Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2016) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable (IMR25)

Constrained mesh generation on shared boundary curves

S. Davis Herring[a],*

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

Abstract

In multidimensional conformal meshing, the mesh on a curve may be subject to multiple constraints derived from the various higher-dimensional geometric objects that include it as a boundary. One common kind of constraint is the hardpoint, where a vertex must be placed; another is the total number of vertices to use, which often must be consistent with other curves for reasons of mesh topology. This note describes algorithms for satisfying these constraints, as well as a third type that prohibits the introduction of vertices on an interval, with the remaining degrees of freedom used to approximate an arbitrary goal mesh according to one of two metrics. The algorithms have been implemented in a production meshing tool at LANL.

Keywords: constraint satisfaction; multi-block meshing; curve meshing

1. Introduction

Boundary curves in a conformal mesh require a vertex at each endpoint. Since an endpoint of one curve may lie amid another, and a curve may contain sharp corners that are to be resolved in the mesh, vertices may be required anywhere on a curve.

In structured multi-block meshing, a curve typically must receive the same number of vertices as the one opposite it on each block. Even with the flexibility afforded by specialized techniques such as dendritic meshing, sometimes a constraint persists involving perhaps half or double the number of edges on an opposite curve. Such constraints propagate along mesh lines, and so a block may come to require one number of vertices on the entirety of a face and another (smaller) number on a portion of it. (The portion may in fact have a particular mesh specified to properly capture a boundary layer or reflect a symmetry of the problem.)

This note presents algorithms for minimally perturbing a proposed mesh on a curve to satisfy constraints of these types, including that the number of vertices is fixed. Minimal perturbation may be defined in terms of least-squares distance (arc length along the curve) between the vertex positions proposed and adopted, or in terms of variation in the edge lengths on the curve relative to those in the proposed mesh. These algorithms' implementation in an existing multi-block meshing library developed at Los Alamos National Laboratory (LANL) is also discussed.

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

2. Problem

Formally: given

1. a *proposed distribution* of parameters $\{t_i\}_{i=1}^N$,
2. a sequence of *hardpoints* $\{h_j\}_{j=1}^H$ $(2 \le H \le N)$, including the endpoints of the curve, and
3. a sequence of *locked intervals* $\{e_k \in \mathbb{Z} \cap [1, H)\}_{k=1}^L$

find a *final distribution* $\{t'_i\}_{i=1}^N$ such that $t'_1 = h_1$, $t'_N = h_H$, $\nexists i, k \ni t'_i \in (h_{e_k}, h_{e_k+1})$, and either

$$\sum_{i=1}^{N}(t'_i - t_i)^2 \tag{1}$$

is minimized (the *snapping* case), or the edge length on each hardpoint interval is multiplied by a number as close as possible to unity (the *smooth* case). All these sequences are strictly increasing. Typically the parametrization is equal-arc-length, so that the distance defined in (1) is (proportional to) the arc length between t_i and t'_i.

In either case, we define the final distribution in terms of a strictly increasing *mapping* $\{m_j \in \mathbb{Z} \cap [1, N]\}_{j=1}^H$ (with $m_1 = 1$ and $m_H = N$):

$$t'_i := \begin{cases} h_j & \text{if } \exists j \ni m_j = i \\ t_i & \text{else} \end{cases} \tag{2}$$

in the snapping case, or in the smooth case

$$t'_i := h_j + (t_i - t_{m_j})\frac{h_{j+1} - h_j}{t_{m_{j+1}} - t_{m_j}} \tag{3}$$

with the largest $j < H \ni m_j \le i$. The snapping treatment preserves the position of as many vertices as possible, simplifying (1) as

$$\sum_{j=1}^{H}(h_j - t_{m_j})^2 \tag{4}$$

In the smooth case, in general all (internal) vertices are moved in order to avoid large ratios between the edge lengths on either side of a hardpoint. It is the absolute value of the logarithm of the fraction in (3) that is minimized.

3. Algorithms

3.1. Smooth case

The smooth case is the simpler. Define the function

$$\tilde{t}(x) := t_{\lfloor x \rfloor} + (x - \lfloor x \rfloor)(t_{\lceil x \rceil} - t_{\lfloor x \rfloor}) \tag{5}$$

which extends the function $t : \mathbb{Z} \to \mathbb{R}$ by linear interpolation. Then define quasi-indices for the hardpoints $\tilde{h}_j := \tilde{t}^{-1}(h_j)$ and (for $1 \le j < H$) a *share* of the proposed edges for each hardpoint interval $s_j := \tilde{h}_{j+1} - \tilde{h}_j$. We then have $n := N - 1$ edges that must be distributed among $b := H - 1$ hardpoint intervals, at least one each (because no vertex can appear at both ends of an interval). This *apportionment problem* is precisely that solved by the Huntington–Hill method[1]. Before applying it, the share for each locked interval is replaced with some vanishing ε, causing it to be assigned only the one edge guaranteed by that method to every recipient and to therefore contain no vertices.

The Huntington–Hill method minimizes the *relative difference* between the coefficients applied to the edge length on each pair of hardpoint intervals. The relative difference of a and b is here defined as

$$\max\left(\frac{a}{b}, \frac{b}{a}\right) - 1 = \exp\left|\log\frac{a}{b}\right| - 1$$

so the minimization may equivalently be said to be of $|\log a/b|$ as stated previously. A trivial implementation based on a heap apportions n objects into b bins in $O(b + n \log b)$ time; no account need be taken of the reduction in $\sum_j s_j$ for locked intervals. The result is a sequence of allocations $\{a_j\}_{j=1}^{b}$ that sums to n; the mapping is simply its cumulative sum (starting with 1).

3.2. Snapping case

The snapping case is more complex. Brute force is unworkable: the mapping is a combination of the proposed vertices, with a number of theoretical possibilities that can easily be $\binom{1000}{10} \approx 2.6\times10^{23}$. Most are inadmissible because they produce a non-monotonic final distribution, but extreme cases such as $H \approx N$ or the presence of a large locked interval make it difficult to categorically reject large portions of the search space. The number even of admissible mappings is exponential in H whenever the hardpoints are separated by pairs of proposed vertices.

3.2.1. Without locked intervals

Consider first the case $L = 0$; we will proceed by considering subsets of the hardpoints. First we establish a lemma about those sets:

Lemma. *If the optimal mapping for a set of hardpoints $\{h_j\}$ (with $L = 0$) is $\{m_j\}$, then for a set $\{h'_j\} \supset \{h_j\}$, the optimal mapping $\{m'_j\} \supset \{m_j\}$.*

Proof. We proceed by induction, adding hardpoints one at a time. For each added hardpoint h', if the nearest proposed vertex $t_{i'} \notin \{m_j\}$, the optimal mapping is simply augmented by that vertex.

Otherwise, consider the longest *run* of consecutive m_j values that contains i'. Note that increasing any one of the h_j values associated with the run $R \subset \mathbb{Z} \cap [1, H]$ monotonically increases the cost difference of (4)

$$\sum_{j\in R}(h_j - t_{m_j})^2 - \sum_{j\in R}(h_j - t_{m_j+1})^2 \tag{6}$$

(so long as $\max R < H$) or else detaches $h_{\max R}$ from the run.

The one or two *flanking* vertices adjacent to the run must each lie outside the interval between the first and last hardpoint in the run, because otherwise it would improve the cost for the run to (separate it and) use that vertex instead. (There is but one if the run includes the first or last vertex; there is at least one because $H < H' \leq N$.)

Consider the auxiliary problem(s) produced by adding a hardpoint (to the original set) at the position of a flanking vertex; plainly it is optimal to map that vertex onto it without disturbing the existing mappings. Since the true h' is closest to a vertex between the flanking vertices, it is between them itself. The true run $\{h'_j\}_{j\in R'}$ can therefore be produced by continuously deforming any auxiliary problem, moving hardpoints only in one direction.

When there are two auxiliary problems, they have adjacent answers in terms of the position of the run of vertices to be used; the optimal result for the true augmented problem must be one of them. If there is but one, it is already extremal and the continuous deformation can produce no other answer. In all cases, the new mapping is a superset of the old. □

The greedy, $O(H \log N)$ algorithm follows the proof exactly: for each hardpoint, find the closest proposed vertex. If it is unused, map it (tentatively) to that hardpoint. Otherwise, if the contiguous run of used vertices containing it can be extended in just one direction, do so. Otherwise, compare the cost (4) associated with each extension to determine the correct direction of growth, even though all the mappings may be subsequently modified in considering further hardpoints. (Nearby runs can grow together, in which case they coalesce for the remaining steps of the algorithm.)

3.2.2. With locked intervals

It remains to treat the case $L > 0$. It is useful to rephrase the locked-interval constraint in terms of the mapping: every proposed vertex that lies within a locked interval must be selected by the mapping. (Exactly the selected vertices are replaced in the final distribution; the hardpoints that replace them are not in any locked interval. A vertex that lies exactly on a hardpoint already must be selected, if only for that hardpoint, because the final distribution must be strictly increasing.) The contiguous run of vertices within a locked interval must be mapped onto a contiguous run of hardpoints; to avoid a non-monotonic final distribution, any proposed vertices between the first and last such hardpoints must also be selected. All included hardpoint intervals therefore may as well be locked intervals: the original locked interval has in effect *grown*, and the process repeats whenever new vertices come to be included.

Starting from some mapping of the vertices in a locked interval, the effective growth of the interval can easily overrun the array of hardpoints. That assumed mapping is thereby known to be inadmissible; in some cases, no mapping is possible for a locked interval and the overall problem has no solution. In practice, this occurs when the proposed mesh for a curve is more refined than the existing mesh in a region; every vertex displaced by the fixed distribution displaces another until there are no more valid locations for them. (The smooth case does not fail so suddenly; the mesh on the unconstrained portions of the curve is merely compressed to make room for the fixed, lower-resolution section.)

When it does not prevent the solution of the problem altogether, the inadmissibility of certain mappings for a locked interval instead provides the means for solving it efficiently in most cases. The interval of admissible mapping offsets for each locked interval is constructed by two binary searches, each testing for runaway growth in a single direction. Then, for each element of the Cartesian product of those offset intervals (for which none of the mappings overlap), the $L = 0$ algorithm is executed with the mapping seeded with the mappings for the locked intervals; the result with the lowest cost (4) is retained.

While that algorithm may not produce optimal results in each case (because the optimality precondition for the lemma is not satisfied), it will do so when the mapping for the locked intervals is (accidentally) optimal; the result from that case will of course defeat any erroneous results.

The cardinality of this Cartesian product can also be exponential (in L); while the algorithm can in fact take exponential time, two mitigating factors make it usable in practice. First, the alternate interpretation of locked intervals in terms of requiring the mapping of all contained vertices allows the coalescence of adjacent locked intervals, reducing the effective L typically to numbers not much greater than 10. Second, it is possible to avoid considering many elements by treating them as leaves of a tree and pruning branches that produce an overlap between mappings.

4. Implementation

These algorithms have been implemented in Altair—the multi-block 2D meshing component of Ingen, the simulation setup Python library developed at LANL—as part of a semi-automated system for matching edge counts on opposing curves on a block. The snapping algorithm with $L = 0$ has been implemented since 2012 for copying non-uniform meshes from one curve to another that might contain hardpoints. In 2014, the smooth algorithm was added to adjust simple uniform distributions on each hardpoint interval to match an opposite curve.

Development started in 2014 on a new topology-based *parts* system for geometry development and meshing in Ingen. It can construct many block topologies based on the same parts, and so places a strong emphasis on propagating resolutions rather than requiring each mesh block's boundaries to be explicitly named and meshed. Altair can automatically create dendrites in only one direction per block, so the resolution propagation functions support copying meshes (or simply their vertex counts) in the other direction (or in both to avoid dendrites altogether). This year, the oft-requested feature was added of copying onto a single (composite) boundary from both sides, necessitating locked intervals. Clients are currently experimenting with both variations of the complete algorithm that supports them.

5. Extensions

There are several obvious extensions to consider, especially for the smooth case:

1. Other parametrizations: in the smooth case, they affect the rescaling in (3) as well as the apportionment.
2. (3) can be refined by using any monotonic interpolation, which could be used to avoid sudden changes in edge lengths at a hardpoint.
3. (5) can be similarly refined to measure shares in a smooth fashion.

Acknowledgements

Funding for this work was provided by the Simulation Setup project in the Advanced Simulation & Computing program.

References

[1] E. V. Huntington, The mathematical theory of the apportionment of representatives, Proceedings of the National Academy of Sciences of the United States of America 7 (1921) 123–127.