Fully Incremental 3D Delaunay Refinement Mesh Generation

Gary L. Miller[*] Steven E. Pav[†] Noel J. Walkington[‡]

August 1, 2002

ABSTRACT

The classical meshing problem is to construct a triangulation of a region that conforms to the boundary, is as coarse as possible, and is constructed from simplices having bounded aspect ratio. In this paper we present a fully incremental Delaunay refinement algorithm. The algorithm is an extension of one introduced by Ruppert. The algorithm is fully incremental in the sense that it does not need any preprocessing to find an initial Delaunay triangulation or an initial refinement to refine away all encroached simplices of input faces. The paper includes a careful statement of the algorithm, an outline of a proof of correctness even when the input may be degenerate, and bounds on the mesh size. The input angle assumption has been relaxed to $\arccos \frac{1}{2\sqrt{2}} \approx 70^\circ$ for all but dihedral angles.

Keywords:Delaunay Refinement, Delaunay Triangulation, Mesh Generation, Ruppert's Algorithm

1 INTRODUCTION

In 1992 Ruppert [1, 2], building upon an idea of Chew [3], introduced a two dimensional Delaunay Refinement meshing procedure that is conceptually very elementary, provably producing meshes within a constant factor of optimal in size. The algorithm as implemented in Triangle [4] works remarkably well in practice, with thousands of users. Moreover, the ideas introduced by Ruppert to establish these results are very elegant and have been useful in a much broader context [5, 6, 7]. Shewchuk [8], has developed a three dimensional version of Ruppert's Delaunay refinement ideas. At the time of this writing we do not know of any publicly available implementation of a three dimensional Delaunay refinement meshing algorithm. We have implemented a fully incremental refinement algorithm with some success [9]. One of the goals of this paper is to give a complete enough exposition so that one may relatively easily implement a three dimensional Delaunay refinement meshing algorithm. We hope to release our version shortly and have included some simple figures from it.

While it is easy to state how Delaunay refinement algorithms should function, many subtle points arise when implementing them in three dimensions. These algorithms construct distinct triangulations of each input edge, face, and region; however, these triangulations can not be maintained independently of each other. Understanding and controlling the interactions between this partial order of meshes is critical for analysis and correctness of any algorithm. This issue is not addressed in the high level descriptions of these algorithms stated in the literature. By making the algorithm fully incremental, and by introducing certain induction hypotheses, these interactions becomes more explicit and easier to understand. Geometric degeneracies also give rise to subtle problems rarely addressed in the high level descriptions of the algorithms. In the absence of degeneracies, upon termi-

[*]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. Supported in part by National Science Foundation Grants CCR–9902091, CCR-9706572, and ACI-0086093.

[†]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213 Supported in part by National Science Foundation Grants DMS–9973285, CCR–9902091, and ACI-0086093.

[‡]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213. Supported in part by National Science Foundation Grants DMS–9973285, CCR–9902091, and ACI-0086093. This work was also supported by the NSF through the Center for Nonlinear Analysis.

nation of the algorithm all of the "distinct" triangulations of the edges and faces agree with the triangulations inherited from the three dimensional region. In practice, geometric degeneracies can cause Delaunay refinement algorithms to recurse infinitely (even if exact arithmetic is assumed). To resolve this problem we introduce a partial order on the "work queue" which guarantees certain compatibilities between the partial order of meshes and in the presence of exact arithmetic eliminates problems associated with degeneracy.

The optimal run time of Delaunay refinement algorithms is still an unknown and we do not address this here. Examples suggest that, in the worst case where there are both very large and very small features, the running time is $\Omega(M^2)$ where M is the size of the output mesh. One goal of this paper is to present an algorithm which is not blatantly quadratic in its run-time. As stated above, many of the algorithms in the literature do not make explicit how certain steps should be implemented. One could assume that they had in mind an auxiliary data structure, such as an octree, to facilitate various searches; the details are usually lacking. We are able to construct our mesh incrementally. That is, we begin by picking a small number of points, say four in 3D, and forming the mesh on these four vertices. At each stage we add either a new input or Steiner point to the mesh induced on these points. In some sense our algorithm is the "minimal perturbation" required to change the Bowyer Watson triangulation algorithm into a meshing algorithm.

While degeneracy is frequently dismissed as "an event having probability zero," this is not the case in practice since mechanical components typically contain rectangular faces and circular holes. Also, algorithms in computational geometry involve many "real to Boolean" operations so are inherently ill posed. As stated above, even if exact arithmetic is assumed geometric degeneracies can cause infinite recursions and other modes of failure. Our analysis assumes all quantities are computed exactly. Shewchuk [10] developed precise real to Boolean predicates, and empirical evidence shows that this approach has minimal impact upon performance. In this situation algorithms must still accommodate the geometric degeneracies. For example, if four points are cocircular, then the Delaunay triangulation is not unique. In two dimensions only one mesh is being constructed, so any runtime breaking-of-symmetry works. As stated above, in three dimensions there is a partial order of independently maintained two and three dimensional meshes and at termination they must agree. Our algorithm was carefully constructed to accommodate this requirement and effectively breaks symmetries in each mesh in a consistent fashion.

In order to show that a mesh is size optimal it is necessary to consider the **local feature size** of a piecewise linear system (PLS) (i.e. a collection polytopes with specified intersection conditions) [7]. Recall that the local feature size at a point $x \in \mathbb{R}^d$ is the distance to the second nearest disjoint feature (polytope) in the PLS. When the input data and output mesh are considered as PLS's, it is known that an output mesh with bounded aspect ratio that conforms to the input is size optimal if and only if the local feature size of the input and output are bounded above and below by a constant of each other.[1] Delaunay refinement algorithms in three dimensions do not produce meshes satisfying this definition of optimality. The meshes they produce are **pointwise optimal** in the sense that (i) they conform to the input, (ii) each simplex has a bounded ratio of circumradius to shortest edge, and (iii) the distance from any point $x \in \mathbb{R}^d$ to the second nearest mesh point, which we denote by $lfs_0(x)$, is bounded below by some constant times the local feature size of the input (see Section 5). In particular, meshes constructed using Delaunay refinement algorithms typically contain slivers. On the other hand, It has been shown that meshes satisfying the radius-edge condition have many important properties, including the fact that the the number of tetrahedra that share a given point is bounded, [5]. Thus if we bound the number of vertices in the mesh we also get a bound on the number of tetrahedrons. Recent results [11, 12] construct "smoothing" algorithms which perturb a pointwise optimal mesh into an optimal mesh.

Prior to Ruppert's paper the only two-dimensional meshing algorithm having a sound theoretical footing was the quadtree algorithm introduced by Bern, Epstein, and Gilbert [13]. Meshes produced by the quadtree algorithm have "Cartesian character" and are not invariant under rotation of the input data; moreover, empirical evidence suggests that while the output size of this algorithm is within a constant of optimal, the constant is significantly larger than that of Ruppert's procedure. The quadtree algorithm was extended to three dimensions by Mitchell and Vavasis [14]. Their octree algorithm is non-trivial; however, they have implemented a very general version of it [14, 15] and it is the only provably correct three dimensional meshing algorithm that we are aware of that has actually been implemented.

2 BACKGROUND

2.1 Notation

In order to clearly define the input we recall the following definitions from Miller, et al. [7]

Definition 2.1 (Polytope and PLS) [2] *A* polytope

[1]This definition of optimal is slightly stronger the usual concept of size optimal.

[2]Some authors have started using the term piecewise-

is the convex combination of finite set of points $P \subset \mathbb{R}^d$. The dimension of the polytope is the dimension of the affine subspace generated by P.

A piecewise-linear system*, or* PLS*, is a set of polytopes $\mathcal{W}$ with the following properties:*

- *The set $\mathcal{W}$ is closed under taking boundaries,* i.e.*, for each polytope P in $\mathcal{W}$, the boundary of P is a union of polytopes in $\mathcal{W}$.*
- *$\mathcal{W}$ is closed under intersection.*
- *For polytopes P, Q in $\mathcal{W}$, if $\dim(P \cap Q) = \dim(P)$ then $P \subseteq Q$ and $\dim(P) < \dim(Q)$.*

The dimension of a PLS is the dimension of the highest-dimensional polytope in that PLS, and we think of the polytopes of a d-dimensional PLS as being embedded in $\mathbb{R}^d$.

Our code does not restrict itself to convex polytopes. In fact the code appears to handle a wider class of input, for example the input as shown in Figure 8, one of whose 2D faces is a square annulus. Future work should specify exactly the class of input "polytopes" allowed.

In several of the proofs it is very useful to know the dimension of the lowest dimensional polytope containing a given point we call this the containing dimension of the point.

Definition 2.2 *Let W be a PLS and p be some point. The* **Containing Dimension** *of p in W, denoted $CD(p)$, is the dimension of the lowest dimensional polytope in W that contains p. If no polytope contains p then $CD(p)$ is the dimension of the underlying space, i.e., d.*

We adopt Ruppert's terminology which distinguishes the set of input edges $\mathcal{S}_0$ in the PLS from other edges by referring to them as *input edges*, and we refer to (them or) their subdivisions as *segments*. In three dimensions the input will also consist of a set of planar two dimensional polytopes denoted by $\mathcal{F}_0$ which we refer to as the *input faces*. These faces will be triangulated and we will refer to the triangles in these triangulations as *facets* and denote their union as $\mathcal{F}$. The "empty sphere" property of Delaunay triangulation plays an important role of our algorithm and the following definitions are useful in this context.

Definition 2.3 *Let $K \subset \mathbb{R}^d$ be a simplex.*

- *The* diametral ball *of K, denoted $B(K)$, is the* **open** *ball of minimal diameter containing the vertices of K. If $\dim(K) = d$ then $B(K)$ is the circumball of K.*
- *Let $p \in \mathbb{R}^d$, then p* encroaches *upon K if $p \in B(K)$. If K is a simplex in a triangulation, we say it is encroached if any vertex of the triangulation is inside its diametral ball.*

  The boundary sphere if $B(K)$ is denoted by $S(K)$.
- *The radius-edge ratio of K is the ratio of the length of the radius of $B(K)$ to the length of the shortest edge of K.*

linear complex(PLC) but the definition is the same as a PLS.

2.2 Incremental Delaunay Algorithm

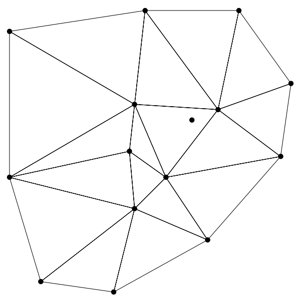
Our meshing algorithm is based upon the Bowyer Watson incremental Delaunay triangulation algorithm which we briefly recall here.

The incremental algorithm for computing the Delaunay triangulation of a set of points is initialized by determining a large bounding simplex (or box) which contains all of the points $\mathcal{P}$ and initializing $\mathcal{T}$ to be this simplex. Next, the points in $\mathcal{P}$ are added one at a time to $\mathcal{T}$ as follows. Given $p \in \mathcal{P}$, remove all of the d-simplices in $\mathcal{T}$ that contain p in their circumballs $B(t)$. This leaves a "cavity" (the *Delaunay Cavity*) in the triangulation bounded by a set of simplices $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$ (see Figure 1). The cavity is filled by adding simplices to $\mathcal{T}$ of the form $K = (p, k_i)$ having apex p and opposite simplex $k_i \in \mathcal{K}$.

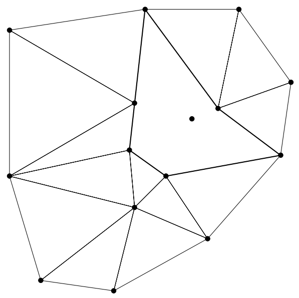
The algorithm for determining the simplices in $\mathcal{T}$ containing p in their circumsphere needs to be specified. This is a crucial step in the analysis of the algorithm. One method is to assign every point $p \in \mathcal{P}$ not yet in the triangulation to a simplex K for which $p \in B(K)$. Given a point-simplex pair, (p, K), all simplices K' for which $p \in B(K')$ can be determined by a local search since their union forms a connected patch (the cavity). As the search proceeds and simplices K' in the cavity are removed, the points $\{p'\}$ that were assigned to them are gathered, and reassigned to one of the new simplices having p as an apex, see [16]. An alternative method is to maintain a data structure for point location which essentially stores the decision tree used to assign the points to simplices in the "gather and reassign" step just described. This procedure has the advantage that the new points (such as segment split points) can be introduced at any stage.

2.3 Degeneracy

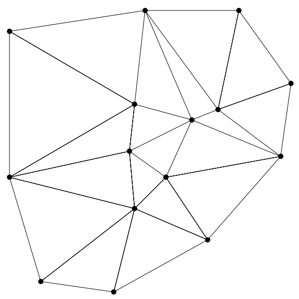
Given a collection of points $\mathcal{P}$, the classical Delaunay "empty ball" criterion states that if an empty ball B contains $\ell+1$ points on its boundary then the ℓ-simplex formed from their convex hull is present in the Delaunay triangulation of $\mathcal{P}$. This statement holds in the absence of degeneracy; that is, when the convex hull



(a) point to be added.



(b) cavity removed.



(c) tent panels added.

Figure 1: The incremental Delaunay construction.

of any $\ell + 1$ points lying on the boundary of an empty ball has dimension ℓ. When the dimension is less than ℓ the empty ball criterion does not determine a triangulation of the convex hull of the input points, instead it defines a decomposition into a unique collection of *Delaunay polytopes*.

Example: If the empty ball criterion is used to "triangulate" the eight vertices of a cube, the decomposition into Delaunay polytopes would consist of the cube, its six square faces and the edges of the cube.

In this situation Delaunay *triangulation* algorithms, represent the Delaunay polytopes as a union of simplices; clearly these triangulations are not unique. The triangulation produced by the incremental algorithm will depend upon the order in which the points are processed. We have defined the circumballs of a simplex to be open; however, the incremental algorithm will also produce a consistent, but different, Delaunay triangulation if the balls are chosen to be closed. The two dimensional *meshing* algorithm introduced below will work if diametral balls are defined to be open or closed; however, in three dimensions degeneracies could result in an infinite recursion if the diametral balls are defined to be closed. By defining circumballs to be open and assigning an order in which points are inserted into the triangulations we avoid the problem with degeneracies encountered by Shewchuk [8]. In [8] distinct Delaunay triangulations of faces input to the three dimensional meshing algorithm had to be adjusted to agree.

3 AN INCREMENTAL 2D MESHING ALGORITHM

3.1 Overview

Our incremental algorithm is a modification of the incremental Delaunay triangulation algorithm described above. Central to the implementation and analysis are the following invariants that we maintain throughout.

Induction Hypotheses: At each stage a set of points $\mathcal{P}$, segments $\mathcal{S}$, and a Delaunay triangulation $\mathcal{T}$ of a subset of $\mathcal{P}$ is maintained. These three sets satisfy

1. If the two end points of a segment $s \in \mathcal{S}$ are in $\mathcal{T}$ then s is an edge in $\mathcal{T}$
2. Segments present as edges in $\mathcal{T}$ are not encroached by vertices in $\mathcal{T}$.

We implicitly assume that the union of the sets of points, $\mathcal{P}$, and segments, $\mathcal{S}$, form a PLS in the sense outlined in Definition 2.1. Notice that in the absence of degeneracy hypothesis (2) implies hypothesis (1).

In order to interpret the induction hypotheses it is important to correctly interpret what is and what is not a segment. Input to the algorithm is a set of points $\mathcal{P}_0$ and a set of pairs of points $\mathcal{S}_0$ which correspond to input line segments. During the execution of our algorithm we maintain a set of points, $\mathcal{P}$, and segments, $\mathcal{S}$. These sets are initialized as $\mathcal{P} = \mathcal{P}_0$ and $\mathcal{S} = \mathcal{S}_0$, and subsequently an edge (p, r) in the triangulation is called a segment if and only if $(p, r) \in \mathcal{S}$. Frequently such a segment (p, r) will be split. This corresponds to removing (p, r) from $\mathcal{S}$ and adding in the two halves $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{(p, r)\}) \cup \{(p, q), (q, r)\}$ where q is the mid point of (p, r). Notice that after this operation (p, r), while still an edge in the triangulation, is no longer considered to be a segment (it is not in $\mathcal{S}$).

```
1. Procedure add-point2d()
2. INPUT: A point $p$, the current triangulation, a
   set $\mathcal{P}$ of points queued for Force'd addition to
   the triangulation, a set of segments $\mathcal{S}$, a queue
   of skinny triangles, and additionally an option
   Force|Provisional.
3. OUTPUT: The updated triangulation (with $p$ inserted),
   segment set and queue of skinny triangles, or an
   exception which returns a segment mid point.
4. Let $t$ be a triangle which $p$ encroaches.
5. DETERMINE THE DELAUNAY CAVITY:
     • Beginning at $t$, search adjacent triangles
       to determine all triangles $t'$ which $p$ encroaches.
     • If removal of a triangle eliminates a segment
       from the triangulation split the segment and,
       (a) if this is a Force'd addition, add the mid point
       to the queue $\mathcal{P}$ and continue; otherwise,
       (b) if this is a Provisional addition, abort and
       return the segment mid point.
     • As each boundary edge $(q, r)$ is located, determine
       if it is a segment. If so, and if $p$ encroaches upon
       it then split the segment and (a) if this is a Force'd
       addition add the mid point to $\mathcal{P}$ and continue;
       otherwise, (b) if this is a Provisional addition,
       abort returning the mid point.
6. FILL THE CAVITY: Remove the triangles in the
   cavity from the mesh and for each edge $(q, r)$ on
   the cavity boundary
     • Add the triangle $(p, q, r)$ to the triangulation.
     • If $(p, q)$ is a segment and $r$ encroaches upon
       it then split $(p, q)$ and add the mid point
       to $\mathcal{P}$. Similarly if $(p, r)$ is a segment and is
       encroached upon by $q$ it is split.
     • If $(p, q, r)$ has radius-edge ratio greater than
       $\rho_2$, add it to the queue of skinny triangles.
7. If $p$ is the end point of a segment $(p, q)$ not in the
   triangulation ($q$ is in the triangulation, but not
   on the cavity boundary) split the segment and
   add the split point to the queue $\mathcal{P}$.
8. If $\mathcal{P} \neq \emptyset$, remove a point $p$ from $\mathcal{P}$ and recursively
   call add-point2d(p, ... ,Force).
```

Figure 2: Point Insertion Algorithm for the 2d Incremental Algorithm

The function `add-point2d()` described in Figure 2 is a variant of the incremental Delaunay procedure designed to maintain the induction hypotheses. This function is the heart of our incremental algorithm and is lazy in the sense that it never splits segments until an encroachment is detected during the construction of a Delaunay cavity. Since the procedure for finding a triangle encroached by the point in Step 4 is identical to that used for the incremental Delaunay algorithm the details for doing this have been omitted. The parameter ρ_2 occurring in Step 6 is threshold radius-edge ratio; triangles having a smaller ratio are declared to be skinny.

The initial call to `add-point2d()` with the `Force` option will produce a mesh that conforms to the input. The final stage, where skinny triangles are eliminated, is accomplished by a function `rm-tri()` which simply makes judicious calls to `add-point2d()` as indicated in Figure 4. The complete algorithm then consists of the following three steps which are illustrated in Figure 3

1. INITIALIZATION: After reading the input sets of points and segments, initialization is exactly as in the incremental Delaunay triangulation algorithm, *i.e.*, construct a bounding triangle that contains all the input points.
2. CONSTRUCT A CONFORMING TRIANGULATION: The initial call to `add-point2d( ... ,Force)` constructs a triangulation which conforms to the input and satisfies the induction hypotheses.
3. ELIMINATE SKINNY TRIANGLES: This is accomplished by a call to `rm-tri()`.

The following two lemmas, which we state without proof, verify that our algorithm will maintain the induction hypotheses and that circumcenters of triangles do not get "too close" to input edges. This later condition being required to prove pointwise optimality of the final mesh.

Lemma 3.1 *If a mesh satisfies the induction hypotheses, then the mesh obtained by inserting a new point into the mesh by* `add-point2d()` *satisfies the induction hypotheses.*

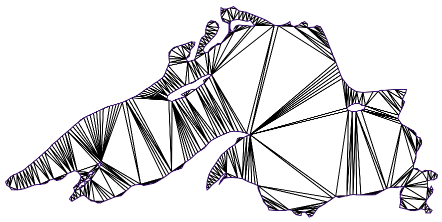
Lemma 3.2 *Let p be the circumcenter of a skinny triangle that is added to the triangulation. Then p doesn't encroach upon any segment in the triangulation when it is added, and never encroaches upon any segment subsequently added to the triangulation.*

4 A 3D IMPLEMENTATION

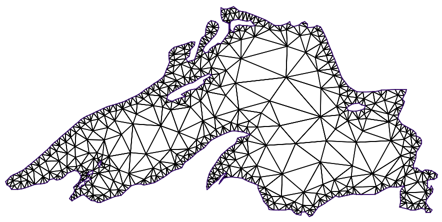
4.1 Overview

We extend the incremental algorithm developed for the two dimensional problem to solve the meshing

(a) Input data.



(b) Conforming triangulation.



(c) Final triangulation.

Figure 3: The three steps of our incremental Ruppert algorithm.

problem in three dimensions. Input points to the three dimensional algorithm will be handled as they were in the two dimensional algorithm; they are always inserted (`Force`'d) into the triangulation. As in the two dimensional algorithm, points will be added to input features to ensure that they are represented in the final (3d) mesh. In order to see the analogy between the two and three dimensional algorithm it is convenient to recall how input edges are accommodated in the two dimensional algorithm.

- A "Delaunay triangulation" of each input edge is maintained (as a set of segments) independently of the two dimensional triangulation.
- The triangulation of an edge was refined in order to ensure it appeared in the two dimensional mesh. The refinement operation, while essentially an atomic operation in two dimensions, could be decomposed into the following steps:
  - If a "simplex" (segment) in the triangulation of an edge was found to violate the induction hypotheses it was queued for removal from the triangulation of the input edge containing it.
  - To annihilate a segment from the triangulation of an edge it's (circum)center was inserted into the edges triangulation.
  - The segment center was then queued for `Force`'d addition into the two dimensional mesh.

1. Procedure `rm-tri()`
2. INPUT: The current triangulation, the segment set, and the queue of triangles to be removed from the mesh.
3. OUTPUT: The triangulation and set of segments.
4. Let t be a triangle in the queue. If t it is no longer in the triangulation remove it from the queue and continue; otherwise, attempt to add its circumcenter, p, to the triangulation using the procedure `add-point2d(p, ... ,Provisional)`. This can have two outcomes:
   - The addition is successful in which case t has been eliminated, so can be removed from the queue of skinny triangles.
   - An exception is raised and a point p' is returned. At this stage the triangulation and queue of skinny triangles have not been modified. p' is now `Force`'d into the triangulation by a call to `add-point2d(p', ... ,Force)` which does update the triangulation, segment set and queue of skinny triangles.
5. Recursively call `rm-tri()` until the set of skinny triangles is emptied.

Figure 4: Function to Eliminate Triangles.

The three dimensional algorithm will maintain independent triangulations of each input edge and input face. We refer to triangles in the triangulation of an input face as **facets** to distinguish them from other triangles appearing in the three dimensional mesh. The facets are then accommodated in the fashion outlined above for the segments with one modification: if the addition of a facet circumcenter to a 2d triangulation encroaches upon a segment (in the same triangulation), then annihilation of the facet is postponed until the segment is annihilated. In essence facets are annihilated like the skinny triangles were in two dimensions, and are processed with `rm-tri()`. If simplices in the triangulations of input edges and faces are queued

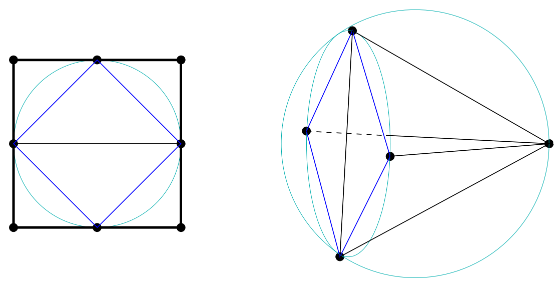


Figure 5: Distinct triangulations of cocircular points may occur in different meshes.

for annihilation it is first necessary to insert circumcenters of the lowest dimensional simplex since these can eliminate a higher dimensional simplices. Below we precisely state the minimal (partial) order required.

Unlike the two dimensional case, angle bounds upon the input may not eliminate infinite recursion in three dimensions arising from degeneracy. This is illustrated in Figure 5 where a square face is required to be represented in a 3d mesh. Since the square has two possible Delaunay triangulations it is possible that distinct triangulations appear in the (independently maintained) 2d triangulation of the face and the 3d triangulation. This would cause the center of the square to be inserted into the 2d triangulation. Since refinement of the square by repeated insertion of triangle circumcenters always results in degeneracies (four or more cocircular points) this process may recurse indefinitely. Assuming exact arithmetic, we show that this recursion can be eliminated by inserting points into each mesh in the same order.

4.2 Description of the Algorithm

A triangulation of each input polytope will be maintained. While the triangulation of an input edge is trivial, it is convenient to view it as a triangulation since all input polytopes are processed similarly. An initial Delaunay triangulation of each face will be constructed which satisfies the two dimensional induction hypotheses stated in Section 3. Our two dimensional algorithm, with the following minor modifications, is used to maintain these triangulations:

- Whenever a segment is split, the split point is queued for addition to the mesh of every polytope containing the edge.
- Similarly, whenever the circumcenter of a triangle in the triangulation of an input face is inserted into a two dimensional mesh, it is queued for addition to the 3d mesh.
- The queue of "skinny triangles" may now contain other facets queued for removal.

A work queue is maintained during the execution of the incremental algorithm. We process jobs with smaller priority value first. The queue contains pairs of the form (p, F), where p is a point to be added to a polytope F of the input PLS, or pairs of the form (K, F) where K is a simplex to be annihilated from the triangulation of F; $dim(K) = dim(F)$. The work queue is processed in an order consistent with the following partial order.

Definition 4.1 (Work Order) *A total order is first assigned to points queued for insertion into the mesh. A fixed linear order of the input points is picked and these points have smallest priority values, and at the time a Steiner point is first inserted into any mesh of any polytope it is ordered to be greater than any point considered previously. Denote this order by $<_p$. We can now define the partial order $\prec$ on the work queue by*

- *If (p, F) is a point-polytope pair and (K, F') is a simplex-polytope pair, then $(p, F) \prec (K, F')$ if $dim(F) \leq dim(F')$.*
- *If (p, F) and (q, F) are point-polytopes pairs with the same polytope, then $(p, F) \prec (q, F)$ if and only if $p <_p q$.*
- *If (K, F) and (K', F') are simplex-polytope pairs, then $(K, F) \prec (K', F')$ if $dim(F) < dim(F')$.*

This partial order states that points queued for addition to the mesh of a polytope F can be inserted at any time; however, they must be done in the order given by $<_p$. A simplex (K, F) can be processed for annihilation only if all queued points (p, F') with $dim(F') \leq dim(F)$ have been processed and all simplices (K', F') in polytopes having strictly lower dimension, $dim(F') < dim(F)$, have been processed. Notice that the recursive calls in the two dimensional algorithm provide an elegant way of processing the points in an order consistent with the above. Since the three dimensional algorithm is more complex, below we explicitly implement a scheduler. In this instance it is assumed that the recursive calls in the two dimensional algorithms are omitted. Our algorithm may attempt to add the circumcenter p of a triangle to the triangulation of an input face; however, such attempts may fail if p encroaches upon a segment. In this situation p is not included in the order $<_p$, it would only be included if the insertion succeeds.

The functions to add a point to the three dimensional mesh and to process a tetrahedra for removal are given in Figures 6 and 7 respectively. The high level description of our incremental algorithm is:

- Initialize the triangulations of each two and three dimensional polytope to be a large bounding simplex containing all of their input points. The initial triangulation of an input edge is the segment containing their two end points.

- Initialize the work queue with the all pairs of the form (p, F) for which p is an input point in the polytope F and $dim(F) > 1$.

- If the work queue is not empty, select a jobs in W of highest priority.
  - If $W = (p, F)$ is a point-polytope pair, call `add-point`i`d( ... ,Force)` where $i = dim(F)$.[3]
  - If $W = (K, F)$ is a simplex-polytope pair, (i) if K is a segment, split and queue the mid-point for addition to every polytope containing it. (ii) If K is a triangle or tetrahedra call `rm-tri()` or `rm-tet()` respectively.

4.3 Induction Hypotheses

As in the two dimensional algorithm, the function `add-point3d()` essentially augments the incremental Delaunay construction to maintain induction hypotheses which, in turn, guarantee that the input features are represented in the mesh.

Induction Hypotheses: A set of points, $\mathcal{P}$, segments, $\mathcal{S}$, and facets, $\mathcal{F}$, and a 3d Delaunay triangulation, $\mathcal{T}$, of a subset of $\mathcal{P}$ is maintained. These sets satisfy:

1. $\mathcal{F}$ is the union of the triangles in the (2d) Delaunay triangulations of the input faces. These triangulations satisfy the induction hypotheses for the two dimensional algorithm.
2. If the two end points of a segment are in the triangulation, then either the segment is present as an edge in $\mathcal{T}$ and its circumball is empty (contains no vertices of $\mathcal{T}$), or the segment has been queued for annihilation.
3. If the three vertices of a facet are in the triangulation, then either the facet appears as a triangle in $\mathcal{T}$ and its circumball is empty (contains no vertices of $\mathcal{T}$), or the facet is in the queue of simplices to be annihilated.

The following two lemmas are the three dimensional analogies of lemmas 3.1 and 3.2 respectively.

[3] Actually a variant of `add-point2d` is used which is consistent with the 3-d work queue implementation.

1. Procedure `add-point3d()`
2. INPUT: A point p to be added to the (3d) mesh and an option `Force|Provisional`. The PLS data structure is also required to facilitate integration of other triangulations and insertion into the work queue.
3. OUTPUT: The updated data structure with the point p added to the triangulation, or an exception returning a lower dimensional segment or facet k' in the triangulation of an input polytope P'.
4. FORM THE CAVITY: The first step is to remove tetrahedra $\hat{T}$ which p encroaches upon, and determine the boundary of the cavity. The tetrahedra are determined by a search starting from a tetrahedra T for which $p \in B(T)$.
   - Beginning at T search the dual graph to find all $\hat{T}$ which p encroaches upon, $p \in B(\hat{T})$.
   - When traversing a face $\hat{t}$ of $\hat{T}$ check to see if it is a facet (a triangle in the triangulation of in input face P;). If so then
     - If this a `Forced` add, $(\hat{t}, P')$ to the work queue.
     - Otherwise, this is a `Provisional` add; raise an exception returning the pair $(\hat{t}, P')$.
5. Examine the removed cavity: if any removed edge is a segment, either raise an exception, returning the segment and the input edge containing it, or insert the segment into the queue of simplices to be annihilated from the input edge, depending on if the add is `Provisional` or `Forced`, as above.
6. Examine the cavity boundary; if any face or edge on the boundary is a facet or segment, and if p encroaches upon it, either raise an exception or queue it for annihilation as above.
7. FILL THE CAVITY: Remove tetrahedra in the cavity from the mesh and for each triangle t on the cavity boundary:
   - Form the tet $T' = (p : t)$ and insert it into the triangulation. Check if any face or edge of T is a facet or segment, and if it is encroached by a vertex of T. If so, queue the encroached segment/facet for annihilation.
   - If T' is of poor quality (radius-edge ratio greater than ρ_3), queue it for annihilation.
8. If p is a vertex of a segment or a facet, and the other vertices of the segment/facet are present in the triangulation but the segment/facet is not, queue the segment or facet for annihilation.

Figure 6: Point addition routine for three dimensional meshes.

1. Procedure `rm-tet()`
2. Input: A tetrahedra T to be removed from the (3d) triangulation, and the PLS data structure to facilitate modification of the triangulation and work queue.
3. Output: Updated triangulation, and work queue.
4. If T is no longer in the triangulation return.
5. Let p be the circumcenter of T. Attempt to add p to the triangulation with `add-point3d(p, ... ,Provisional)`. There are two possible outcomes:
   - An exception was raised returning an encroached simplex t' in some sub-polytope P'. Add (t', P') the work queue and return.
   - Otherwise the point addition succeeded and T is annihilated. Remover T from the work queue and return.

Figure 7: Function to annihilate tetrahedra for the Incremental 3-d Algorithm.

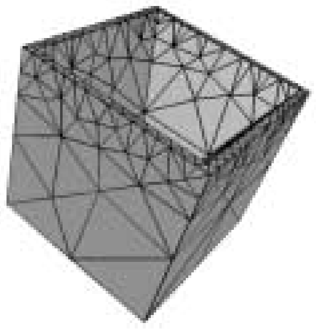
Lemma 4.2 *The induction hypotheses are satisfied.*

Lemma 4.3 *Circumcenters of tetrahedra that get added to the triangulation are never inside the circumballs of any segments or facets.*

The work order specified in Definition 4.1 guarantees that vertices common to distinct input polytopes get inserted into the (distinct) triangulations of each polytope in the same order. In this situation our algorithm will produce the same triangulation of geometrically degenerate points in each polytope; in particular, the problem illustrated in Figure 5 will not occur. The proof of this elementary observation appears as Lemma 4.7 in [9] and is used to establish the following important corollary.

Corollary 4.4 *At the time a simplex K is queued for annihilation from the mesh of the input polytope F containing K ($dim(F) = dim(K)$) either (a) K was encroached, or (b) K was skinny, or (c) the circumsphere of K contains a vertex $v \notin F$ from the triangulation of another polytope.*

Statement (c) of this corollary states that if a simplex in the triangulation of an input polytope is annihilated for no "apparent reason" (i.e. (a) or (b) fail), then the circumsphere of the simplex contains a vertex in the triangulation of another input polytope. This shows that the circumsphere intersects two distinct input features from which estimates on the local feature size can be deduced.


(a) Conforming triangulation.

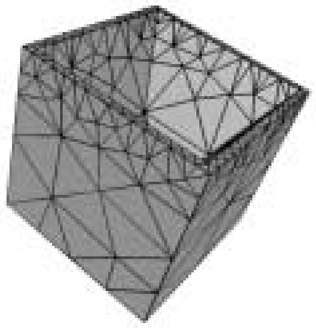

(b) Final triangulation.

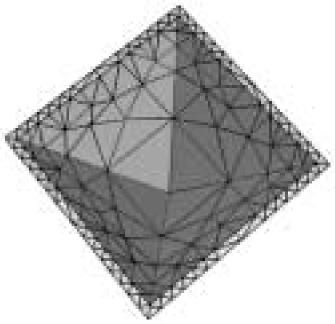
Figure 8: The two steps of our fully incremental refinement algorithm. The input was given by a PLS of a cube with a pyramid removed.

5 CORRECTNESS OF THE ALGORITHMS

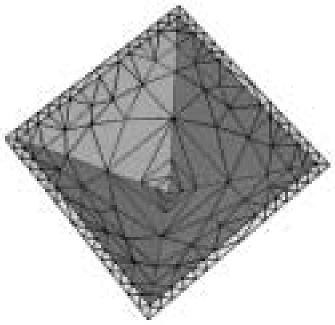
In this section we state our main theorem showing that the meshes produced by our incremental algorithm are pointwise optimal. Space limitations prohibit the inclusion of the proofs which may be found in the technical report [9].

The functions occurring in the following definition are the fundamental quantities that characterize the mesh point densities produced by our algorithm. Ruppert introduced the definition of local feature size. Here we consider a slight generalization to his definition which will allow us to state our main theorem about the output mesh size. Let W be a PLS.

Definition 5.1 *The* **local feature size** *at x in W, denoted* $\text{lfs}[W](x)$ *equals the distance to the second nearest disjoint polytope in W, i.e. the radius of the smallest radius closed ball centered at x which inter-*

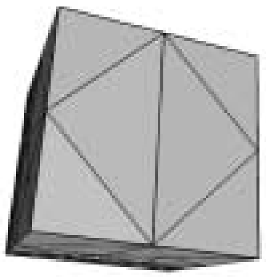


(a) Conforming triangulation.

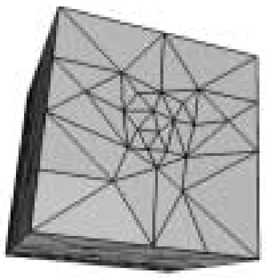


(b) Final triangulation.

Figure 9: The same figure as Figure 8, but viewed from above.



(a) Conforming triangulation.



(b) Final triangulation.

Figure 10: The same figure as Figure 8, but viewed from below.

sects two feature in W which do not intersect.

We let lfs equal lfs[W] where W in the input PLS when there is no confusion. Since the final mesh is also a PLS we may speak of its local feature size as well. It will be useful to restrict the number of polytopes in a PLS we consider. We let $W|_i$ denote the PLS consisting of polytopes of dimension at most i. Thus $\mathrm{lfs}[W|_0](x)$ is the distance from x to the second nearest point. We let $\mathrm{lfs}_0[W](x)$ denote $\mathrm{lfs}[W|_0](x)$

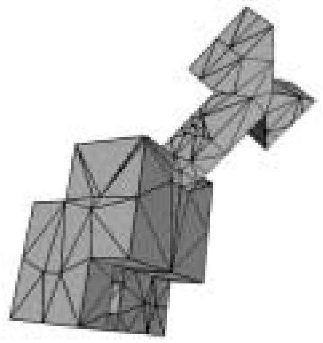
We can now state our main theorem and the corollary which establishes pointwise optimality of our meshes. The proofs are available in [9].

Theorem 5.2 *Suppose that the input PLS satisfies:*

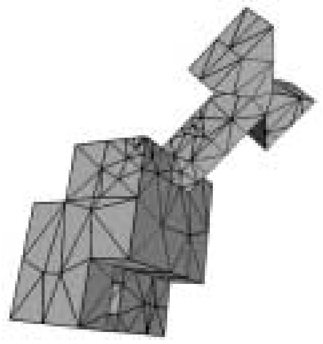
- *the angles between any two non-disjoint input segments is bounded below by $\theta > \pi/3$;*
- *the angle between a non-disjoint face and segment is bounded below by $\phi > \cos^{-1}(1/2\sqrt{2})$;*
- *if two faces meet at a point x, then the angle between any pair of rays emanating from x into each plane is bounded below by $\phi > \cos^{-1}(1/2\sqrt{2})$;*
- *the dihedral angle between non-disjoint faces is bounded below by $\pi/2$.*

Let the radius-edge ratio threshold for splitting skinny triangles satisfy $\rho_2 > \sqrt{2}$, and the threshold for splitting skinny tetrahedra be $\rho_3 > 2\sqrt{2}$. Then there exist constants C_i, $0 \leq i \leq 3$, such that at the time an attempt is made to insert the circumcenter p of a simplex K into the (unique) mesh W' of dimension $dim(K)$, then $\mathrm{lfs}[p] \leq C_i \mathrm{lfs}_0[W'](p)$ where $i = CD(p)$.

Corollary 5.3 *Suppose the algorithm is run on input that satisfies the hypotheses of the lemma. Then for all vertices p in the mesh $\mathrm{lfs}[p] \leq (1 + C_1) lfs_0[W'](p)$ where W' is the mesh output..*



(a) Conforming triangulation.



(b) Final triangulation.

Figure 11: The mesh of a complicated input which has been meshed by the algorithm.

6 COMMENTS ON IMPLEMENTATION

At several places in the procedures we needed to determine for a given point a simplex whose circumsphere contains the point, *i.e.*, the point location problem. Besides point location the two dimensional algorithm is conceptually easy to implement once the fundamental data structures to represent a triangulation and the segments set have been developed. We have a second implementation of the algorithm in C++ for 2D, [17]. This implementation uses the edge flipping algorithm as described in [16, Chapter 9] to guide the point location and it associates points queued for addition to the mesh with triangles. The only time a segment is split and there is no triangle to assign the mid point to occurs in Step 7. In this step the two end points of a segment are present in the mesh but the segment does not appear. In this situation we have just inserted the second end point in the mesh, so have a triangle containing this second point. To locate a triangle containing the mid point we implement a geometric search along the segment starting from a triangle containing the end point.

The 2D code also accommodates inputs with small angles. We implemented a version of the technique suggested by Ruppert [2]. Essentially the input is "groomed" so that segments forming a small angle form isosceles triangles in the mesh, and these isosceles triangles are never declared to be skinny. Similar approaches were considered by Shewchuk in [18]. We point out that all known heuristics for handling small angle do not have optimality results associated with them–one only proves that the algorithm terminates, so it generates a finite, though possibly very large, mesh.

For both the two and three dimensional codes precise in-sphere tests were used to determine if points encroached upon simplices. These functions use the variable precision floating point arithmetic proposed in [10] and were automatically generated from "straight line code" using the compiler developed in [19]. When generating the two dimensional Delaunay meshes of an input face embedded in three dimensions the three dimensional in-sphere test was used to determine encroachment. This was done so that the two and three dimensional components of the code would always use the same function to determine encroachment. In particular, we did not map the faces into the (x, y) plane and back, since the associated roundoff errors may cause the two and three dimensional routines to have a different view of degenerate data.

For our three dimensional code, when a point p was inserted into a mesh of a polytope F for the first time (the mesh of dimension $dim(F) = CD(p)$ containing p) it was inserted (`Force`'d) into the meshes of all polytopes containing F. It is easy to verify that this is consistent with the work order in Definition 4.1. As p is inserted into each mesh the simplices having p as a vertex are carried forward. This provides a convenient way to implement Steps 7 of the two dimensional algorithm and Step 8 of the three dimensional algorithm where the simplices containing the point being inserted are required.

As an initializiation step, our code creates, for each input polytope, a large bounding simplex in the supporting hyperplane of the polytope and which contains all the input points of that polytope. The use of a bounding simplex avoids the need for code to add a point exterior to a triangulation. Moreover, maintaining a mesh of a convex connected region which contains the input polytope simplifies our point location procedure. We must be careful, however, to avoid requiring that triangles outside the 2-d polytope not be treated as facets, which we require to be in the 3-d mesh. The simplest way of dealing with this problem is by marking triangles in the 2-d meshes as belonging "in" or "out" of the input polytope. When points are added to the 2-d meshes, the algorithm may not be able to discern those which are inside and those outside the input polytope. However, by the ordering of the work queue (Definition 4.1), all one dimensional annihilation requests have been processed before the algorithm works on removing two dimensional features, and these midpoints have been added to the 2-d meshes. The algorithm can thus, when a batch of one dimensional requests have been all processed, sweep through the 2-d meshes and properly mark unknown triangles as "in" or "out." A request to annihilate a triangle which happens to be "out" can be safely ignored: it is immaterial if the triangle is of poor quality, as it will be removed before the final meshes are output; that it does not appear in the 3-d mesh is of no relevance (in fact we don't want to force it to be there); a request to kill it will not be put back on the queue.

References

[1] Ruppert J. "A new and simple algorithm for quality 2-dimensional mesh generation." *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, TX, 1993)*, pp. 83–92. ACM, New York, 1993

[2] Ruppert J. "A Delaunay refinement algorithm for quality 2-dimensional mesh generation." *J. Algorithms*, vol. 18, no. 3, 548–585, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993)

[3] Chew L. "Guaranteed-quality triangular meshes." CS 89-983, Cornell, 1989

[4] Shewchuck J.R. "Triangle: A two-dimensional Quality Mesh Generator and Delaunay Triangulator.", 1995. See: `http://www.cs.cmu.edu/~quake/triangle.html`

[5] Miller G.L., Talmor D., Teng S., Walkington N.J. "A Delaunay Based Numerical Method For Three Dimensions: generation, formulation and partition." *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pp. 683–692. ACM Press, 1995

[6] Miller G.L., Talmor D., Teng S., Walkington N.J. "On the Radius–Edge Condition in the Control Volume Method." *SIAM J. Numer. Anal.*, vol. 36, no. 6, 1690–1708, 1998

[7] Miller G.L., Talmor D., Teng S., Walkington N.J., Wang H. "Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening." *5th International Meshing Round Table, '96*, pp. 47–62. Sandia National Laboratories, 1996

[8] Shewchuk J.R. "Tetrahedral Mesh Generation by Delaunay Refinement." *Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota)*, pp. 86–95. ACM, June 1998

[9] Miller G., Pav S.E., Walkington N.J. "An Incremental Delaunay Meshing Algorithm." Tech. rep., Department of Mathematics, Carnegie Mellon University, 2002. URL `www.math.cmu.edu/cna`

[10] Shewchuk J.R. *Delaunay Refinement Mesh Generation.* Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997. Available as Technical Report CMU-CS-97-137

[11] Edelsbrunner H., Li X., Miller, Stathopoulos G., A. Talmor D., Teng S., Ungor A., Walkington N.J. "Smoothing and cleaning up slivers." *ACM Symposium on Theory of Computing*, pp. 273–277. 2000. URL `citeseer.nj.nec.com/edelsbrunner00smoothing.html`

[12] Li X.Y., Teng S.H. "Generating well-shaped Delaunay meshed in 3D." *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pp. 28–37. ACM Press, 2001

[13] Bern M., Eppstein D., Gilbert J.R. "Provably good mesh generation." *31th Annual Symposium on Foundations of Computer Science*, pp. 231–241. Oct. 1990

[14] Mitchell S.A., Vavasis S.A. "Quality mesh generation in three dimensions." *Proceedings of the ACM Computational Geometry Conference*, pp. 212–221. ACM Press, 1992. Also appeared as Cornell C.S. TR 92-1267

[15] Mitchell S.A., Vavasis S.A. "Quality mesh generation in higher dimensions." *SIAM J. Comput.*, vol. 29, no. 4, 1334–1370 (electronic), 2000

[16] de Berg M., van Kreveld M., Overmars M., Schwarzkopf O. *Computational Geometry.* Springer, 2000

[17] Kadow C. "A fully incremental Delaunay refinement algorithm." Posterpresentation 10th International Meshing Roundtable, Oct. 2001

[18] Shewchuk J.R. "Mesh generation for domains with small angles." *Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong, 2000)*, pp. 1–10 (electronic). ACM, New York, 2000

[19] Nanevski A., Blelloch G., Harper R. "Automatic Generation of Staged Geometric Predicates." *International Conference on Functional Programming*, pp. 217–228. Florence, Italy, September 2001