Guaranteed-Quality Delaunay Meshing in 3D (Short Version)

L. Paul Chew *
Department of Computer Science
Cornell University
Ithaca, NY 14853

1 Introduction

The main contribution of this paper is a new mesh generation technique for producing 3D tetrahedral meshes. Like many existing techniques, this one is based on the Delaunay triangulation (DT). Unlike existing techniques, this is the first Delaunay-based method that is mathematically guaranteed to avoid *slivers*. A *sliver* is a tetrahedral mesh-element that is almost completely flat. For example, imagine the tetrahedron created as the (3D) convex hull of the four corners of a square; this tetrahedron has nicely shaped faces — all faces are 45 degree right-triangles — but the tetrahedron has zero volume. Slivers in the mesh generally lead to poor numerical accuracy in a finite element analysis.

The Delaunay triangulation (DT) has been widely used for mesh generation. In 2D, the DT maximizes the minimum angle for a given point set; thus, small angles are avoided. There is no analogous property involving angles in 3D. We make use of the *Empty Circle Property* for the DT of a set of point sites: the circumcircle of each triangle is empty of all other sites. In 3D, the analogous property holds: the circumsphere of each tetrahedron is empty of all other sites. The Empty Circle Property can be used as the definition of the DT.

There is a vast literature on mesh generation with most of the material emanating from the various applications communities. We refer the reader to the excellent survey by Bern and Eppstein [BE92]. We consider here only work related to the topic of mesh generation with mathematical quality guarantees. Chew [Che89] showed how to use the DT to triangulate any 2D region with smooth boundaries and no sharp corners to attain a mesh of uniform density in which all angles are greater than 30 degrees. An optimality theorem for meshes of nonuniform density was developed by Bern, Eppstein and Gilbert [BEG94] using a quadtree-based approach. Ruppert [Ru93] later showed that a modification of Chew's algorithm could also attain the same strong results as Bern, Eppstein and Gilbert. More recently, Chew [Che93] extended his own work to non-uniform density meshes for 2D curved surfaces embedded in 3D. Mitchell and Vavasis [MV92] used octrees to extend the Bern-Eppstein-Gilbert ideas to 3D.

There are several reasons why the Delaunay-based technique presented here might be preferred over the octree technique. For instance, the Delaunay-based technique produces a known bound on the aspect ratio of the tetrahedra while the mathematical theory of Mitchell and Vavasis proves the existence of such a bound for their octree-based technique without actually determining its numerical value. In addition, octree-based algorithms tend to produce meshes with some features that are undesirable in practice: the worst elements are along the boundary of the domain — typically, this is where the user would like to have the highest quality elements — and the mesh has a clear orientation in the sense that one can usually determine the orientation of the octree axes by examining the resulting mesh.

Due to space limitations, most of the proofs in this paper have been either condensed or eliminated. A more complete version of this paper is available as a tech report.

2 A Simple 2D Technique

We start with a simple 2D meshing technique to illustrate some of the ideas and to motivate the development of the 3D meshing algorithm. This technique is similar to that used in [Che89, Che93], although for ease of presentation, the version used here makes several simplifying assumptions about the region to be meshed. These simplifying assumptions are not strictly necessary and will be dropped in later sections. We assume that

1. the region to be meshed is convex,
2. the goal is to create a mesh with constant density, and
3. the boundary of the region has already been subdivided in such a way that all boundary edges have lengths between 1 and $\sqrt{3}$ units and no two vertices are closer than 1 unit.

The size of the *unit* here controls the size of the elements in the resulting mesh. The first assumption can easily be dropped, but by restricting ourselves to convex regions we avoid having to discuss Delaunay edges that occur outside the region.

The assumptions about the boundary subdivision imply something about the shape of the region to be meshed.

*This work was supported by DARPA under contract N00014-96-1-0699 and by the Cornell Theory Center which receives funding from its its Corporate Research Institute, NSF, New York State, DARPA, NIH, and IBM Corporation.

Computational Geometry 97 Nice France

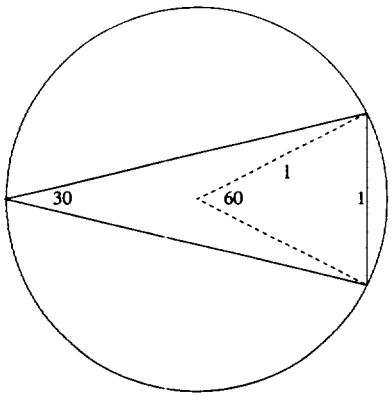

Figure 1: The minimum angle is 30 degrees.

In particular, such a subdivision cannot be obtained if the region-to-be-meshed has a boundary angle of less than 30 degrees. In practice, boundary angles less than 30 degrees can be removed from the problem region and triangulated in a postprocessing step; this type of "lopping off" technique is used in [BEG94, Ru93].

```
Algorithm A:
Form the DT of the boundary points
while there is a triangle with circumradius > 1 do
    Add the triangle's circumcenter to
     the set of sites and update the DT
```

Theorem 1 *Algorithm A halts, producing a mesh in which all triangles have angles between 30 and 120 degrees.*

Proof: First, note that no two vertices are ever closer than 1 unit; this holds initially and continues to hold throughout the algorithm. Second, note that Algorithm A must halt: the region has to fill up after some finite time since the vertices are all 1 unit apart. When the algorithm halts, each triangle has a circumcircle with radius less than 1 and each edge has length ≥ 1. It's easy to see that the smallest angle occurs when the shortest possible edge is used in the largest possible circumcircle (see Figure 1). In this case, the central angle is easily seen to be 60 degrees and the corresponding angle of the triangle is 30 degrees. The 30 degree bound for the minimum angle immediately implies the 120 degree bound for the maximum angle. □

The 2D algorithm, Algorithm A, can be generalized in a fairly straightforward way to produce a 3D algorithm: Algorithm B.

Theorem 2 *Algorithm B produces a mesh in which all triangular faces of tetrahedra have angles between 30 and 120 degrees.*

Proof: The proof requires some some fairly strong assumptions about the boundary and is fundamentally similar to the proof for Algorithm A. □

Unfortunately, knowing that a tetrahedron has nicely-shaped faces is not enough to know that it's not a sliver. The tetrahedron created from the (3D) convex hull of a square has faces that are very nicely-shaped, but the tetrahedron is the worst possible kind of sliver, with no volume at all.

3 Randomness and Approximate Circumcenters

Let us reconsider the 2D technique. Suppose we don't compute exact circumcenters. Instead, we'll choose a point randomly within a ball of radius $f < 1$ about the true circumcenter. If we do this, it's easy to see that the minimum edge length is $1 - f$. Some simple geometry shows that a chord of length $1 - f$ in a circle of radius 1 has an opposite angle of $\arcsin(\frac{1-f}{2})$. This is the minimum angle that occurs in a 2D mesh when we approximate circumcenters with points within f of the true circumcenter. Thus, $f = 0$ gives us a minimum angle of 30 degrees, as expected; $f = \frac{1}{2}$ gives us a minimum angle of about 14.5 degrees.

This technique gives us some hope for 3D. In Algorithm B, we had no choice in where to place each new vertex; if some new vertex created a sliver, we were stuck with it. But by choosing a new vertex from somewhere within an interior ball, we might be able to position the vertex in such a way that all slivers are avoided. Ideally, one might hope to calculate exactly where the new vertex should be placed within this interior ball, but this approach has not been successful, since the correct placement depends on the positions of potentially many surrounding vertices. Fortunately, randomness comes to our rescue: we don't need to determine exact placement as long as we can estimate the probability of finding a good placement when choosing random positions within the interior ball.

As a concrete basis for calculations, we take $f = \frac{1}{2}$; this is not necessarily the best value for f. Note that, with $f = \frac{1}{2}$, we expect that no two vertices will be closer that 0.5.

As before, we make some fairly strong assumptions about the region to be meshed. We assume that

1. the region to be meshed is a convex polyhedron,
2. the goal is to create a mesh with constant density, and
3. the boundary of the region has already been subdivided into triangles in such a way that all circumcircles of boundary triangles have diameter $\leq \sqrt{3}/2$ units and no two vertices are closer than 0.5 units.

```
Algorithm C:
Form the 3D DT of the boundary points
while there is a tetrahedron with circumradius > 1 do
    Randomly pick a point within 0.5 units of the
     circumcenter, repeating until we find one that
     does not form a sliver
    Add this point to the set of sites and update
     the DT
```

The exact definition of *sliver* as used in Algorithm C is determined by the requirements of the proof of the following theorem.

Theorem 3 *Algorithm C halts, producing a mesh in which all triangular faces of tetrahedra have angles between* $\arcsin(\frac{1}{4})$ *(about 14.5 degrees) and about 151 degrees. There are no slivers in the resulting mesh except possibly for tetrahedra that have all their vertices on the boundary.*

Proof: Assuming that the algorithm halts, it's easy to see that the bound on the face angles holds and that the only possible slivers are those that appear due to the initial boundary subdivision. The bound on face angles follows from the maximum circumsphere size of 1 and the minimum edge size of 1/2. The only remaining part of the proof is to show that the algorithm halts. In other words, we need to show that,

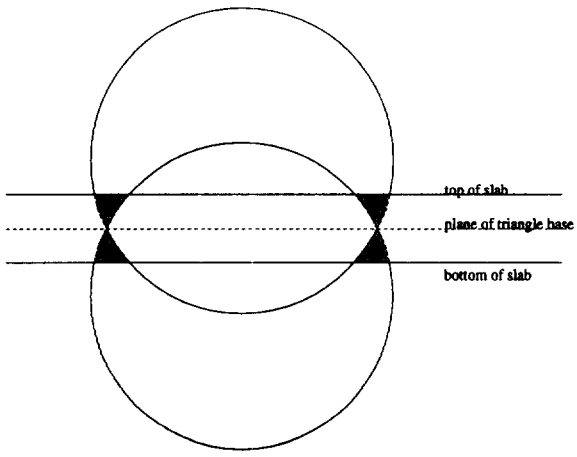

Figure 2: The fourth point of the sliver must lie in the grey region near the circumcircle of the base triangle.

when we randomly pick a point within 0.5 units of the circumcenter, we will eventually find one that forms no slivers.

First, observe that there are some slivers that we don't have to worry about. Any sliver that has a large circumsphere (with radius > 1) is caught later in the algorithm and the corresponding tetrahedron is destroyed. With this observation, we define a *sliver* as used in Algorithm C to be a tetrahedron that has a unit-radius circumsphere and that has height less than ϵ where ϵ is a value to be determined later in the proof. Note that new tetrahedra all use our new near-circumcenter point as their apex. We refer to the remaining three vertices of a new tetrahedron as the *base triangle* for for that tetrahedron. Height is measured from the plane of this base triangle.

The remainder of the proof is based on the observation that slivers have a very special shape. Given a base triangle, the apex vertex of a sliver can't be just anywhere. To be a sliver, the apex vertex must lie almost within the plane of the base triangle. Thus, a base triangle determines a slab (i.e., a thickened plane) of thickness 2ϵ with the property that an apex vertex chosen outside the slab cannot possibly form a sliver with the base triangle. Furthermore, the apex vertex must lie near the circumcircle of the base triangle. The exact shape is a ring with an hourglass cross-section; see Figure 2. Thus each potential base-triangle has just a small volume of space (called the *disallowed region*) where a forth vertex could make a sliver with that base-triangle.

Note that there are just a finite number of potential base triangles and consider the volume covered by the disallowed regions of all the potential base triangles. By choosing ϵ sufficiently small, we can ensure that all of these disallowed regions fail to completely cover the *picking-sphere* (the inner sphere of radius 0.5 about the true circumcenter). We can force some fraction of the volume within the picking-sphere to be uncovered; thus, the picking process must halt in constant expected time. □

Currently, the value of ϵ needed for the above proof is about 0.00058. This value is a fairly crude estimate and follows from a number of rather crude bounds on geometric quantities. Several of these bounds are multiplied together and/or cubed; thus any over-estimation can become quite significant. Rough experimental evidence indicates that a better estimate is $\epsilon = 0.1$. Better estimates should be available when the algorithm is fully implemented.

4 Creating a Practical Algorithm

There are three problems that need to be resolved: (1) as presented in Algorithm C, the boundary is completely meshed into *very* nice triangles during an unspecified (and possibly impractical) preprocessing step, (2) if initially there are some slivers whose vertices are all on the boundary then these slivers can remain in the final mesh, and (3) Algorithm C produces a constant density mesh while practitioners generally want control of mesh density, with small tetrahedra in "interesting" areas for accuracy and large tetrahedra elsewhere for computational efficiency. For the third problem, a relatively small modification of the algorithm allows varying density, but this leads to large changes in the bounds used in the proofs. We briefly discuss resolutions for the first two problems. Full explanations, with proofs, would significantly lengthen this paper.

Boundary preprocessing is unnecessary; the boundary meshing can be done during the tetrahedral meshing. The idea is to split a boundary triangle by adding its circumcenter whenever it becomes clear that a boundary triangle is too large.

This leaves the problem of how to get rid of slivers whose vertices are all boundary vertices. The idea here is to allow approximate circumcenters of boundary triangles, picking a random point near the true circumcenter. By adjusting ϵ appropriately one can show that disallowed *areas* along the boundary can be avoided using a constant expected number of picks.

References

[BE92] M. Bern and D. Eppstein, Mesh Generation and Optimal Triangulation, *Computing in Euclidean Geometry*, edited by F. K. Hwang and D.-Z. Du, World Scientific, 1992. Also appears as Tech Report CSL-92-1, Xerox PARC, March 1992.

[BEG94] M. Bern, D. Eppstein, and J. R. Gilbert, Provably Good Mesh Generation, *Journal of Computer and System Sciences* 48, 384–409, 1994. An earlier version appears in *Proceedings of the 31st IEEE Symposium on the Foundations of Computer Science*, 231-241, 1990.

[Che89] L. P. Chew, *Guaranteed-Quality Triangular Meshes*, Department of Computer Science Tech Report TR 89-983, Cornell University, 1989.

[Che93] L. P. Chew, Guaranteed-Quality Mesh Generation for Curved Surfaces, *Proceedings of the Ninth Symposium on Computational Geometry* (1993), ACM Press, 274–280.

[MV92] S. A. Mitchell and S. A. Vavasis, Quality Mesh Generation in Three Dimensions, *Proceedings of the Eighth Annual Symposium on Computational Geometry*, 212-221, ACM Press, 1992. Full version appears as Department of Computer Science Tech Report TR 92-1267, Cornell University, 1992.

[Ru93] J. Ruppert, A new and simple algorithm for quality 2-dimensional mesh generation, *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, 83–92, 1993.