# A PARALLEL VARIATIONAL MESH QUALITY IMPROVEMENT METHOD FOR TETRAHEDRAL MESHES

Suzanne M. Shontz[1] Maurin A. Lopez Varilla[2] Weizhang Huang[3]

[1]*Department of Electrical Engineering and Computer Science, Bioengineering Program, Information and Telecommunication Technology Center, University of Kansas, Lawrence, KS USA*
[2]*Husky Injection Molding Systems Ltd., Luxembourg, Luxembourg*
[3]*Department of Mathematics, University of Kansas, Lawrence, KS USA*

## ABSTRACT

There are numerous large-scale applications requiring mesh adaptivity, e.g., computational fluid dynamics and weather prediction. Parallel processing is needed for simulations involving large-scale adaptive meshes. In this paper, we propose a parallel variational mesh quality improvement algorithm for use with distributed memory machines. Our method parallelizes the serial variational mesh quality improvement method by Huang and Kamenski. Their approach is based on the use of the Moving Mesh PDE method to adapt the mesh based on the minimization of an energy functional for mesh equidistribution and alignment. This leads to a system of ordinary differential equations (ODEs) to be solved which determine where to move the interior mesh nodes. An efficient solution is obtained by solving the ODEs on subregions of the mesh with overlapped communication and computation. Strong and weak scaling experiments on up to 128 cores for meshes with up to 160M elements demonstrate excellent results.

**Keywords: parallel mesh quality improvement, variational method, tetrahedral mesh, distributed computing**

## 1. INTRODUCTION

There are numerous large-scale scientific applications requiring adaptive meshes with millions to billions of elements, e.g., [1, 2, 3, 4]. Such large computational simulations are possible due to the availability of massively parallel supercomputers which integrate central processing units (CPUs) and accelerators, such as graphics processing units (GPUs), Phi coprocessors, and field programmable gate arrays (FPGAs). New parallel mesh generation, parallel mesh adaptation, and parallel mesh quality improvement algorithms have been developed to take advantage of these novel architectures.

Although there are numerous parallel mesh generation algorithms [5], only a few parallel mesh quality improvement algorithms have been developed [6, 7, 8, 9, 10]. The methods presented in [6, 7, 8, 11] are solely devoted to improving the mesh quality, whereas the ones in [9, 10] combine mesh untangling and mesh quality improvement procedures. However, to the best of our knowledge, no parallel variational mesh adaptation methods have been developed.

Since very few parallel mesh quality improvement algorithms and no parallel mesh adaptation algorithms have been proposed, we also review the serial methods developed for such purposes. The vast majority of sequential mesh quality improvement and mesh adaptation methods employ optimization techniques to improve the mesh quality or to adapt the mesh to changes in the geometry or the physics of the application. Optimization-based mesh quality improvement and mesh adaptation algorithms adjust the po-

sitions of the node while preserving the mesh topology [12, 13, 14, 15, 16, 17, 18, 19, 20].

Variational methods for mesh adaptation and mesh quality improvement have recently received considerable attention from the meshing community (e.g., [19, 20, 21, 22, 23, 24]). Whereas most optimization-based mesh quality improvement algorithms use gradient-based techniques to minimize an objective function, Huang and Kamenski [19, 20] instead use the Moving Mesh PDE (MMPDE) method to discretize and find the minimum of an appropriately constructed meshing functional [25, 26, 27]. The minimizer of the meshing functional is a bijective mapping which generates an improved quality mesh as an image of the initial mesh.

In this paper, we present a novel, efficient parallel variational mesh quality improvement algorithm and the corresponding implementation for distributed memory machines. Our parallel method is based on the sequential method by Huang, Ren, and Russell [21] and the recent implementation by Huang and Kamenski [20]. The method finds the minimizer of a meshing functional by solving a system of ordinary differential equations (ODEs) for the nodal velocities. We first review the key concepts of variational mesh methods and the implementation of the sequential MMPDE method in Section 2. In Section 3, we describe our parallel variational mesh quality improvement method for distributed memory systems, along with the implementation. Our method employs a domain decomposition approach in order to divide the workload among the cores. We reorganize the computation within each subregion in order to facilitate the overlap of communication with computation. We analyze the computational complexity of the method in Section 4. We carry out numerical experiments on tetrahedral meshes and determine the strong and weak scaling properties of the proposed method. The numerical experiments and the associated results are discussed in Section 5. We present our conclusions on the work and several directions for future work in Section 6.

## 2. VARIATIONAL MESH ADAPTATION METHODS

In this section, we present an overview of variational mesh adaptation and the corresponding methods. In the variational approach, an adaptive mesh is generated as the image of a reference mesh under a coordinate transformation. The coordinate transformation is determined as the minimizer of a meshing functional. The mesh concentration is typically controlled through a scalar or a matrix-valued function. This is referred to as the metric tensor or monitor function. Monitor functions are defined based on error estimates and/or physical considerations.

Several authors have reported on variational mesh adaptation methods with various types of meshing functionals. For example, Winslow [28] developed an equipotential method based on variable diffusion. Brackbill and Saltzman developed a method combining mesh concentration, smoothness, and orthogonality [29]. Dvinsky developed another approach based on the energy of harmonic mappings [30]. Variational methods based on the conditioning of the Jacobian matrix of the coordinate transformation were developed by Knupp [15] and Knupp and Robidoux [26]. More recently, equidistribution and alignment conditions were used by Huang [31] and Huang and Russell [32] to develop mesh adaptation methods.

The Moving Mesh PDE (i.e., MMPDE) method, which was proposed by Huang, Ren, and Russell in 1994 [21] is the basis upon which many other variational mesh adaptation methods have been developed. In 2015, Huang and Kamenski developed a more efficient implementation of the serial MMPDE method [20].

### 2.1 New implementation of the variational mesh adaptation method

In this subsection, we focus on Huang and Kamenski's new implementation of the MMPDE method [20]. Consider a domain $\Omega \subset \mathbb{R}^d (d \geq 1)$ and $\mathcal{T}_h = \{K\}$ be a simplicial mesh containing $N$ elements and $N_v$ vertices on $\Omega$. Denote the affine mapping $F_K : \hat{K} \to K$ and its Jacobian matrix by $F'_K$, where $\hat{K}$ is the master element. Let the edge matrices for $K$ and $\hat{K}$ be $E_K$ and $\hat{E}$. Assume that a metric tensor (or a monitor function) $\mathbb{M} = \mathbb{M}(\mathbf{x})$ is given on $\Omega$ which provides directional and magnitude information for the elements.

A key idea of the MMPDE method is to view an adaptive mesh as a uniform one in the metric $\mathbb{M}$ such that the following two properties hold. First, the size of all elements $K$ in the metric $\mathbb{M}_K$ is the same. Second, all elements $K$ in the metric $\mathbb{M}_K$ are similar to $\hat{K}$.

These two properties give rise to the equidistribution and alignment conditions:

$$|K|\sqrt{\det(\mathbb{M}_K)} = \frac{\sigma_h}{N}, \quad \forall K \in \mathcal{T}_h$$
$$\frac{1}{d}\mathrm{tr}\left((F'_K)^T \mathbb{M}_K F'_K\right) = \det\left((F'_K)^T \mathbb{M}_K F'_K\right)^{\frac{1}{d}}, \quad \forall K \in \mathcal{T}_h,$$

where $|K|$ is the volume of $K$ and $\sigma_h = \sum_K |K|\sqrt{\det(\mathbb{M}_K)}$.

Then an energy function for the equidistribution and

alignment conditions is given by

$$I[\mathcal{T}_h] = \sum_K |K|\, G_K, \text{ where}$$
$$G_K = \frac{1}{3}\sqrt{\det(\mathbb{M}_K)}\left(\mathrm{tr}(\mathbb{J}\mathbb{M}_K^{-1}\mathbb{J}^T)\right)^d + \frac{1}{3}d^d\sqrt{\det(\mathbb{M}_K)}\left(\frac{\det(\mathbb{J})}{\sqrt{\det(\mathbb{M}_K)}}\right)^2 \text{ and}$$
$$\mathbb{J} = (F_K')^{-1} = \hat{E}E_K^{-1}.$$

Minimization of the energy function will result in a mesh that closely satisfies the equidistribution and alignment conditions.

The MMPDE moving mesh equation is then defined as the (modified) gradient system of $I[\mathcal{T}_h]$, i.e.,

$$\frac{d\mathbf{x}_i}{dt} = -\frac{\det(\mathbb{M}_i)^{\frac{1}{d}}}{\tau}\frac{\partial I[\mathcal{T}_h]}{\partial \mathbf{x}_i}, \quad i = 1, ..., N_v,$$

where $\tau > 0$ is a parameter for adjusting the response time scale of mesh movement to the change in $\mathbb{M}$.

For mesh quality improvement, we choose $\mathbb{M} = \mathbb{I}$, which means we want the mesh to be as uniform as possible in the Euclidean space. In this case, the moving mesh equation reads as

$$\frac{d\mathbf{x}_i}{dt} = \sum_{K\in\omega_i} |K|\mathbf{v}_{i_K}^K, \quad i = 1, ..., N_v, \tag{1}$$

where $\omega_i$ is the element patch associated with $\mathbf{x}_i$, $i_K$ is the local index for $\mathbf{x}_i$ on $K$, and $\mathbf{v}_{i_K}$ is the local nodal velocity contributed by $K$ to the node $x_i$. The analytical formula of the local nodal velocities is given in [20]. In the case when $\mathbb{M} = \mathbb{I}$,

$$\begin{bmatrix} (\mathbf{v}_1^K)^T \\ \vdots \\ (\mathbf{v}_d^K)^T \end{bmatrix} = -G_K E_K^{-1} + E_K^{-1}\frac{\partial G_K}{\partial \mathbb{J}}\hat{E}E_K^{-1} + \frac{\partial G_K}{\partial \det(\mathbb{J})}\frac{\det(\hat{E})}{\det(E_K)}E_K^{-1}, \tag{2}$$
$$(\mathbf{v}_0^K)^T = -\sum_{j=1}^d (\mathbf{v}_j^K)^T.$$

The partial derivatives in equation (2) are obtained by substituting $\mathbb{M} = \mathbb{I}$ into the formula for $G_K$ shown above and then differentiating with respect to $\mathbb{J}$ and $\det(\mathbb{J})$.

The nodal velocities of the boundary nodes are set to 0. They can also be modified to let the boundary nodes slide along the boundary.

To determine the locations of the interior nodes, Equation (1) is then solved using the adaptive fourth-order Runge-Kutta-Fehlberg ODE solver (RKF45). It has been shown analytically and numerically in [33] that the mesh governed by the MMPDE moving mesh equation will stay nonsingular (i.e., no crossing nor tangling will occur) if it is nonsingular initially.

# 3. PARALLEL VARIATIONAL MESH QUALITY IMPROVEMENT ALGORITHM

In this section, we present our novel parallel algorithm and implementation for distributed memory systems based on the moving mesh method described in the previous section.

## 3.1 Sequential algorithm

For the sequential algorithm, the adaptive fourth-order Runge-Kutta Fehlberg ODE solver (RKF45) with fifth-order error estimator is employed to solve (1) (see [34] for details). The RKF45 method approximates the solution of an ODE system in the form

$$\frac{dy}{dt} = f(t, y) \tag{3}$$

using a non-constant, optimal step size $dt$ in each iteration. The method determines the step size $dt$ in each iteration by comparing a fourth-order approximation, $y_{i+1}$, and a fifth-order approximation, $z_{i+1}$, to the solution. These approximations are given by

$$y_{i+1} = y_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5, \tag{4}$$

and

$$z_{i+1} = y_i + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6, \tag{5}$$

respectively. Here

$$\begin{aligned}
k_1 &= dtf(t_i, y_i),\\
k_2 &= dtf\left(t_i + \frac{1}{4}dt, y_i + \frac{1}{4}k_1\right),\\
k_3 &= dtf\left(t_i + \frac{3}{8}dt, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),\\
k_4 &= dtf\left(t_i + \frac{12}{13}dt, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),\\
k_5 &= dtf\left(t_i + dt, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),\\
k_6 &= dtf\left(t_i + \frac{1}{2}dt, y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).
\end{aligned} \tag{6}$$

The error is given by the $\infty$-norm of the difference between the two solutions, i.e., $err = \|z_{i+1} - y_{i+1}\|_\infty$. If $err$ is smaller than a given tolerance, $tol$, then the solution $y_{i+1}$ is accepted. One can show that the optimal step size is given by $q * dt$, where

$$q = 0.84 \left( \frac{tol * dt}{err} \right)^{\frac{1}{4}}. \quad (7)$$

**Algorithm 1** Sequential variational mesh quality improvement algorithm

1: Input: nodal coordinates, topology, boundary nodes
2: Define: Initial $dt$, $t_{final}$, $tol$, $t = 0$, $i = 0$
3: **while** $(t < t_{final})$ **do**
4: **for** each node in the mesh **do**
5: Compute $k_1 - k_6$ from equations (6) and the ODE (1)
6: **end for**
7: Compute $y_{i+1}$, $z_{i+1}$ (equations (4) and (5)) and error $(err)$
8: $dt = q * dt$ where $q$ is given by equation (7)
9: **if** $(err < tol)$ **then**
10: Accept $y_{i+1}$ as a solution
11: **else**
12: $dt = \max(q * dt, 0.1 * dt)$;
13: **end if**
14: $t = t + dt$
15: **end while**
16: Output: new nodal coordinates

In Algorithm 1, (i.e., the algorithm for the method proposed in [20]), the calculation of the nodal velocities is directly related to the calculation of the $k_i$, which is the most computationally-intensive step (i.e., step 5). To calculate the values, $k_i$, in the RKF45 method, the algorithm loops over all elements calculating partial nodal velocities for each node. This requirement is the basis for our distributed data approach in the parallel algorithm.

## 3.2 Overview of the parallel algorithm

In this subsection, we highlight three important aspects of our parallel algorithm (Algorithm 2): the distribution of the workload, the communication strategy, and the termination criteria. Although there exist multiple strategies to distribute the work among cores, we employ a domain decomposition approach in which we divide the domain into $p$ regions, where $p$ is the number of cores (i.e., steps 4 and 5 in Algorithm 2). Each region is (ideally) composed of one connected component. We use this approach because according to Equation (2) the nodal velocity of a particular node $\mathbf{x}_m$, such as the one in Fig. 1, is calculated based on the edge matrices of elements $E_1$, $E_2$, $E_3$, $E_4$, and $E_5$. Therefore, a decomposition of the elements of the domain into regions is the strategy that yields the best performance. To accomplish this, we use METIS [35], which is a library for partitioning meshes and graphs. We employed the mpmetis scheme to partition the mesh into regions so that each region has roughly the same number of elements and the number of interfaces between regions is minimized.

Once we have the mesh partition, core $P_0$ reads and distributes the information concerning the topology and nodal coordinates to the rest of the cores. In this step, each core creates a list (`SharedNodes_p[]`) whose size is equal to the number of nodes along partition boundaries (i.e., the number of shared nodes). Each core stores partial nodal velocities to specific locations in the list and fills-in the rest with zeros.

Whereas each core computes the new nodal positions for the interior nodes of its corresponding region, the new nodal positions for nodes along partition boundaries (corresponding to the shared nodes) requires communication and verification steps (i.e., steps 17 and 19 in Algorithm 2). In our parallel algorithm, all communication steps are reduction operations. To compute the new nodal positions for shared nodes, we perform a reduction operation (summation) over the list `SharedNodes_p[]` in which we store the partial nodal velocities of the shared nodes. We also need communication steps to calculate the global error (i.e., step 23 in Algorithm 2) and the average mesh quality (i.e., step 31 in Algorithm 2). To calculate the global error, we require a reduction operation to calculate the maximum. We also require a summation reduction for the average mesh quality.

Finally, we employ a tetrahedral mesh quality metric in order to evaluate the quality of the mesh on each iteration. We utilize the mesh quality information in the termination criteria. The mesh quality metric implemented in our algorithm is given by

$$q = \frac{CR}{3 * IR}, \quad (8)$$

where $CR$ is the circumsphere radius and $IR$ is the inscribed sphere radius. For this metric, $q \in [1, \infty)$ where $q = 1.0$ is the optimal mesh quality. We terminate the parallel variational mesh quality improvement algorithm when the difference in the average mesh quality on two consecutive iterations is small (i.e., less than a specified tolerance). Also, note that we do not use unnecessary synchronization calls (**MPI_Barrier**) in the algorithm. However an implicit synchronization might be performed (if necessary) with **MPI_Wait**. **MPI_Barrier** is only used for timing purposes.

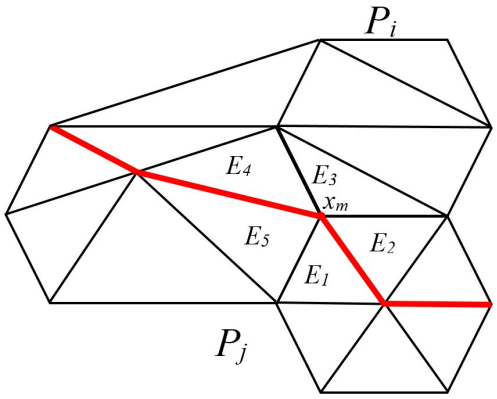



**Figure 1**: Patch of elements with $\mathbf{x}_m$ as one of its vertices.

### 3.3 Overlapping communication with computation

As we mentioned before, the most computationally-intensive step in the parallel variational mesh quality improvement algorithm is the computation of the nodal velocities. For the case in Fig. 1, core $P_i$ is unable to compute the nodal velocity for node $\mathbf{x}_m$ because the core does not have access to elements $E_1$ and $E_5$. Therefore, $P_i$ calculates only a portion of the nodal velocity at this node. The same is true for core $P_j$.

According to the previous description, we design the parallel algorithm such that every core $P_i$ loops once over its own elements to calculate the nodal velocities for the interior nodes within a region. However, for shared nodes, the nodal velocities are incomplete. Therefore, in this case, e.g., for $\mathbf{x}_m$ in Fig. 1, cores $P_i$ and $P_j$ store the partial nodal velocities in the `SharedNodes_p[]` list. Finally, the nodal velocities for the shared nodes require a reduction operation (summation) over `SharedNodes_p[]` and a verification step (i.e., steps 17 and 19 in Algorithm 2).

It is possible to overlap the communication and computation and reduce the overall run time by reorganizing the data in the data structures. To this end, each core splits the list of local elements `Elements_proc[]` into two new lists, i.e., `Elements_proc1[]` and `Elements_proc2[]` (i.e., step 7 in Algorithm 2). The algorithm stores the elements that contain at least one shared node in the data structure `Elements_proc1[]`, whereas the elements whose nodes are interior nodes are stored in `Elements_proc2[]`. Thus, we calculate the nodal velocities in two steps (i.e., steps 15 and 18 in Algorithm 2). After the first step, the algorithm will have partial nodal velocities for the shared nodes. Therefore, we can initiate the communication using the non-blocking collective command

**Algorithm 2** Parallel algorithm for variational mesh quality improvement

1: Input: nodal coordinates, topology, boundary nodes, domain decomposition information
2: Define: Initial $dt$, $errtol$, $tol$, and $t = 0$
3: // Mesh partition, and data structure creation
4: Partition the mesh using METIS
5: Create and distribute data structures among cores
6: Compute: local and global mesh quality $Q_{new}$ using **MPI_Iallreduce**
7: Split elements into two sets, `Elements_proc1[]` and `Elements_proc2[]`
8: Check that the communication is completed (**MPI_Wait**)
9: Set $Q_{old} = 1.0$
10: // Solve differential equation (1)
11: **for** all $p$ cores in parallel **do**
12: **while** $(|Q_{old} - Q_{new}| > errtol)$ **do**
13: $Q_{old} = Q_{new}$;
14: **for** i=1 to 6 **do**
15: Compute $k_i$ from equations (6) using only nodes from `Elements_proc1[]`
16: Copy shared nodes (from $k_i$) to a global shared node array in $p$
17: Communicate and sum all global shared node arrays (**MPI_Iallreduce**)
18: Compute $k_i$ from equations (6) using only nodes from `Elements_proc2[]`
19: Check that the communication has been completed (**MPI_Wait**)
20: Update $k_i$ with new shared node information
21: **end for**
22: Compute $y_{i+1}$, $z_{i+1}$ (equations (4))
23: Compute local error ($err$) and apply **MPI_Allreduce** to obtain global error
24: $dt = q * dt$ where $q$ is given by equation (7)
25: **if** $(err < tol)$ **then**
26: Accept $y_{i+1}$ as a solution
27: **else**
28: $dt = \max(q * dt, 0.1 * dt)$
29: **end if**
30: $t = t + dt$
31: Compute: local and global mesh quality $Q_{new}$ using **MPI_Iallreduce**
32: **end while**
33: **end for**
34: Output: New nodal coordinates

**MPI_Iallreduce** which immediately calculates the nodal velocities for the interior nodes (i.e., step 18 in Algorithm 2). Once the algorithm finishes the calculation of nodal velocities for the interior nodes, the algorithm checks to see if the communication has completed using **MPI_Wait** (i.e., step 19 in Algorithm 2). Finally, the algorithm updates the nodal velocities for

the shared nodes (i.e., step 20 in Algorithm 2).

## 4. PARALLEL RUNTIME ANALYSIS

In this section, we discuss the runtime performance of the parallel algorithm described in the previous section. In particular, we analyze the average parallel runtime.

First, we assume the partition of the initial mesh is given to the algorithm as input data. Recall that we use METIS to accomplish this step. Once the algorithm reads the input data, core $P_0$ distributes the information among cores according to the partition file. This overhead computation is performed sequentially and occurs just once throughout the execution of the algorithm. We assume this step takes $t_N$ time.

Since we performed the mesh partitioning step over the elements of the mesh, assuming that $N$ is the total number of mesh elements, each core contains (ideally) $\lceil N/p \rceil$ elements. With this information, the splitting operation performed within each region to overlap communication with computation takes $\lceil N/p \rceil (d+1)$ operations, where $d$ is the dimension. This step is also performed once in the algorithm.

For the next step, we solve the differential equation (1) by calculating the values $k_i$. To calculate $k_i$, the algorithm loops over the mesh elements. If the maximum time to calculate the nodal velocity for each node is $t_{vn}$, then the total serial time to calculate the nodal velocities is $N(d+1)\,t_{vn}$. Therefore, the parallel time is $\lceil N/p \rceil (d+1)\,t_{vn}$. Moreover, we define the number of elements containing at least one shared node in the region corresponding to core $P_i$ as $N_{sh}^{(P_i)}$, and the number of elements containing only interior nodes as $N_{int}^{(P_i)}$. Note that $\lceil N/p \rceil = N_{sh}^{(P_i)} + N_{int}^{(P_i)}$. For the communication process, first, we extract the local shared nodes. Assuming the time to copy one node from the local to the global list is $t_c$ and the number of shared nodes in the mesh is $V_{sh}$, then this step takes $V_{sh}\,t_c$ time. Similarly, assuming that the time to send a vector with $V_{sh}$ nodes is $t_s$, the communication process takes $\log_2(p)\,t_s + p\,V_{sh}\,t_c$. Note that this was implemented as a non-blocking communication process using the computation time $N_{int}^{(p)}(d+1)\,t_{vn}$ to overlap communication and computation. Thus, the time to compute these two processes is $Tc_{total}$, where

$$Tc_{total} = \begin{cases} T_{int}, & \text{if } T_{int} > T_{comm} \\ T_{int} + |T_{int} - T_{comm}|, & \text{otherwise.} \end{cases} \tag{9}$$

Here $T_{int} = N_{int}^{(p)}(d+1)\,t_{vn}$ and $T_{comm} = \log_2(p)\,t_s + p\,V_{sh}\,t_c$.

To copy the information from the global to the local list in each core costs $V_{sh}\,t_c$. Assuming that the time to compute the error in each coordinate of each node is $t_e$, the total serial time is $N(d+1)d\,t_e$. In parallel, it is $\lceil N/p \rceil (d+1)d\,t_e$ plus the time to calculate the maximum error among cores which is $\log_2(p)$. Finally, if $t_q$ is the time to calculate the quality of one element, then $Nt_q$ is the time to calculate the quality for the serial algorithm, and $\lceil N/p \rceil t_q$ is the time for the parallel one. The time to calculate the average quality among cores is $\log_2(p)$. With this information, the total parallel time per iteration is

$$\begin{aligned} T_P =& p\,V_{sh}\,t_c + 2log_2(p) + (d+1)(\lceil N/p \rceil_{sh}\,t_{vn} \\ &+ \lceil N/p \rceil d\,t_e) + \lceil N/p \rceil t_q + Tc_{total}. \end{aligned} \tag{10}$$

The major source of overhead due to communication in (10) is $Tc_{total}$ and $p\,V_{sh}\,t_c$ due to the fact that the number of shared nodes always increases with the number of mesh elements and the number of cores, therefore $p\,V_{sh}\,t_c$ increases. Hence, excellent timing results are expected in the cases for which the number of interior nodes in each partition is large compared with the number of shared nodes in the mesh.

## 5. NUMERICAL EXPERIMENTS

Our algorithm was implemented in C/C++ using the message-passing interface (OpenMPI version 1.8.7). We ran our experiments on the high performance computing cluster available to us through the Advanced Computing Facility (ACF) at the University of Kansas. More specifically, we ran the experiments on twenty-one Dell R730 servers, each of them equipped with 2x dodeca-core Intel Haswell processors running at 2.5 GHz with 128GB of RAM, 1TB HDD, and FDR Infiniband.

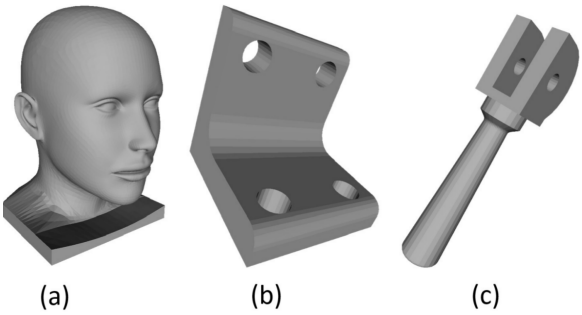



**Figure 2**: Domains used to test the parallel algorithm: (a) bust, (b) bracket and (c) double cam tool

To test the performance of our parallel algorithm, we constructed several tetrahedral meshes based on three geometries from various applications and with different characteristics. Figure 2 illustrates the three-dimensional domains used in our experiments. We chose the geometries from different online databases, Fig. 2(a) is part of the 3dcadbrowser project [36],

while Fig. 2(b) and (c) are part of the French Institute for Research in Computer Science and Automation (INRIA) databases [37]). We used GHS3D [37] and MeshLab [38] to generate a new surface mesh and to scale the domain to meet our needs. Based on these surface meshes, we generated tetrahedral volume meshes using Tetgen [39] with the numbers of elements specified in Tables 1 and 2. Finally, we randomly perturbed the nodes of each mesh to reduce their quality. The resulting tetrahedral meshes are used to test the performance of our parallel variational mesh quality improvement (Parallel VMQI) algorithm.

We used the meshes for the bust and the double cam tool domains to test the algorithm for strong scaling and the meshes for the bracket domain to test weak scaling.

**Table 1**: Size of tetrahedral meshes for the bust and the double cam tool domains

| Mesh | # Nodes | # Elements |
|---|---|---|
| bust | 12,895,493 | 80,000,012 |
| double cam tool | 7,089,753 | 41,405,684 |

**Table 2**: Various mesh sizes for the bracket domain

| Mesh | # Nodes | # Elements |
|---|---|---|
| bracket | 450,960 | 2,500,032 |
| | 864,028 | 5,000,025 |
| | 1,716,222 | 9,999,990 |
| | 3,269,784 | 19,999,978 |
| | 6,497,224 | 40,000,000 |
| | 12,957,609 | 80,000,037 |
| | 24,177,335 | 159,745,245 |

For our first experiment, we employed a tetrahedral mesh with approximately 80M elements for the bust domain (see Fig. 3). We ran the algorithm with different numbers of cores using $dt = 10^{-14}$, $errtol = 10^{-5}$, and $tol = 0.001$ as input parameters (see Algorithm 2). These values guarantee that the algorithm will run until convergence with an error of $errtol = 10^{-5}$. Figure 4 shows the average mesh quality versus the number of iterations. This demonstrates the ability of the algorithm to improve the average mesh quality.

Figures 5(a) and 5(b) show that for a small number of cores, the runtime, and speedup achieved by the parallel algorithm are very close to the ideal ones. A small deviation in the speedup for a larger number of cores is also observed. The deviation is clearer at sixteen cores and it does not grow much for a higher number of cores. It is clear that the pre-processing step (distribution of nodes, elements and boundary nodes

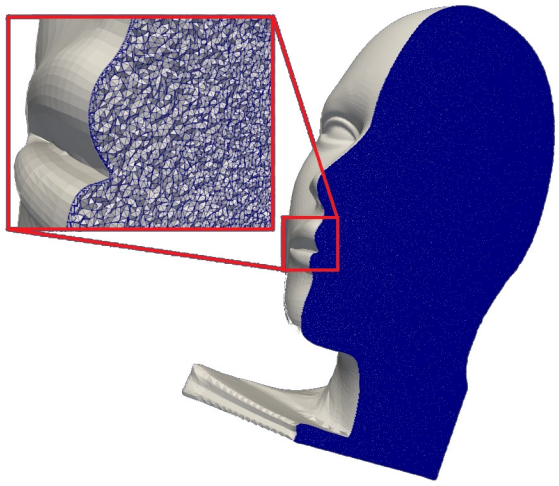


**Figure 3**: 80M element tetrahedral mesh of the bust domain

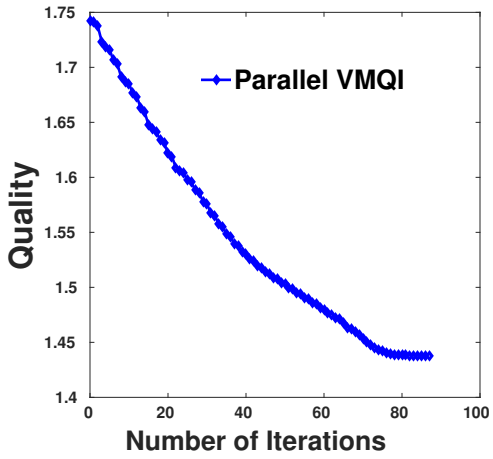
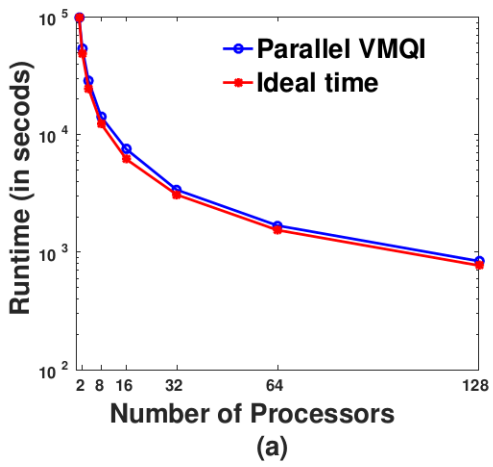



**Figure 4**: Average quality versus number of iterations for the 80M element tetrahedral mesh of the bust domain

and identification of shared nodes) is a major source of overhead that significantly contributes to the discrepancy between the calculated and ideal speedup. On the other hand, when the number of interior nodes on each core is high compared with the number of shared nodes, it is more likely that the communication steps (when solving the differential equation) overlaps with the calculations of the nodal velocities for the interior nodes; therefore, the communication steps contribute less to the performance degradation in such a case. The runtimes reported in Fig. 5 are the average of five experiments.

Our second experiment is a strong scaling experiment using the double cam tool domain and a tetrahedral mesh with approximately 40M elements (see Fig. 6),



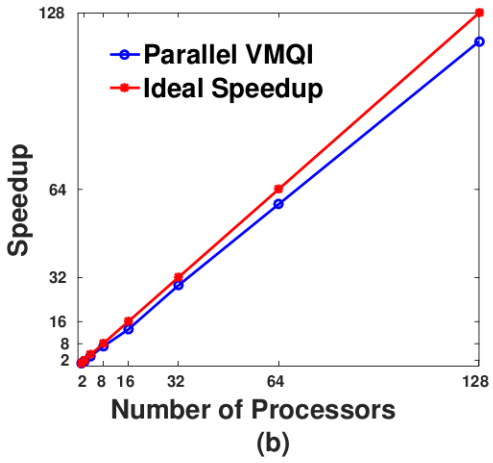




**Figure 5**: (a) Total runtime and (b) speedup for the Parallel VMQI algorithm for the 80M element tetrahedral mesh of the bust domain

which is approximately half the number of elements used for the first experiment. We decided to include this test case so as to measure the performance of the algorithm when the number of interior nodes per core is reduced. In this case, it may be more challenging to overlap communication with computation in an effective manner. The initial parameters ($dt$, $errtol$, $tol$) are the same as in the first strong scaling test. Figure 7 shows the average mesh quality versus the number of iterations for this tetrahedral mesh.

Figures 8(a) and 8(b) show the total runtime and speedup for the 40M element mesh of the double cam tool. The results are in general similar to the ones for the first test case. This demonstrates that our parallel algorithm scales very well with the resources used at the University of Kansas. It is worth mentioning that the maximum number of cores report in our experiments is limited by our accessibility to the cluster.

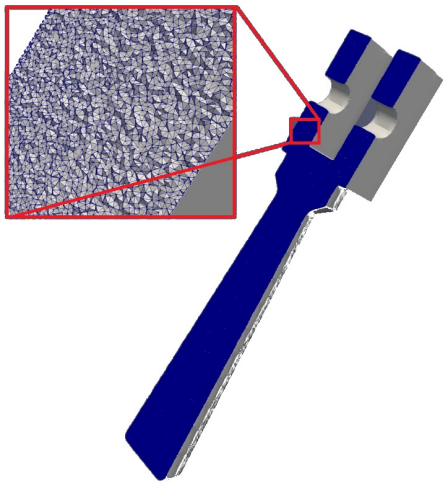


**Figure 6**: 40M element tetrahedral mesh of the double cam tool domain

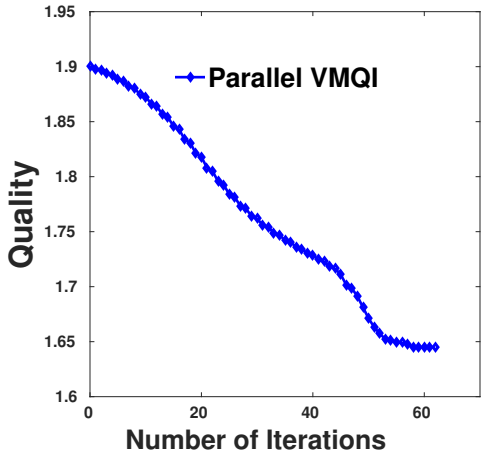



**Figure 7**: Average quality versus the number of iterations for the 40M element tetrahedral mesh of the double cam tool domain

We attribute the good results from the previous two examples to the the ability of our parallel algorithm to overlap communication with computation thus reducing the runtime. When this is possible, the major source of performance degradation, i.e, $Tc_{total}$ from equation (9), is reduced. Figures 9(a) and 9(b) show the computation and communication time for the bust and double cam tool test cases. The figure shows the time employed by one core to calculate the nodal velocities for the interior nodes in its own region (computation). The communication time is the time employed to communicate the shared nodes. Note that, for these two cases, the computation time is always significantly higher than the communication time, which guarantees a good performance of the algorithm. Also, it is

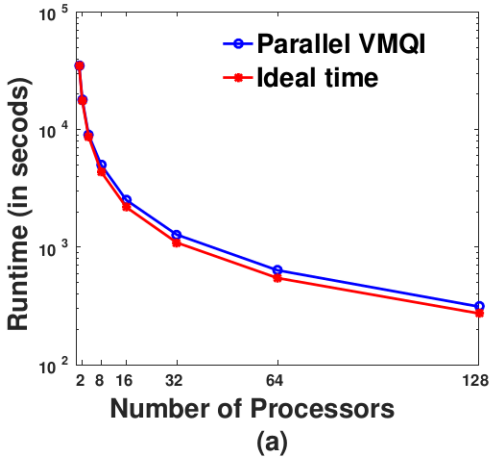



**Figure 8**: (a) Total runtime and (b) speedup for the Parallel VMQI algorithm for the 40M element tetrahedral mesh of the double cam tool domain

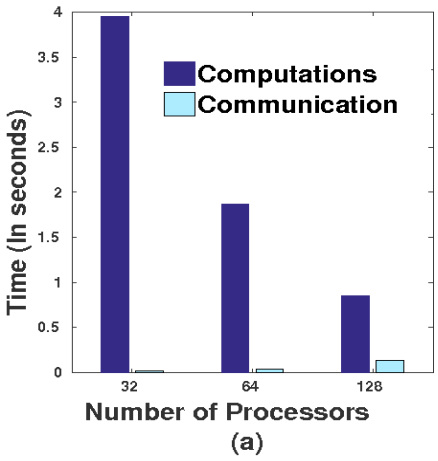



**Figure 9**: Communication and computation times employed for one iteration to calculate the nodal velocities in one region of (a) 80M element mesh of the bust domain and a (b) 40M element mesh of the double cam tool domain

expected that the computation time is reduced by half each time we double the number of cores. However, the communication time does not show a clear growth pattern. Theoretically, for the ideal case, the communication time should exhibit logarithmic growth, but in practice this is not the case. For our case, the communication time is related to the architecture of the cluster and with the distribution and availability of nodes and cores at runtime.

We also performed a weak scaling test, which investigates how the solution time changes with respect to the number of cores (and assuming a constant workload per core), using various tetrahedral meshes for the bracket domain (see Tab. 2 and Fig. 10). For this experiment, we used the same parameters as in the previous test case, except for the initial $dt$ value, which was $dt = 10^{-6}$ for this case. We made this change to better control the number of iterations in each computational simulation. Figure 11 shows the weak scaling result for the algorithm. We observe a small deviation in the runtime for various numbers of cores. This deviation is at most six seconds, which is a deviation of less than 5% from the mean value. This behavior is a typical weak scaling result on unstructured mesh computations, as it is very difficult to double exactly the problem size as the number of cores is doubled. Also, since the number of iterations for each simulation might be different (see Fig. 12), the results from Fig. 11 correspond to the time the algorithm takes to run only ten iterations.

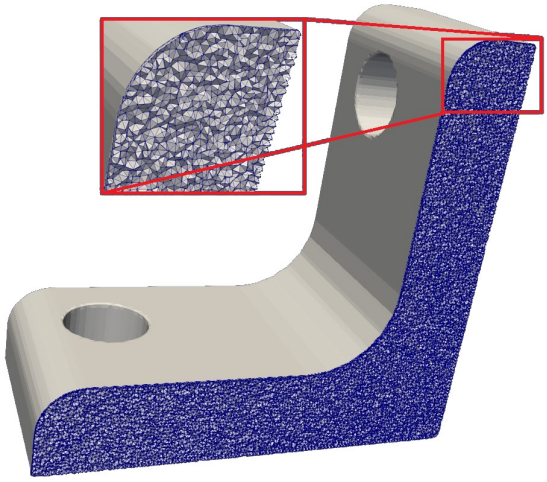


**Figure 10**: 20M element tetrahedral mesh of the bracket domain

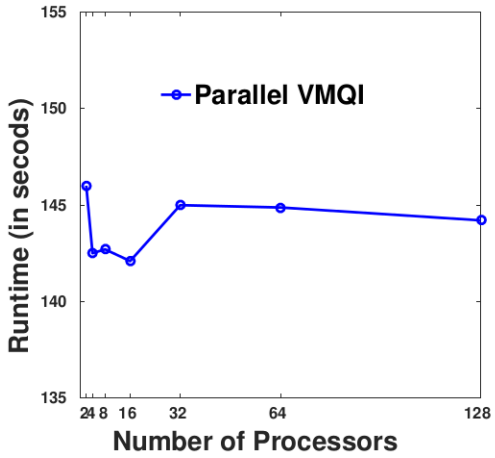



**Figure 11**: Runtime versus number of cores to test the weak scaling efficiency

## 6. CONCLUSIONS AND FUTURE RESEARCH

We proposed a parallel variational mesh quality improvement algorithm and an associated implementation for the method in [20, 21] for distributed memory machines. To distribute the workload among cores, we use METIS to generate a mesh partition composed by regions of connected elements. The algorithm identifies the elements in each region that contain at least one node that is shared by multiple regions (shared nodes). After distribution of the data (nodal coordinates, topology, boundary nodes), each core organizes its corresponding elements into two sets, i.e., the elements composed of only interior nodes and the elements which have at least one shared node.

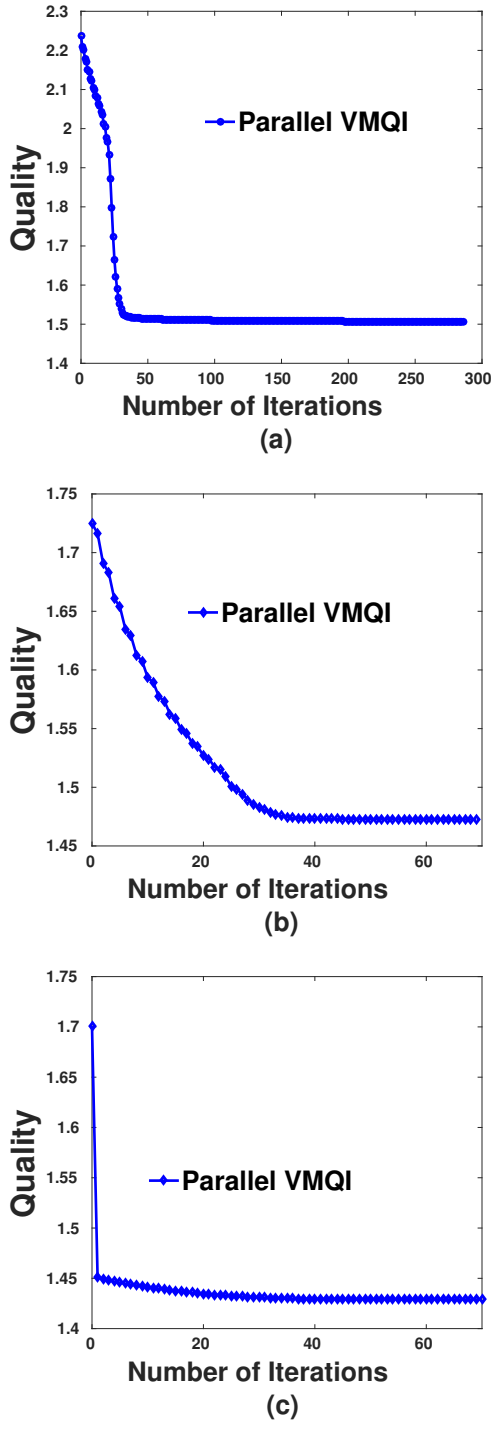



**Figure 12**: Quality versus number of iterations for the bracket domain with (a) 2.5M, (b) 10M, and (c) 40M elements

We employed the RKF45 method to solve the system of ODEs associated with the interior nodes. For this

process, the parallel algorithm loops over all elements on each core to calculate the nodal velocities for each interior node. Whereas each core is able to calculate the nodal velocities for the interior nodes within its region, computing the nodal velocities of the shared nodes requires communication among cores. To do this efficiently, the algorithm first calculates the nodal velocities for elements containing at least one shared node. Then we communicate the partial nodal velocities of the shared nodes using a non-blocking collective instruction to overlap communication with computation of the nodal velocities for the interior nodes. When the number of interior nodes per core is high, a total overlap of communication and computation is achieved. Finally, the algorithm calculates the average quality of the mesh in each iteration and uses this information to terminate the computations when no significant improvement of the average mesh quality is observed.

We tested our parallel variational mesh quality improvement algorithm on three different 3D domains which were discretized using tetrahedral meshes. The results of our numerical experiments show good strong scalability and speedup for the meshes with 80M and 40M elements on up to 128 cores. The efficiency observed in the experimental results is the consequence of the complete overlap of communication and computation when calculating the nodal velocities (see Fig. 9). For the test cases presented in this paper, the major source of overhead occurs in the pre-processing step, i.e., where $P_0$ distributes the data and identifies the shared nodes. In addition to this, if the number of interior nodes on each core is relatively small compared with the total number of shared nodes, then the communication time among cores increase relative to the runtime. Hence we obtain a performance degradation, as a complete overlap of communication and computation is not possible. The weak scaling results we obtained are typical for unstructured meshes.

For future research, we plan to explore different communication strategies to minimize the memory consumption and communication time. A local-blocking communication strategy might decrease the performance for small number of cores, but it will perform better for a larger number of cores. In addition, a parallel pre-processing step will reduce the runtime and memory consumption for $P_0$. Another possible avenue for research is the adoption of a different domain decomposition strategy such as node coloring. In regards to applications, one can extent the same ideas presented in this paper to the variational mesh adaptation algorithms such as the one in [20].

## 7. ACKNOWLEDGMENTS

The work of the first author was supported in part by NSF grants OAC-1500487 (formerly OAC-1330056 and OAC-1054459), CCF-1717894, and OAC-1808553. The work of the second author was supported by NSF grant OAC-1500487. The work of all three authors was supported through instrumentation funded by the Army Research Office under contract W911NF-15-1-0377.

## References

[1] Lei J., Wang X., Xie G., Lorenzini G. "Turbulent flow field analysis of a jet in cross flow by DNS." *J. Eng. Thermophys.*, vol. 24, 259–269, 2015

[2] Aliabadi S., Johnson A., Abedi J., Zellars B. "High Performance Computing of Fluid-Structure Interactions in Hydrodynamics Applications Using Unstructured Meshes with More than One Billion Elements." *Proc. of the 2002 International Conference on High Performance Computing (HiPC 2002)*, vol. 2552, pp. 519–533. Springer Berlin Heidelberg, 2002

[3] Chan H., Lu Z., Chi X. "Large-scale parallel simulation of high-dimensional American option pricing." *J. Algorithm Compt. Technol.*, vol. 6, 1–16, 2012

[4] Tian F., Dai H., Luo H., Doyle J., Rousseau B. "Fluid-structure interaction involving large deformations: 3D simlations and applications to biological systems." *J. Comput. Phys.*, vol. 258, 451–469, 2014

[5] Chrisochoides N. "A survey of parallel mesh generation methods." A. Bruaset, A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, pp. 237–264. Springer Berlin Heidelberg, 2006

[6] Freitag L., Jones M., Plassmann P. "A parallel algorithm for mesh smoothing." *SIAM J. Sci. Comput.*, vol. 20, 2023–2040, 1999

[7] Jiao X., Alexander P. "Parallel feature-preserving mesh smoothing." *Computational Science and its Applications ICCSA*, vol. 3483, pp. 1180–1189. 2005

[8] Gorman G., Southern J., Farrell P., Piggott M., Rokos G., Kelly P. "Hybrid OpenMP/MPI anisotropic mesh smoothing." *Procedia Computer Science*, vol. 9, pp. 1513–1522. 2012

[9] Bentez D., Rodrguez E., Escobar J., Montenegro R. "Performance evaluation of a parallel algorithm for simultaneous untangling and smoothing

of tetrahedral meshes." *Proc. of the* $23^{rd}$ *International Meshing Roundtable*, pp. 579–598. 2014

[10] Sastry S., Shontz S. "A parallel log-barrier method for mesh quality improvement and untangling." *Eng. Comput.*, vol. 30, 503–515, 2014

[11] Cheng Z., Shaffer E., Yeh R., Zagaris G., Olson L. "Efficient parallel optimization of volume meshes on heterogeneous computing systems." *Eng. Comput.*, vol. 33, 717–726, 2017

[12] Branets L., Carey G. "A local cell quality metric and variational grid smoothing algorithm." *Eng. Comput.*, vol. 21, 19–28, 2005

[13] Zhang Y., Hamza A. "PDE-based smoothing from 3D mesh quality improvement." *Electro/Info Tech IEEE Int. Conf.*, pp. 334–339. 2006

[14] Freitag L., Knupp P. "Tetrahedral mesh improvement via optimization of the element condition number." *Int. J. Numer. Meth. Eng.*, vol. 53, 13771391, 2002

[15] Knupp P. "Jacobian-weighted elliptic grid generation." *SIAM J. Sci. Comput.*, vol. 17, 1475–1490, 1996

[16] Shontz S., Vavasis S. "A robust solution procedure for hyperelastic solids with large boundary deformation." *Eng. Comput.*, vol. 28, 135–147, 2012

[17] Kim J., Panitanarak T., Shontz S. "A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling." *Int. J. Numer. Meth. Eng.*, vol. 94, 2042, 2013

[18] Freitag L., Knupp P., Munson T., Shontz S. "A comparison of two optimization methods for mesh quality improvement." *Eng. Comput.*, vol. 22, 61–74, 2006

[19] Huang W., Kamenski L., Si H. "Mesh smoothing: An MMPDE approach.", 2015. Research note at the $24^{th}$ International Meshing Roundtable

[20] Huang W., Kamenski L. "A geometric discretization and a simple implementation for variational mesh generation and adaptation." *J. Comput. Phys.*, vol. 301, 322–337, 2015

[21] Huang W., Ren Y., Russell R. "Moving mesh partial differential equations (MMPDEs) based upon the equidistribution principle." *SIAM J. Numer. Anal.*, vol. 31, 709–730, 1994

[22] Huang W., Kamenski L., Russell R. "A comparative study of meshing functionals for variational mesh adaptation." *J. Math. Study*, vol. 48, 168–186, 2015

[23] Alliez P., Cohen-Steiner D., Yvinec M., Desbrun M. "Variational tetrahedral meshing." *ACM Trans. Graph*, vol. 24, 617–625, 2005

[24] Hachem E., Feghali S., Codina R., Coupez T. "Anisotropic adaptive meshing and monolithic variational multiscale method for fluid-structure interaction." *Comput. Struct.*, vol. 122, 88–100, 2013

[25] de Almeida V. "Domain deformation mapping: Application to variational mesh generation." *SIAM J. Sci. Comput.*, vol. 4, 12521275, 1999

[26] Knupp P., Robidoux N. "A framework for variational grid generation: Conditioning the Jacobian matrix with matrix norms." *SIAM J. Sci. Comput.*, vol. 21, 2029–2047, 2000

[27] Liao G. "Variational approach to grid generation." *Numer. Meth. PDE*, vol. 8, 143–147, 1992

[28] Winslow A. "Adaptive mesh zoning by the equipotential method." Tech. Rep. UCID-19062, Lawrence Livermore National Laboratory, 1981

[29] Brackbill J., Saltzman J. "Adaptive zoning for singular problems in two dimensions." *J. Comput. Phys.*, vol. 46, 342–368, 1982

[30] Dvinsky A. "Adaptive grid generation from harmonic maps on Riemannian manifolds." *J. Comput. Phys.*, vol. 95, 450–476, 1991

[31] Huang W. "Variational mesh adaptation: Isotropy and equidistribution." *J. Comput. Phys.*, vol. 174, 1643–1666, 2005

[32] Huang W., Russell R. *Adaptive Moving Mesh Methods*. Springer, 2011

[33] Huang W., Kamenski L. "On the mesh nonsingularity of the moving mesh PDE method." *Math. Comp.*, vol. 87, 1887–1911, 2018

[34] Mathews J., Fink K. *Numerical Methods Using MATLAB*. Prentice Hall, third edn., 1999

[35] Karypis G., Kumar V. "A fast and highly quality multilevel scheme for partitioning irregular graphs." *SIAM J. Sci. Comput.*, vol. 20, 359392, 1999

[36] *3D Models, CAD Solids - 3D CAD Browser*, 2001-2019 (accessed 6/17/2019). URL `http://www.3dcadbrowser.com/`

[37] *GAMMA3: Automatic mesh generation and adaptation methods*, (accessed 6/17/2019). URL `https://team.inria.fr/gamma3/`

[38] *MeshLab*, (accessed 6/21/2019). URL `http://www.meshlab.net`

[39] Si H. *Tetgen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator*, (accessed 6/17/2019). URL `http://wias-berlin.de/software/`