Parallel Smoothing for Grid-Based Methods

Steven J. Owen

[†]Sandia National Laboratories, Albuquerque, New Mexico, U.S.A.
sjowen@sandia.gov

Summary. We focus on the problem of mesh quality improvement for parallel grid-based hex meshing from an implicit geometry representation. We outline a practical set of tools that utilize a combined Laplacian and optimization approach using Jacobi-based algorithms to improve element quality and achieve parallel consistency.

Key words: grid-based, overlay grid, hexahedral mesh generation, parallel meshing, smoothing, optimization, mesh quality

1 Introduction

Generation of hexahedral meshes using grid-based approaches hold tremendous promise for full automation and scalability. Among the challenges faced, however is the critical mesh quality issue. Although grid-based methods can be general purpose and fully automatic, typical results can often yield unusable elements near the boundaries. In this work we present a practical approach to mesh quality improvement through smoothing of hexahedral meshes. We also focus, on the parallel problem and the challenges of smoothing in a distributed environment.

To begin, we utilize the distributed meshing procedure, described in [1] and limit our application to volumetric domains bounded by implicit surface representations. Numerical procedures relying on a geometric decomposition of the domain often generate small differences in the solution, based upon the number of processors used or the selected decomposition strategy. For this application, parallel consistency where the exact same result, to machine accuracy, is expected, regardless of the number of processors used or decomposition strategy employed.

This work advocates and leverages the excellent work of many authors and practitioners in this field [2]-[6]. For our purposes, we define acceptable quality

[†]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000

in terms of the minimum scaled Jacobian, J_s of the element. The eight scaled Jacobian values, $(J_s)_I$ at the nodes of a hex can be computed by taking the determinant of its three ordered normalized edge vectors $E_{i,j,k}$ as illustrated in figure 1 and equation (1). The scaled Jacobian metric for a hex is then taken as the minimum of the eight determinant calculations as in equation (2).

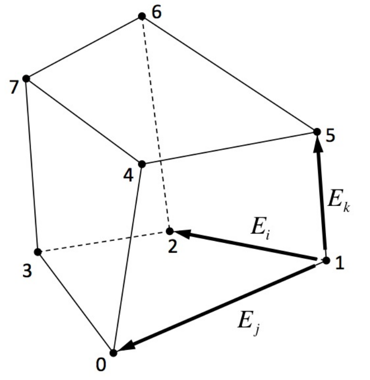

Fig. 1. Ordered edges E_i, E_j, and E_k are used to compute Scaled Jacobian at 1

$$(J_s)_I = det\left\{\hat{E}_i \hat{E}_j \hat{E}_k\right\}^{\top} \tag{1}$$

$$J_s = min\left((J_s)_I, I = 0, 1, ...7\right) \tag{2}$$

A value of $J_s = 1.0$, indicates an ideal element where all angles are precisely 90 degrees, however a value of $J_s \leq 0.0$ normally indicates an unacceptable element for computational purposes. Initial projection of nodes to interfaces and insertion of the boundary hex layer can result in many elements where $J_s \leq 0$ or *inverted*. Depending on the requirements of the analysis, an acceptable value for scaled Jacobian can vary, but normally a value of $J_s \geq 0.2$ is permissible. Smoothing methods are intended to increase the value for J_s for all elements in the mesh to acceptable standards for computation.

2 Algorithm

Figure 2 summarizes the procedure we propose in this work. We begin with a distributed grid-based mesh that we define as Ω_M that contains the set of mesh entities, $M^i, i = 0, 1, 2, 3$ as nodes, edges, faces and hexes. We also begin with an associated implicit geometry definition Ω_G, containing geometric entities, $G^i, i = 0, 1, 2, 3$, vertices, curves, surfaces and volumes respectively. Individual processors within the distributed domain, Ω_M are defined as Ω_M^p that have been established to include ghosted nodes and elements at their boundaries, as outlined in [1].

Following the establishment of mesh and geometry, parallel communication is first initialized for ghosted nodes, followed by several iterations of smoothing operations. The smoothing operations consist of curve and surface smoothing operations that project to an implicit geometry, followed by volume smoothing operations. Initial smoothing iterations smooth all nodes using a fast Laplace method, while later iterations, focus only on hex elements that fall below a designated threshold using an Optimization approach.

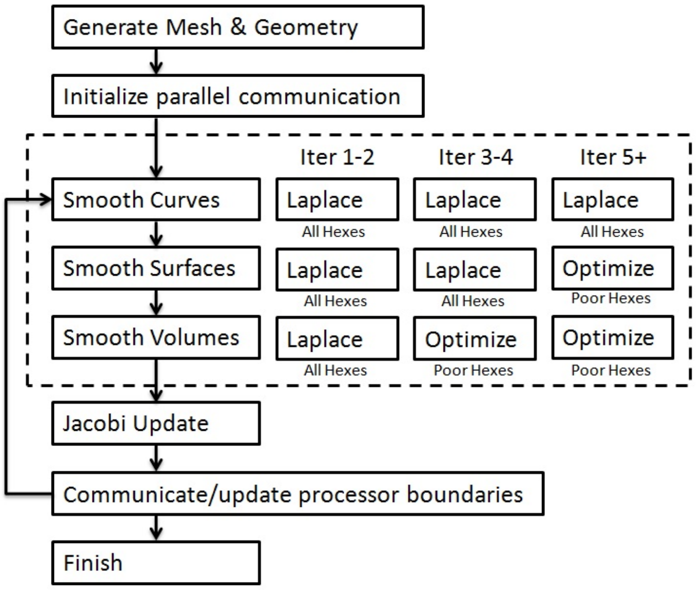

Fig. 2. smoothing procedure

Since we require a parallel-consistent result, differences in the node locations resulting from a particular distribution strategy are unacceptable for our application. To meet this objective, we utilize a Jacobi-based approach for smoothing, where initial node locations are used in the optimization procedure. Node locations are only updated after an entire pass through the nodes has been completed.

As outlined in figure 2, this work depends heavily on initial Laplacian smoothing to improve node locations. In practice, applying only two iterations of Jacobi-based Laplacian smoothing, appears to correct the majority of inverted elements. This is followed by targeted optimization to those nodes near elements that fall below a threshold quality metric. We choose $J_s \leq 0.2$ as the threshold criteria and expand one layer of elements to include additional nodes in the smoothing domain.

For optimization, to facilitate parallel-consistent Jacobi-based smoothing, we propose a simple node-by-node optimization method that attempts to optimize the value of minimum scaled Jacobian at a node as outlined below:

1. Compute the minimum scaled Jacobian J_s from the hexes attached to node M_i^0. Use this as the objective function we are optimizing.
2. Compute the numerical gradient, ∇J_s by offsetting the location of M_i^0 in the positive x, y, and z directions a small value, ε.
3. Find an improved J_s by incrementally moving M_i^0 in the direction ∇J_s

These steps are repeated until a convergence criteria is achieved or a maximum number of iterations has been reached. For our purposes, 2 or 3 iterations were sufficient to raise the mesh quality to a computable range. Although the basic approach does not deviate significantly from standard optimization techniques, some significant differences are noted.

The objective function used for optimization is the scaled Jacobian at the node from equation 1. Since only the node will be in motion, we can neglect the Jacobian at other nodes of the adjacent hexes. The objective function is therefore:

$$J_s = min\left((J_s)_I, I = 0, 1, ...n_{hex}\right) \tag{3}$$

where $(J_s)_I$ is the scaled Jacobian at the node for the I^{th} adjacent hex and n_{hex} is the number of adjacent hexes. To control for mesh size, a size scale factor S_f on the scaled Jacobian as shown in equation 4 is also used.

$$(J_s)_I = S_f det\left\{\hat{E}_i \hat{E}_j \hat{E}_k\right\}^\top \tag{4}$$

$$S_f = \begin{Bmatrix} e_s \leq S_t, \; \frac{e_s}{S_t} \\ e_s > S_t, \; \frac{S_t}{e_s} \end{Bmatrix} \tag{5}$$

$$e_s = min(\|E_i\|, \|E_j\|, \|E_k\|) \tag{6}$$

where S_t is a target edge size.

We also point out that the objective function, J_s is continuous through zero, which allows us to optimize node locations, even when the initial $J_s < 0$. This avoids having to compute a separate optimization step for untangling as is needed by some applications.

The numerical gradient ∇J_s is computed by offsetting the location of the node in the x, y and z directions and recomputing J_s at three locations. For example the x component of ∇J_s is computed as:

$$(\nabla J_s)_x = \left\{\frac{J_s(x_0 + \varepsilon_x) - J_s(x_0)}{\varepsilon}\right\} \tag{7}$$

where x_0 is the initial location at the node and ε_x is the vector $\{\varepsilon, 0, 0\}$. The components, $(\nabla J_s)_y$ and $(\nabla J_s)_z$ are computed in a similar manner using $\varepsilon_y = \{0, \varepsilon, 0\}$ and $\varepsilon_z = \{0, 0, \varepsilon\}$.

When applying optimization to nodes on surfaces, the same steps can apply, however, the movement of the node must be restricted to the surface manifold. We can also utilize the J_s of the adjacent hexes to the surface as the objective function for the optimization. To restrict the motion of the node to the surface manifold, we can compute ∇J_s on a tangent plane to the surface. Orthogonal tangent vector T_u and T_v are computed numerically based upon the surrounding facets on the surface to the node. Offsets ε_u and ε_v are applied to compute gradients on the surface. where $\varepsilon_u = \varepsilon\hat{T_u}$ and $\varepsilon_v = \varepsilon\hat{T_v}$. Surface node optimization can then proceed in the same manner as volume optimization, except that following computation of a new x_n, the location is updated by relaxing to the surface approximation prior to computing $J_s(x_n)$.

Once a vector gradient ∇J_s is established, we can begin searching for a new location x_n that provides an improved value for J_s along the gradient direction, $x_n = x_0 + \alpha\nabla\hat{J_s}$. We choose a maximum value for α as the initial average edge length at the node and do a few iterations of a binary chop to locate an improved value for J_s. Since only the minimum $(J_s)_I$ is optimized, it is necessary that we do not severely distort other surrounding elements in the process. To do so, we count the initial number of $(J_s)_I < 0$ and do not update the node location unless the number of negative $(J_s)_I$ is maintained or improved. For our application, 4 to 6 iterations were normally sufficient.

3 Observations

The proposed method was implemented in the context of the system described in [1]. Although many test cases were employed and validated, figure 3 illustrates one typical problem we were attempting to address. In this example, the boundary layer hexes wrap a high curvature feature. We integrated the Mesquite [7] ShapeImprover method and compared with the proposed approach. In this case, and many others, we were unable to untangle and improve the quality using Mesquite. Furthermore, the Mesquite methods, from our experience, did not achieve parallel consistency. The proposed method however, was able to improve mesh quality in this case and other similar cases, as well as achieve parallel consistency.

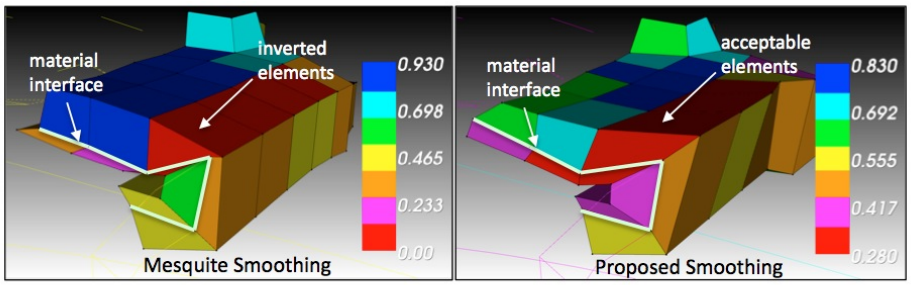

Fig. 3. Comparison of smoothing results

References

1. Owen SJ, Staten ML, Sorenson MC (2011) Parallel Hex Meshing From Volume Fractions, In: Proceedings, 20th International Meshing Roundtable 161–178
2. Canann SA, Tristano JR, Staten ML (1998) An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes, In: Proceedings, 7th International Meshing Roundtable 479–494
3. Knupp PM (2000) Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II: A framework for volume mesh optimization and the condition number of the Jacobian matrix, International Journal for Numerical Methods in Engineering, 48:1165–1185
4. Freitag LA, Jones M, Plassmann PE (1995) An Efficient Parallel Algorithm for Mesh Smoothing, In: Proceedings, 4th International Meshing Roundtable 47–58
5. Knupp PM (2001) Hexahedral and Tetrahedral Mesh Untangling, Engineering With Computers, 17:261–268
6. Knupp PM (2003) A method for hexahedral mesh shape optimization, International Journal for Numerical Methods in Engineering, 58:319–332
7. Brewer M, Freitag-Diachin L, Knupp PM, Leurent T and Melander D (2003) The Mesquite Mesh Quality Improvement Toolkit, In: Proceedings, 12th International Meshing Roundtable 239–250