7.50M	1.25M	2576	12:15
	0.007	LocVol: $\kappa = 1M$	8	7.50M	1.25M	1920	4:05
	0.007	CGAL	-	6.98M	1.16M	2212	9:15
Bimba	0.0065	LocVol: $\kappa = 2M$	1	7.75M	1.25M	2616	12:05
	0.0065	LocVol: $\kappa = 1M$	8	7.75M	1.25M	1968	2:35
	0.0065	CGAL	-	7.20M	1.17M	2287	18:10
Buddha	0.0075	LocVol: $\kappa = 2M$	1	7.90M	1.27M	2616	12:25
	0.0075	LocVol: $\kappa = 1M$	8	7.90M	1.27M	1936	3:35
	0.0075	CGAL	-	7.33M	1.19M	2292	12:30

model	λ	**V**ersion	#leaf nodes	#tets	#verts	**m**em (MB)	**t**ime (hr:min)
Buddha	0.006	LocVol: $\kappa = 1M$	8	15.43M	2.47M	2005	18:40
Buddha	0.0065	LocVol: $\kappa = 1M$	8	12.14M	1.95M	1961	11:20
3Holes	0.0105	LocVol: $\kappa = 1M$	8	10.25M	1.66M	1912	6:30

Table 2: *Top table: Time and memory usage for different models for CGAL (CGAL 3.8, release mode, -O3 optimization) and single- and multi-node mesh generation with* LocVol. *Bottom table: aborted CGAL experiments for which a comparable* LocVol *terminates.*

compare to a reputable meshing tool. Second, the method employed by CGAL's volume meshing tools is readily applicable to meshing volumes bounded by smooth surfaces, and therefore appropriate for comparison to methods such as LocVol that are geared toward meshing volumes bounded by smooth surfaces.

We find that our implementation of the algorithm described herein achieves a considerable speedup over CGAL's "mesh_polyhedral_domain" demo. This is mainly due to the memory swapping that occurs during the CGAL experiments, as they average 320 swaps/second while our localized experiments average 10 swaps/second. Furthermore, we find that there are some experiments that CGAL cannot complete because the OS aborts them due to their memory requirements, but LocVol is able to complete comparable experiments (even with 15M tetrahedra) because its memory footprint for a reasonable value of κ is smaller. Bottom table of Table 2 shows some examples of this. Figure 9 shows the time plots comparing CGAL and LocVol, where one may observe that after 5.5M, though the slope (which corresponds to the number of hours per tetrahedron) of LocVol's plot increases, its increase in slope is considerably less than that of CGAL's plot, showing that LocVol does indeed scale better than CGAL.

6. Discussions

Our volume meshing algorithm has many notable features: (i) it always guarantees a manifold output irrespective of the choice of the scale parameter while ensuring shape quality (radius-edge ratio) for all tetrahedra, (ii) it captures the input

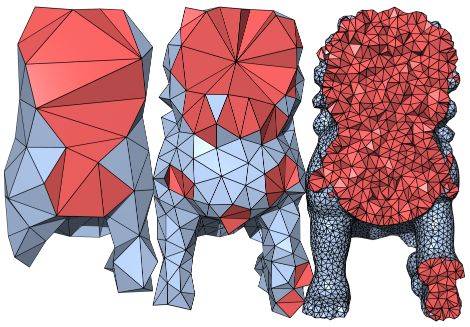

Figure 10: *Lion: Meshes with different resolutions and approximation quality can be produced by changing λ ($\lambda = 0.2, 0.1, 0.025$ from left to right).*

topology and geometry when the scale parameter is sufficiently small, (iii) it scales well, (iv) it can lead to a possible parallelization of the Delaunay refinements for smooth surfaces and volumes, and above all (v) it beats a state-of-the art meshing tool by many folds for large meshes while not sacrificing the mesh quality.

There is one limitation of our algorithm worth mentioning: the method cannot get rid of slivers, a notorious problem known for Delaunay refinement. Recently, optimization techniques have been combined with Delaunay refinement to suppress the occurrence of slivers in practice [TWAD09]. It would be an interesting exercise to investigate whether our localized Delaunay refinement technique can be extended to this approach.

Our algorithm is geared toward meshing volumes bounded by smooth surfaces. Non-smooth volumes bounded with piecewise smooth surfaces is a well known challenge in the meshing community. The algorithms proposed in [CDR08, DL09] which leverage weighted Delaunay triangulations to protect non-smooth features have provable guarantees in meshing. We plan to investigate whether this approach can be adapted to incorporate our localized Delaunay refinement framework.

Acknowledgments.

Most of the models used in this paper are taken from the AIM@SHAPE repository. We acknowledge CGAL consortium for making the Delaunay triangulation code available for experiments. The work on this research is partially supported by NSF grants CCF-0830467 and CCF-0915996.

References

[AB99] AMENTA N., BERN M.: Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry 22* (1999), 481–504. 6

[ACR03] AMENTA N., CHOI S., ROTE G.: Incremental constructions con BRIO. In *Proceedings of the 19th Annual Symposium on Computational Geometry* (2003), pp. 211–219. 1

[ACSYD05] ALLIEZ P., COHEN-STEINER D., YVINEC M., DESBRUN M.: Variational tetrahedral meshing. *ACM Transactions on Graphics 24*, 3 (July 2005), 617–625. 1

[BO05] BOISSONNAT J.-D., OUDOT S.: Provably good surface sampling and meshing of surfaces. *Graphical Models 67* (2005), 405–451. 1, 7

[BO06] BOISSONNAT J.-D., OUDOT S.: Provably good sampling and meshing of Lipschitz surfaces. In *Proceedings of the 22nd Annual Symposium on Computational Geometry* (2006), pp. 337–346. 1

[CDR08] CHENG S.-W., DEY T. K., RAMOS E. A.: Delaunay refinement for piecewise smooth complexes. *Discrete & Computational Geometry* (2008). 1, 10

[CDRR07] CHENG S.-W., DEY T., RAMOS E., RAY T.: Sampling and meshing a surface with guaranteed topology and geometry. *SIAM Journal on Computing 37* (2007), 1199–1227. 1, 7

[cga] http://www.cgal.org. 1, 7

[Che89] CHEW L. P.: *Guaranteed-quality triangular meshes*. Tech. Rep. Report TR-98-983, Department of Computer Science, Cornell University, Ithaca, New York, 1989. 1

[Dey06] DEY T. K.: *Curve and surface reconstruction : algorithms with mathematical analysis*. Cambridge University Press, New York, 2006. 6

[DL09] DEY T. K., LEVINE J. A.: Delaunay meshing of piecewise smooth complexes without expensive predicates. *Algorithms 2* (2009), 1327–1349. 10

[DLS10] DEY T. K., LEVINE J. A., SLATTON A. G.: Localized Delaunay refinement for sampling and meshing. *Computer Graphics Forum 29* (2010), 1723–1732. 1, 2, 5, 6, 7

[ILSS06] ISENBURG M., LIU Y., SHEWCHUK J., SNOEYINK J.: Streaming computation of Delaunay triangulations. *ACM Trans. Graphics 25*, 3 (2006), 1049–1056. 1

[NCC04] NAVE D., CHRISOCHOIDES N., CHEW L.: Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications 28* (2004), 191–215. 1

[ORY05] OUDOT S., RINEAU L., YVINEC M.: Meshing volumes bounded by smooth surfaces. In *Proceedings of the 14th International Meshing Roundtable* (2005), pp. 203–219. 1, 2, 6

[Rup95] RUPPERT J.: A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms 18* (1995), 548–585. 1

[She98] SHEWCHUK J. R.: Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry* (1998), pp. 86–95. 1

[TWAD09] TOURNOIS J., WORMSTER C., ALLIEZ P., DESBRUN M.: Interleaving Delaunay refinement and optimization for practical isotroic tetrahedron mesh generation. *ACM Trans. Graphics 28* (2009). 1, 10