Fast Sizing Calculations for Meshing

Gary L. Miller Todd Phillips Don Sheehy

Carnegie Mellon University
September 16, 2008

Abstract

Provably correct algorithms for meshing difficult domains in three dimensions have been recently developed in the literature. These algorithms handle the problem of sharp angles ($< \pi/2$) between segments and between facets by constructing protective collars around these regions. The collars are approximately sized according to the *local feature size* of the input. With the eventual goal of developing time-efficient algorithms for the same mesh generation problems, we give a method for estimating the feature size of a 3D piecewise-linear-complex of size n on domain Ω in time $O(n \log \Delta + m)$, where Δ is the *spread* of the input. The linear term $m \in O(\int_\Omega 1/\operatorname{lfs}^3)$ is bounded above by the output size of a quality generated mesh. Our algorithm is based on early termination of the Sparse-Voronoi-Refinement (SVR) meshing algorithm, which is not guaranteed to terminate in the presence of sharp angles.

1 Local Sizing Functions

There are several algorithms [2, 4, 1, 5] for computing quality meshes of three-dimensional PLCs that all use some form of "collars" as protective regions around sharp angles. The sizing for these collars is determined by some variant of the following two functions.

The first is the **local feature size** ($\operatorname{lfs}(x)$), the smallest x-centered ball that intersects two disjoint features of $\mathcal{C}$, originally due to Ruppert [6]. The second is the **gap-size** ($\operatorname{gs}(x)$), the smallest x-centered ball that intersects two features of $\mathcal{C}$, one of which does not contain x . The lfs is bounded away from zero everywhere and is 1-Lipschitz. The gs may be very discontinuous, but it is 1-Lipschitz along the interior of any feature. The definitions imply that $\operatorname{gs} \leq \operatorname{lfs}$ everywhere.

Meshing algorithms wish to know these values at corners and along creases. Some algorithms take these functions as given, naively requiring brute-force computations taking $\Omega(n^2)$. Later algorithms calculate approximations to sizing procedurally. This works well in practice, however the methods used do not have good runtime guarantees. Our contribution is a work-efficient procedure for approximating sizing functions.

The goal of this research is to develop a provably efficient algorithm for generating 3D meshes of arbitrary PLCs. The ability to estimate sizing functions quickly is an important first step.

We will return a pointwise sample of the domain with the exact values of lfs and gs at every point. The Lipschitz conditions will provides enough smoothness to give appropriate guarantees on the quality of our sample.

As desired by most of these algorithms, our sample includes as a subset all the corners of of the PLC and a good sample along all the input segments.

2 SVR Algorithm

The problem of sizing estimation is in some sense identical to mesh generaiton, a proper mesh provides a sizing estimater. It is no coincidence then that our size estimation procedure is obtained by modifying the work-efficient SVR meshing algorithm [3].

The SVR algorithm iteratively maintains a Voronoi diagram V of inserted points. Additionally, it maintains queue of uninserted points and protective circumballs around unrecoved input features. The current Voronoi diagram contains bi-directional pointers maintaining the intersection of the Voronoi diagram with the queue.

Secondly, $\mathcal{V}$ is always a τ**-well-spaced** Voronoi diagram for some $\tau > 1$. If V is a Voronoi cell of vertex v in $\mathcal{V}$, let R_v be the radius smallest v-centered ball con-

taining V, and let r_v be the largest contained ball. V is τ-well-spaced if $R_v/r_v < \tau$.

The iterative SVR algorithm proceeds as a work queue, processing events until termination. Events can be prioritized according to the geometric size of the cells being refined (approximately-smallest-first), so that when a vertex v is inserted, there is a constant β such that any other vertex u has $R_u \geq \beta R_v$. The algorithm terminates for a non-sharp PLC with total work $O(n \log \Delta + m)$.

3 SVR with Feature Sizes

We will add to SVR, an invariant that every vertex at any point during the algorithm knows it's exact lfs and gs. These will be calculated explicitly when a vertex is initially added to the mesh. When a vertex v is added, the termination proof of SVR gives a constant C such that the new Voronoi cell is at least feature size, that is $CR_v \geq \mathrm{lfs}(v) \geq \mathrm{gs}(v)$.

Let $R = \min\{R_v, \min_f |v - f|\}$ where f is any feature in the new Voronoi cell V. R is then clearly a lower bound on the gs. We can then calculate the function values at v by searching out all features within a ball of radius CR. Because we are refining smallest first, we need only look at a constant number of nearby Voronoi cells.

There may be a very large number of uninserted features in these nearby cells, and the work to determine lfs and gs exactly will be linear in this number. We will amortize this work and charge it to the features being queried. An input feature is only queried when there is a large empty region (the new Voronoi cell V) relatively nearby (constant C). Packing arguments identical to those bounding the work for point location in [3] can show that this implies each input feature will be charged at most $O(\log \Delta)$ many times. Thus, the additional new calculations can be added to SVR without any new asymptotic work.

4 Terminating SVR

Without modification, the previous scheme is correct, except that it will not terminate (because SVR will not terminate) in the presence of sharp angles.

Since we know the sizing at the center of all our current Voronoi cells, we can determine the quality of our sample by how much smaller the current Voronoi cell is than the actual local feature size.

The items on the work queue for SVR are in general one of two types. CLEAN moves that attempt to improve the quality of mesh vertex distribution, and BREAK moves that attempt to recover unresolved features.

To ensure termination, we add the following additional rule to the iterative SVR algorithm: - If any Voronoi cell $V \in \mathcal{V}$ centered on at point v is small enough, let i be the dimension of the lowest dimensional feature containing v. Ignore any BREAK moves on the queue of dimension $> i$ that intersect V.

The argument for termination follows because of the work removed by the new rule. Normally, SVR could recurse indefinitely when BREAK moves mutually create each other on adjacent features, as happens at sharp angles. With the new rule, once the corner where these features meet becomes disjoint-empty, the ping-pong effect of mutual encroachment is circumvented.

References

[1] S.-W. Cheng and S. hung Poon. Three-dimensional delaunay mesh generation. *Discrete Comput. Geom*, 36:419–456, 2006.

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

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

[4] S. E. Pav and N. J. Walkington. Robust Three Dimensional Delaunay Refinement. In *Thirteenth International Meshing Roundtable*, pages 145–156, Williamsburg, Virginia, Sept. 2004. Sandia National Laboratories.

[5] A. Rand and N. Walkington. 3d delaunay refinement of sharp domains without a local feature size oracle. In *17th Internation Meshing Roundtable. (To Appear.)*, 2008.

[6] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993).