Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Voronoi Meshes with Prescribed Boundaries

S. Davis Herring[a,*], Brian Jean[a]

[a]Los Alamos National Laboratory, Los Alamos, NM 87544, USA

Abstract

The common technique of 2D mesh generation based on Voronoi diagrams uses a boundary polygon (sometimes taken to consist of generators for the diagram) and a set of internal generators. The mesh on the boundary, however, is not an input but a consequence of the diagram's intersection with it. To support constructing a conforming multi-block mesh, either to incorporate conformal boundaries or to use other mesh constructions alongside a Voronoi region, the Voronoi diagram must be engineered to obtain a desirable boundary mesh. Similarly, relaxing a portion of an existing mesh demands retaining the connectivity to the surrounding mesh. This note describes an algorithm for arranging for a prescribed mesh on the boundary of a Voronoi region that has been implemented in a production meshing tool at LANL.

Keywords: Voronoi diagram; hybrid meshing; mesh relaxation

1. Introduction

Planar Voronoi diagrams have a number of features desirable for an automatically generated mesh[1]:

1. all cells are always convex
2. all nodes have degree 3 (for generators in general position)
3. average number of edges per cell is ≤ 6
4. small changes in generator position produce small changes in the mesh (continuous geometric changes or, occasionally, local topological changes)[2]
5. number of internal generators provides direct control of cell count
6. construction is efficient — $O(n \log n)$
7. Lloyd's algorithm[3] provides natural smoothing

However, using a Voronoi diagram in a bounded region is rather less natural. Simply restricting (clipping) a diagram to a non-convex region can produce concave or even disconnected Voronoi cells; defining the boundary in terms of a polygon of generators and requiring distances to be measured only within the region (producing a *constrained Voronoi tessellation*[1]) avoids those problems but partitions the boundary into edges that are frequently collinear and whose number is hard to control.

[*] Corresponding author. Tel.: +1-505-667-1337 ; fax: +1-505-665-3470.
E-mail address: herring@lanl.gov

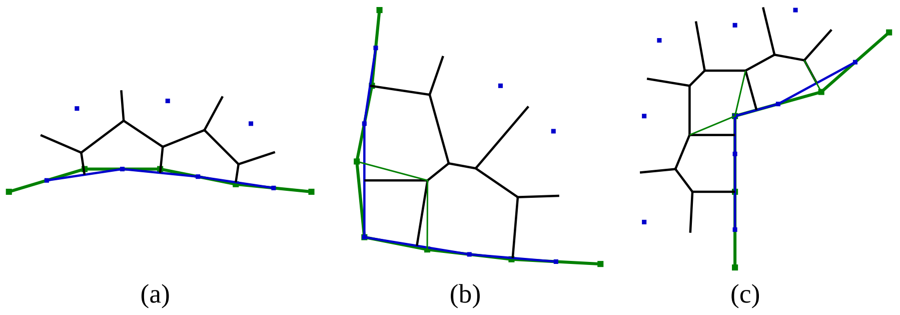

Fig. 1. Strategies for *(a)* side, *(b)* end, and *(c)* corner vertex types. Thick green lines are the boundary; blue points and lines are the generators and the generator polygon; black lines are the mesh. Thin green lines show significant adjustments made to connect the mesh to the boundary.

When a mesh is to be defined only partially from a Voronoi diagram, either because another mesh type is desired for an adjoining region or because a preexisting mesh must be preserved there, the treatment of the boundary becomes critical. This note presents a simple algorithm for constructing an approximation to a Voronoi diagram that preserves a specified boundary mesh. This algorithm's implementation in an existing hybrid meshing library developed at Los Alamos National Laboratory (LANL) is also discussed.

2. Algorithm

Given

1. interior generators
2. boundary polygon
3. a *type* for each vertex (to be discussed)

we seek to create a mesh that is a Voronoi diagram except for small geometric and topological changes required to respect the boundary vertices (and so allow connection to another mesh). We will use the terms *vertex*, *facet*, and *region* in discussing the boundary polygon and *node*, *edge*, and *domain* for the (Voronoi) mesh. By "nodes" we particularly mean those on the domain boundary that are not generators.

The algorithm is based on the idea that a constrained Voronoi tessellation always produces a node between each pair of adjacent boundary generators. In outline, we

1. construct generator polygon whose sides correspond to vertices,
2. compute constrained Voronoi tessellation from that polygon and the interior generators,
3. remove boundary generators from the mesh,
4. and replace nodes with corresponding vertices.

The basic strategy for Step 1 is to place a generator at the midpoint of each facet. This arrangement is illustrated in Fig. 1(a). The domain differs from the region (*i.e.*, the blue and green lines do not coincide), but the nodes closely approximate the vertices. To be precise, each vertex's node is displaced one quarter of the way along the diagonal of the parallelogram having the vertex's two incident facets as sides.

The mesh boundary typically consists of pairs of collinear edges through a node. Step 3 replaces the generator's two incident edges with one longer edge that closely approximates a facet. The pattern of straight and (in general) non-straight interior angles is thereby shifted by half a cell to align with the boundary polygon. The distortion in Step 4 therefore is frequently invisibly small.

However, it is not desired for all boundary vertices to receive one edge each from (the interior of) the mesh. The number desired depends on the *vertex type* familiar from the paving technique: a typical *side* vertex receives one (interior) edge, an *end* receives none (and so has merely the two facets), and a *corner* receives two[4]. (The number

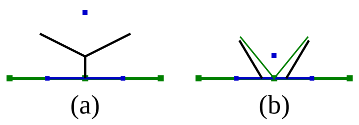
Fig. 2. *(a)* An internal generator near a vertex. *(b)* The internal generator is close enough to require resolving more than one edge to a single vertex. Colors as in Fig. 1.

of cells using a vertex is one greater than the number of interior edges.) The vertex types are the final input required for the algorithm.

Simple modifications to the generator placement address these other two cases, as illustrated in Fig. 1(b) and (c). The two generators beside an end vertex are coalesced onto that vertex, thus avoiding the generation of a edge between them. A corner vertex is added as an additional generator, producing a edge on either side of it that we may connect to the vertex. In these cases, for a vertex spacing of h an error of about $h/4$ is committed in the placement of the boundary vertices adjacent to the vertex generator; the thin green lines in Fig. 1 show the edges as adjusted to meet the vertices.

When an interior generator is near a vertex, its Voronoi cell may lie on the boundary, as illustrated in Fig. 2. The two nodes that result are again an average distance of about $h/4$ away from the vertex; we simply replace both of them with the same vertex, creating a cell that touches the boundary at just one point. (With very closely spaced internal generators, more than one may be near the same vertex, producing three or more edges that are all assigned to that vertex.)

3. Implementation

This algorithm has been implemented in Altair—the multi-block 2D meshing component of Ingen, the simulation setup Python library developed at LANL—as one technique for constructing unstructured meshes. Each mesh block (structured or no) in Altair is constructed after specifying a 1D mesh on all its boundaries (which are often material interfaces). The single, shared description of the boundary between two blocks avoids redundancy in describing them and guarantees them conforming meshes.

For a Voronoi block, the user specifies the boundary as a set of connected faces. The angle between successive edges at each face boundary is used to select a vertex type. In the current implementation, vertex types are set strictly by angle; the user may override an end or corner type by joining the incident faces, but there is no way to compel another vertex type. The generator polygon is then constructed and the Voronoi diagram is computed by the library ShaPo (developed separately, by KitWare).

Fig. 3 shows an example result from this process. The Voronoi block was created from generators initially chosen at random within the semicircle and then smoothed with 100 iterations of Lloyd relaxation. More complicated shapes do not admit so trivial a scheme for generator selection; a deterministic algorithm to pick plausible internal generators for more general regions without user intervention is under development.

4. Issues

Several issues remain to be addressed to make this capability general and robust, mostly involving generators very close to the boundary:

1. When the two facets incident on a side vertex are not parallel, the situation in Fig. 4(a) may arise, where the internal generator is well within the region but is outside the domain. It is still possible to construct the Voronoi diagram, but the cell associated with the aberrant generator may be very small or nonexistent. Our current implementation treats this case as an error that must simply be avoided by the user.

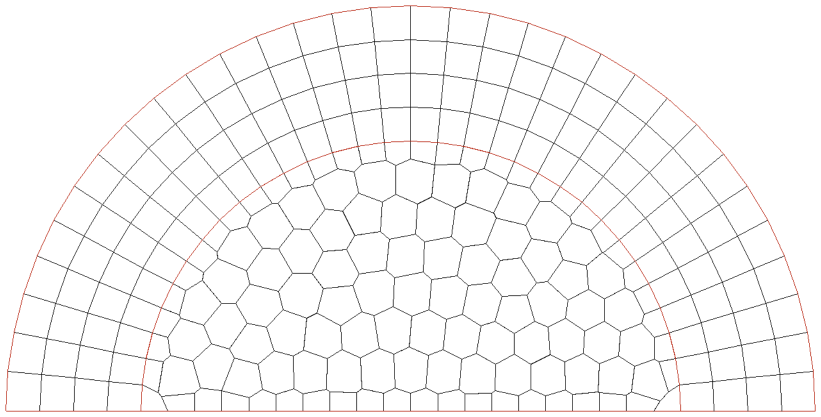

Fig. 3. Mesh with one structured and one Voronoi block. Red edges are block boundaries.

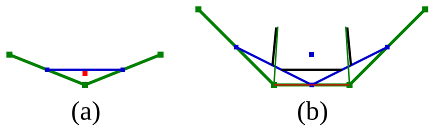

Fig. 4. *(a)* An internal generator (red) is within the region but outside the domain. *(b)* Degenerate cell caused by a generator very close to a facet; the red edge is the horizontal black edge, adjusted into coincidence with the facet. Other colors as in Fig. 1.

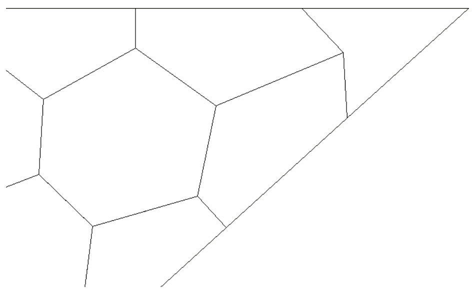

Fig. 5. A concave cell produced in a small angle by the adjacent boundary generators.

2. In the vicinity of an end vertex, the domain very closely approximates the region because of the boundary generator created at the vertex. However, a nearby interior generator can case the boundary generator's cell to become concave or even overlap the interior cell after edge adjustment. A similar effect can occur merely because of a small angle between facets, as illustrated in Fig. 5. Straightforward mesh smoothing applied after fitting the mesh to the boundary may be able to resolve these issues.

3. An internal generator that is instead very near a facet can produce the situation shown in Fig. 4(b): both pairs of added nodes are coalesced as in Fig. 2, but because two of those are the ends of one edge, that edge becomes coincident with the facet between the two vertices onto which the nodes were moved. Such a cell might simply be removed, but this loses an attractive property of the method that the number of cells is precisely controlled by the number of vertices and their types.

4. The strategies described in Fig. 1 do not cover reversals, where four cells should share a vertex; it may be possible to add a pair of boundary generators with a very slight separation at the reversal vertex, but the numerical behavior may be undesirable in that case.

Acknowledgements

Funding for this work (and for ShaPo) was provided by the Lagrangian Hydrodynamics Research project in the Advanced Simulation & Computing program.

References

[1] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Wiley Series in Probability and Statistics, second ed., Wiley, 2009.

[2] D. Reem, The geometric stability of voronoi diagrams with respect to small changes of the sites, in: Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry, SoCG '11, ACM, New York, NY, USA, 2011, pp. 254–263. doi:10.1145/1998196.1998234.

[3] Q. Du, V. Faber, M. Gunzburger, Centroidal voronoi tessellations: Applications and algorithms, SIAM Review 41 (1999) 637–676.

[4] T. D. Blacker, M. B. Stephenson, Paving: A new approach to automated quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (1991) 811–847.