
Parallel Hex Meshing from Volume Fractions

Steven J. Owen, Matthew L. Staten, Marguerite C. Sorensen

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

Summary. In this work, we introduce a new method for generating Lagrangian computational meshes from Eulerian-based data. We focus specifically on shock physics problems that are relevant to Eulerian-based codes that generate volume fraction data on a Cartesian grid. A step-by-step procedure for generating an all-hexahedral mesh is presented. We focus specifically on the challenges of developing a parallel implementation using the message passing interface (MPI) to ensure a continuous, conformal and good quality hex mesh.

Key words: grid-based, overlay grid, hexahedral mesh generation, parallel meshing, non-manifold

1 Introduction

Computational simulation must often be performed on domains where materials are represented as scalar quantities or volume fractions at cell centers of a Cartesian or octree-based grid. Common examples include bio-medical, geotechnical or shock physics calculations where interface boundaries are represented only as discrete statistical approximations. Sandia Lab's, CTH code, is an example of an application that utilizes an Eulerian grid as its computational domain. The results of a CTH calculation are represented as volume fractions in the individual cells of the domain. In practice, this is represented as a 3-dimensional array of scalar values ranging from 0.0 to 1.0, where 1.0 represents material that completely fills the volume of the cell, and 0.0 represents the absence of material. Values that fall between represent the percentage of material, by volume, that is filling the volume of the cell.

We wish to provide a capability for the results from an Eulerian-based code to be used as input to a Lagrangian, or finite element based code. To accomplish this, the scalar volume fraction data array must be interpreted

[†]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

and converted into a boundary aligned hexahedral mesh that is of sufficient quality to be used in a finite element calculation.

In this work we introduce new approaches to solving the all-hex meshing problem from volume fraction data that specifically address the problem in the context of distributed memory parallel processing. We also introduce improved methods applicable for both serial and parallel processing. For example a new primal-contouring approach is introduced for defining the material domains. We describe a step-by-step procedure that includes new methods for node smoothing, resolving non-manifold conditions as well as defining geometry for parallel subdomains.

The development of general-purpose unstructured hexahedral mesh generation procedures for an arbitrary domain have been a major challenge for the research community. A wide variety of techniques and strategies have been proposed for this problem. It is convenient to classify these methods into two categories: *geometry-first* and *mesh-first*. In the former case, a topology and geometry foundation is used upon which a set of nodes and elements is developed. Historically significant methods such as plastering [1], whisker weaving [2] and the more recent unconstrained plastering [3] can be considered *geometry-first* methods. These methods begin with a well defined boundary representation and progressively build a mesh. Most of these methods define some form of advancing front procedure that requires resolution of an interior void and have the advantage of conforming to a prescribed boundary mesh where resulting element quality is normally high at the boundary. Although work in the area is on-going, the ability to generalize these techniques for a comprehensive set of B-Rep configurations has proven a major challenge and has yet to prove successful for a broad range of models.

In contrast, the mesh-first methods start with a base mesh configuration. Procedures are then employed to extract a topology and geometry from the base mesh. These methods include grid-overlay or octree methods. In most cases these methods employ a Cartesian or octree refined grid as the base mesh. Because a complete mesh is used as a starting point, the interior mesh quality is high, however the boundary mesh produced cannot be controlled as easily as in geometry-first approaches. As a result the mesh may suffer from reduced quality at the boundary and can be highly sensitive to model orientation. In spite of some of the weaknesses of grid-overlay methods, they have proven effective in a variety of applications, especially those with minimal topology or feature capture requirements. In particular, bio-medical models [4] [5] [6], metal forming applications [7] [8], and viscous flow [9] methods. The method we describe in this work also utilizes a mesh-first approach.

As one of the first to propose an automatic overlay-grid method, Schneiders [7] developed techniques for refining the grid to better capture geometry. He utilized template-based refinement operations, later extended by Ito [6] and H. Zhang [10] to adapt the grid so that geometric features such as curvature, proximity and local mesh size could be incorporated. For our implementation, we do not require sharp feature recovery or adaptively refined hexes. Instead,

we focus on generation of a high resolution homogeneous hex mesh from a *flat* grid of volume fraction data.

Y. Zhang [4] [5] and Yin [11] independently propose an approach known as the *dual contouring* method that discovers and builds features into the model as the procedure progresses. The dual contouring method for generating a hexahedral mesh described by Y. Zhang [4] begins by computing intersections of the geometry with edges in the grid. Intersection locations are used to approximate normal and tangent information for the geometry. One point per intersected grid cell is then computed using a minimization procedure that is based upon Hermite approximations from the tangents computed at the grid edges. The base mesh in this case is defined as the dual of the Cartesian grid, using the cell centroids and interpolated node locations at the boundary. The proposed method, although similar in many respects, in contrast uses the grid itself as the basis for the FEA hex mesh rather than the dual of the grid to compute the intersection and normal information. As a result, nodes of the *primal* grid are used in the final mesh rather than generating the mesh from the *dual* entities. This new *primal contouring* approach that we introduce is important for our parallel implementation which avoids splitting individual cells in the domain across multiple processors and also avoids additional interpolation of cell centered data to the nodes of the grid.

2 Algorithm

The following is a brief outline of the procedure used for generating a hexahedral mesh from volume fraction data in parallel.

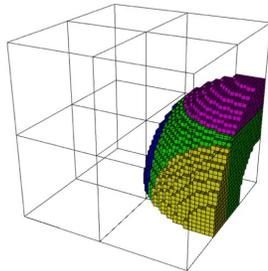


Fig. 1. 8 domains defined showing cells with volume fraction = 1.0

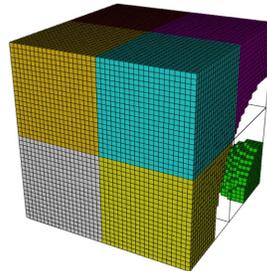


Fig. 2. cells with volume fraction = 0.0

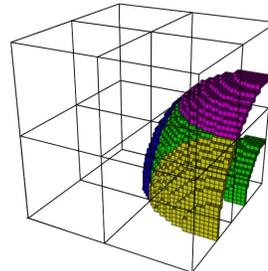


Fig. 3. cells with volume fraction between 0.0 and 1.0

It is assumed that the application that provides volume fraction data, also provides a parallel decomposition of an axis aligned global Cartesian grid. Figure 1 shows an example of an eight processor decomposition containing volume fraction data spread across subdomains. The number of parallel processors used is the same as the number of rectilinear subdomains, where each

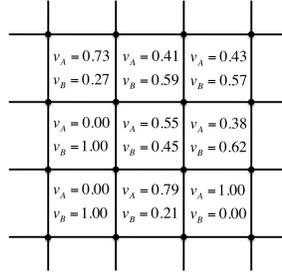


Fig. 4. Establish parallel Cartesian grid

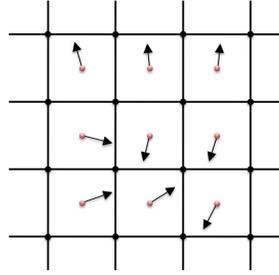


Fig. 5. Estimate gradients at cell centers

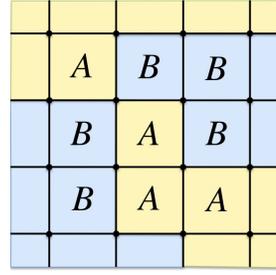


Fig. 6. Assign materials to cells

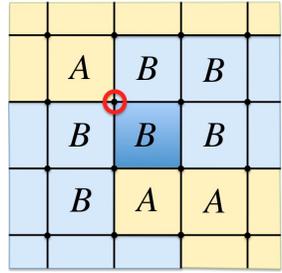


Fig. 7. Resolve non-manifold cases

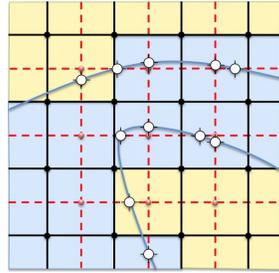


Fig. 8. Compute virtual edge crossings

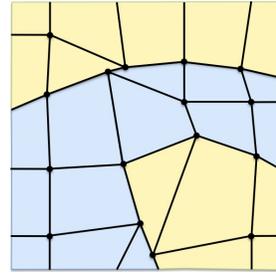


Fig. 9. Move grid points to iso-surface

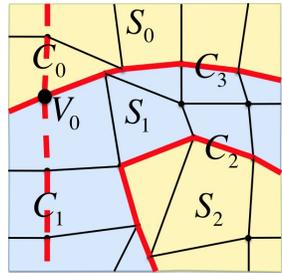


Fig. 10. Create geometry definition

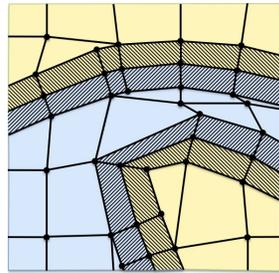


Fig. 11. Insert hex buffer layer

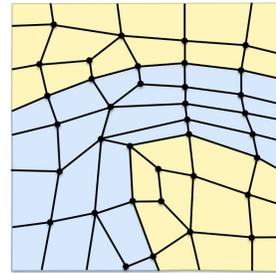


Fig. 12. Smooth

processor assumes the task of generating the complete hex mesh for its portion of the global grid. Also illustrated in figures 1 to 3, each cell of the global Cartesian grid has been assigned at least one scalar volume fraction value, assumed to be at the cell-center of each cell of the grid. Multiple materials may also be represented, where for each cell in the grid, exactly N_{mat} scalar values, v_n are provided, where N_{mat} is the number of materials represented in the model and for each cell in the grid, $\sum_{n=0}^{n < N_{mat}} v_n \leq 1.0$. For each domain in the global Cartesian grid, the following represents the procedure performed

on each processor and is illustrated in figures 4 to 12 and further outlined in the following 9 steps:

Procedure for Generating A Parallel Hex Mesh from Volume Fractions

1. Establish Parallel Cartesian Grid: A light-weight grid data structure with ghosted cells is established to store and work on the data. (Fig. 4)
2. Estimate Gradients at Cell Centers: Based upon the cell-centered data field, gradient vectors are approximated. (Fig. 5)
3. Assign Materials: A dominant material is identified for each cell in the Cartesian grid. (Fig. 6)
4. Resolve Non-Manifold Cases: Cells are added or deleted from the base set of hexes for each material to resolve cases that would result in non-manifold connections in the final mesh. (Fig. 7)
5. Compute Virtual Edge Crossings: Identify virtual edges (connecting adjacent cell centers) that have endpoints bounding the iso-value. (Fig. 8)
6. Move Grid Points to Iso-Surface: Interpolating the virtual edge crossing data, virtual cell centers (grid nodes) are projected to an interpolated iso-surface. (Fig. 9)
7. Create Geometry Definition: A geometry description comprised of volumes, surfaces, curves and vertices is established. (Fig. 10)
8. Insert Hex Buffer Layer: A layer of hexes is inserted at iso-surface boundary and hex elements generated with geometry associativity. (Fig. 11)
9. Smooth: Smoothing procedures are employed for curves, surfaces and volume mesh entities to improve mesh quality. (Fig. 12)

Sections 2.1 to 2.9 provide a more in-depth description of each of the steps of this procedure.

2.1 Establish Parallel Cartesian Grid

For our application, the size of the axis-aligned Cartesian grid for a processor is determined by the Eulerian shock hydro code. This is convenient, since the same domain distribution used in the physics code, can be used in the Lagrangian mesh generation procedure. We can define the Cartesian grid on processor rank p as $\Omega_M^p = \{M_i^r | r = 0, 1, 2, 3\}$ where for example M^0 is a node of the grid, M^1 is an edge, and so forth. The location of grid nodes and size of cells of Ω_M^p is established by defining three independent arrays in each coordinate direction: $X_\Omega = \{x_0, x_1, x_2, \dots, x_{nx+1}\}$, $Y_\Omega = \{y_0, y_1, y_2, \dots, y_{ny+1}\}$, $Z_\Omega = \{z_0, z_1, z_2, \dots, z_{nz+1}\}$, where nx , ny and nz are the number of cells in the grid in coordinate directions x , y and z respectively. Subsequent algorithms described here, utilize the entities $M_i^r | r = 0, 1, 2, 3$, however for our purposes, a lightweight representation of Ω_M^p is established, implicitly defining M_i^r only as needed.

For parallel efficiency, we also establish two layers of ghost cells on Ω_M^p . The global domain, Ω_M^G is itself described as a 3D Cartesian grid containing N_p

subdomains $\Omega_M^p \mid 0 \leq p < N_p$ and $N_p = N_I \cdot N_J \cdot N_K$ where N_I, N_J, N_K are the number of subdomains in each Cartesian direction. For the general case of a processor subdomain on the interior of Ω_M^G , up to 26 neighboring subdomains may be present. Communication is established from processor $p_{I,J,K}$ to all of its neighboring processors $p_{I\pm, J\pm, K\pm}$ and their associated Cartesian grids, $\Omega_M^{p\pm}$. Figure 13 illustrates the communication for a simple 2D configuration. Once ghosted cells have been communicated, $X_\Omega, Y_\Omega, Z_\Omega$ and nx, ny, nz are augmented appropriately for Ω_M^p .

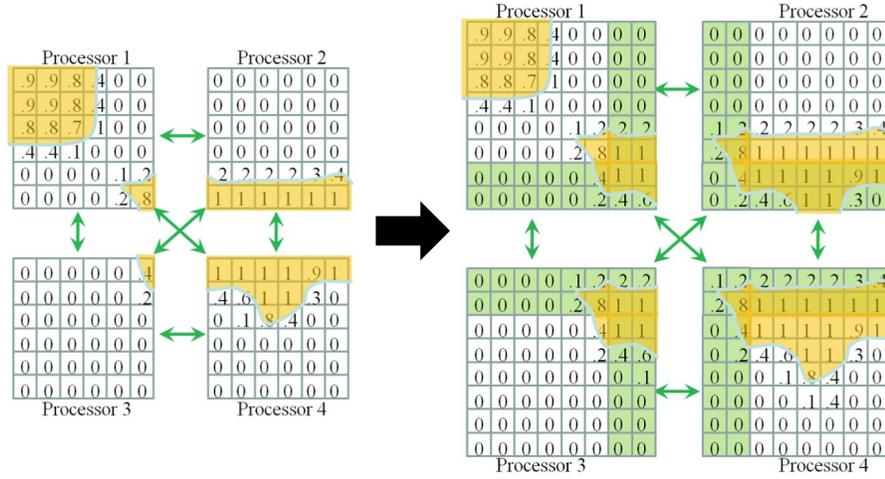


Fig. 13. Two layers of ghosting for each domain is established

2.2 Estimate Gradients at Cell Centers

Our objective is to generate hexes so that interfaces between materials are captured. To do so, we can begin by approximating the gradient at cell centers for each material defined on the domain. For each cell, exactly N_{mat} scalar values $v_n = v_0, v_1, \dots, v_{N_{mat}-1}$ are provided. We can represent the gradient for material n at cell center (i, j, k) as $\nabla v_n(i, j, k) = \left(\frac{\partial v_n}{\partial x}, \frac{\partial v_n}{\partial y}, \frac{\partial v_n}{\partial z} \right)$. For each cell $M_{i,j,k}^3$ in the grid, the differences of 26 neighboring values Δv_n , and cell center coordinate locations $(\Delta x, \Delta y, \Delta z)$ at $M_{i\pm, j\pm, k\pm}^3$ can be used to approximate the gradient. Solving for $\left(\frac{\partial v_n}{\partial x}, \frac{\partial v_n}{\partial y}, \frac{\partial v_n}{\partial z} \right)$ in equation (1) produces the least squares approximation to the gradient for material n .

$$\begin{bmatrix} \sum \Delta x_i^2 & \sum \Delta x_i \Delta y_i & \sum \Delta x_i \Delta z_i \\ \sum \Delta x_i \Delta y_i & \sum \Delta y_i^2 & \sum \Delta y_i \Delta z_i \\ \sum \Delta x_i \Delta z_i & \sum \Delta y_i \Delta z_i & \sum \Delta z_i^2 \end{bmatrix} \begin{Bmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial z} \end{Bmatrix} = \begin{bmatrix} \sum \Delta x_i \Delta (v_n)_i \\ \sum \Delta y_i \Delta (v_n)_i \\ \sum \Delta z_i \Delta (v_n)_i \end{bmatrix} \quad (1)$$

The gradients in much of the grid will be undefined where Δv_n is small or zero. Since the interfaces we seek are defined only where $|\Delta v_n| \gg 0$, we can normally ignore cases where the gradient is undefined.

Note also that for cells at the boundary of the grid, fewer than 26 adjacent cells are available to for the summations in equation (1) to compute ∇v_n . Where neighboring processors are present, this will result in inconsistent results for gradients at the processor boundaries computed on the outer layer of ghost cells. If not resolved, this may effect the smoothness of the grid across processors and whether the nodes conform at all. To avoid this condition, once the gradients are computed, communication is established with neighboring processors, $\Omega_M^{p\pm}$ and gradients in the outer layer of ghost cells on each processor are sent and received via MPI.

2.3 Assign Materials

In this step we must assign each cell $M_{i,j,k}^3$ in the grid to one of the N_{mat} materials in the model. Illustrated in figure 6, this is done simply by identifying material n in $M_{i,j,k}^3$ with the greatest volume fraction v_n . In many cases, the void space, or absence of any material is required to be meshed. In this case, we keep track of the void as a separate material where the volume fraction is defined as $v_{void} = 1.0 - \sum v_n$. Separate lists of cells for each material are then maintained.

2.4 Resolve Non-Manifold Cases

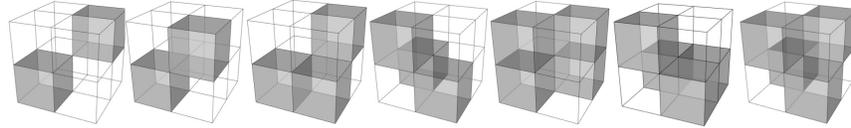


Fig. 14. Seven unique cases for non-manifold conditions in 3D at a node

Figure 6 shows a simple 2D case where materials A and B meet at a non-manifold point. This configuration results in an invalid finite element mesh and must be resolved prior to generating the mesh. Figure 7 shows a simple resolution of the condition by reclassifying the assigned material in one of the cells from material A to B. For 3D, seven unique cases have been identified, as shown in figure 14, where a non-manifold condition may exist at a node. Algorithm 1 illustrates how 3D non-manifold resolution is accomplished by temporarily modifying the volume fraction v_n by a small value, ϵ until all non-manifold conditions have been resolved. To avoid oscillation, the value for ϵ_k incrementally increases using a prime-like progression of floating point values. The function *resolve_non_manifold_at_node()* in algorithm 1 identifies which of the seven unique 3D non-manifold conditions exist at a node

and enumerates hexes to be added or subtracted for the current material. In practice, algorithm 1 normally converges within 2 to 3 iterations.

```

 $\varepsilon_k = \{0.05, 0.07, 0.11, 0.13, 0.17, \dots\}; k = 0;$ 
while non-manifold condition exists do
  foreach material  $n = 0, 1, \dots, N_{mat}-1$  do
    //initialize lists of hexes to add and subtract for material  $n$ 
     $L_{add}(M^3) = \emptyset, L_{sub}(M^3) = \emptyset;$ 
    foreach non-manifold  $(M_i^0) \in \Omega_M^p$  do
       $M_I^3 = M_j^3 | j = 0, 1, \dots, 7 \in M_i^0;$ 
      //add to lists  $L_{add}$  and  $L_{sub}$  from hexes  $M_I^3$ 
      resolve_non_manifold_at_node $(M_I^3, L_{add}, L_{sub});$ 
    end
     $v_n(M_i^3) =$  volume fraction of material  $n$  for  $M_i^3;$ 
    foreach  $M_i^3 \in L_{add}$  do  $v_n(M_i^3) = v_n(M_i^3) + \varepsilon_k;$ 
    foreach  $M_i^3 \in L_{sub}$  do  $v_n(M_i^3) = v_n(M_i^3) - \varepsilon_{k+1};$ 
  end
  reclassify material assignment for  $M_i^3$  in  $\Omega_M^p;$ 
  communicate material assignment of ghost cells to  $\Omega_M^{p\pm};$ 
   $k = k + 1;$ 
end

```

Algorithm 1: Algorithm for resolving non-manifold conditions in Ω_M^p

Note also that a parallel communication step is required following each iteration of this procedure. Because identification of non-manifold conditions depends upon checking the status of all surrounding cells of M_i^0 , the non-manifold state of grid nodes at the boundary can only be established through interprocessor communication.

2.5 Compute Virtual Edge Crossings

The next stage in defining the material interfaces for the hexes is to compute *virtual edge crossings*. We would like to compute all locations on Ω_M^p where the iso-value $s = 0.5$ crosses one of the edges of the grid. We will use these locations in the next section to help move the grid nodes to the interpolated iso-surface. We choose $v_n = 0.5$ as the most likely volume fraction value where the interface surface will exist. For convenience, rather than directly computing the crossing locations on the edges, M_i^1 of the grid, we define the *virtual edges* of the grid as the segments connecting the midpoints of adjacent cells. We can uniquely identify a virtual edge in the grid from a face M_i^2 . The midpoint of its two adjacent cells, $M_{j=0,1}^3$ can be defined as $P_{j=0,1} = center(M_{j=0,1}^3)$. Similarly, the volume fraction for material n in each adjacent cell to face M_i^2 can be defined as $(v_n)_{j=0,1}$. Equation (2) can then be used to compute a location $P_{cross}(M_i^2)$ for any vir-

tual edge where its two associated cells are assigned to different materials, $material(M_0^3) \neq material(M_1^3)$.

$$P_{cross} = P_0 + \frac{s - (v_n)_0}{(v_n)_1 - (v_n)_0} (P_1 - P_0) \quad (2)$$

Also required is the normal or gradient of material n at the location of P_{cross} which can be interpolated similarly. The gradient at cells $M_{j=0,1}^3$ can be described as $(\nabla v_n)_{j=0,1}$, (see section 2.2).

$$N_{cross} = (\nabla v_n)_0 + \frac{s - (v_n)_0}{(v_n)_1 - (v_n)_0} [(\nabla v_n)_1 - (\nabla v_n)_0] \quad (3)$$

Since P_{cross} and N_{cross} values may be used frequently, they are precomputed for each material and associated with its respective virtual edge M_i^2 in Ω_M^p .

2.6 Move Grid Points to Iso-Surface

With information computed up to this stage of the procedure, we can now compute locations for nodes in the grid that will form the interface between materials. A node M_i^0 in Ω_M^p is defined as *movable* if at least 2 unique materials are identified from its eight surrounding cells $M_{j=0,1,\dots,7}^3$. One common material assigned to all $M_{j=0,1,\dots,7}^3$, indicates an interior node.

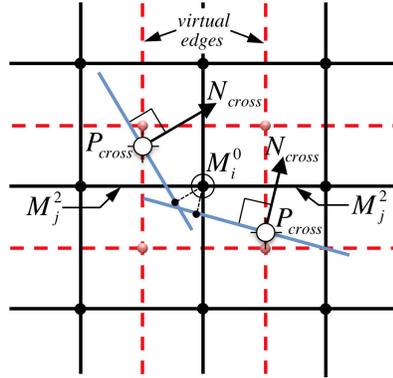


Fig. 15. Variables used to move node M_i^0 to an interpolated iso-surface

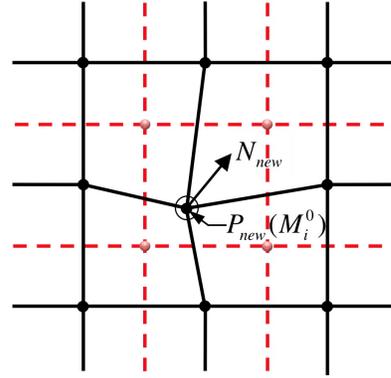


Fig. 16. Node M_i^0 after having been moved to the iso-surface

Figures 15 and 16 illustrate the procedure for computing the new location, P_{new} for M_i^0 that has been identified as at a material interface. The 12 grid faces in Ω_M^p that contain node M_i^0 can be defined as $M_{j=0,1,\dots,11}^2$. These faces also uniquely define a virtual cell who's 12 virtual edges are defined by $M_{j=0,1,\dots,11}^2$ with M_i^0 as the centroid. Using the values computed for P_{cross} and N_{cross} for each M_j^2 (see section 2.5) we can define tangent planes at the

material interface that cross the virtual edges. Equation (4) computes P_{new} by taking an average of the projection of M_i^0 onto each of the tangent planes.

$$P_{new} = \frac{1}{nc} \sum_{i=0}^{i < nc} P_0 - (N_{cross})_i \cdot (P_0 - (P_{cross})_i) \times (N_{cross})_i \quad (4)$$

$$(N_{new})_n = \left| \sum_{i=0}^{i < nc_n} (N_{cross})_i \right| \quad (5)$$

where nc is the number of virtual edge crossings where a value for P_{cross} has been computed on the virtual cell surrounding M_i^0 and P_0 is the initial location of M_i^0 . We also compute $(N_{new})_n$ as the normalized average of $(N_{cross})_i$. This provides the local surface normal information needed for the subsequent buffer layer insertion and smoothing operations. The subscript n in equation (5) indicates that a separate normal is computed with respect to each material using only $(N_{cross})_i$ vectors that originate from material n .

Note that we have not distinguished between materials for the computation of P_{new} . As a result, all materials that have virtual edge crossings defined on M_i^0 's virtual cell, will contribute to the new location. Where multiple materials meet at M_i^0 , this has the effect of averaging the contribution from all materials resulting in a reasonable approximation of the surfaces at that point.

2.7 Create Geometry Definition

Having captured the material interfaces in Ω_M^p , we turn our attention to improving the mesh so that hexes are of sufficient quality for FE analysis. Prior to doing this, however, we have found it useful to generate a boundary representation or *B-Rep* of the hex structure that captures the material interfaces and domain boundaries. This will prove valuable in the next stages when we add a buffer layer of hexes, smooth the elements as well as assisting in encapsulation and transfer of data. The B-Rep on processor p , which we will define as Ω_G^p consists of entities $\{G_i^r | r = 0, 1, 2, 3\}$ where G^0 is a vertex of the B-Rep, G^1 is a curve, G^2 a surface and G^3 , a volume. To generate the B-Rep, we must find groups of grid entities, M_i^r of dimension r that will be assigned to, or *owned* by corresponding B-Rep entities, G_i^r of the same dimension. Figure 17 shows an example of the resulting B-Rep on one processor subdomain of the model shown in figures 1 to 3. Note that geometry entities are generated at the processor interfaces. To better facilitate smoothing, one layer of ghost cells is also used in the definition of the geometry. This ensures that two layers of overlapping hexes are established between domains Ω_G^p and $\Omega_G^{p\pm}$ facilitating the subsequent Jacobi smoothing procedure.

Starting with volumes, G_i^3 are established by gathering contiguous sets of cells $M_{i,j,k}^3$ with the same material assignment. Surfaces G_i^2 are then built by *skinning* or traversing the faces M_i^2 at the boundary of each volume, including the boundaries at processor subdomains. Grid faces, M_i^2 at a subdomain

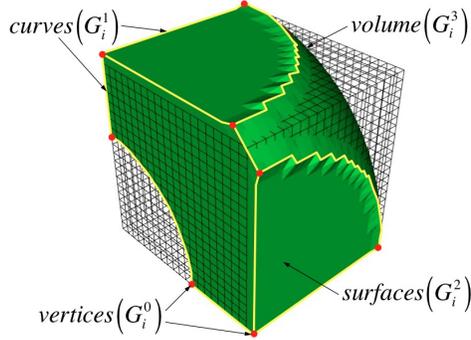


Fig. 17. A B-Rep, Ω_G^p , is built from the entities of the Cartesian grid Ω_M^p

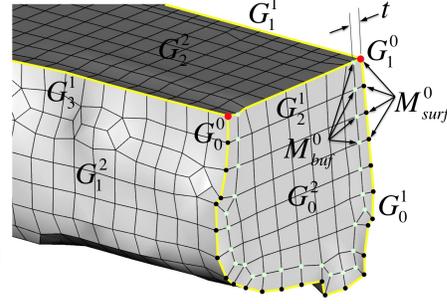


Fig. 18. Buffer layer hexes are generated to be continuous across processor boundaries

boundary can be distinguished from those on the interior of the grid which will facilitate generation of curves, G_i^1 from grid edges M_i^1 along their interface. Curves can also be generated along the edges where at least three materials meet. Finally, vertices G_i^0 are established where more than two surfaces share a common grid node M_i^0 .

2.8 Insert Hex Buffer Layer

The first stage in improving element quality is to insert a layer of hexes at all material interfaces. To accomplish this we identify all nodes on the material interfaces, M_{surf}^0 . These are the same nodes for which we have computed P_{new} and N_{new} in equations 4 and 5. For each M_{surf}^0 , one offset buffer layer node M_{buf}^0 must be generated for each material adjacent M_{surf}^0 . The location of M_{surf}^0 can be defined as:

$$P(M_{buf}^0)_n = P(M_{surf}^0) + tN(M_{surf}^0)_n \quad (6)$$

where t is the thickness of the buffer layer and $N(M_{surf}^0)_n$ is the normal for material n computed in equation 5. We chose the thickness of the buffer layer to be $\frac{1}{4}$ the diagonal distance of the grid cell.

Because the buffer layer must be continuous across subdomain boundaries, the geometry ownership described in section 2.7 can be used to aid in defining its placement. Figure 18 shows an example where a volume G^3 intersects the subdomain boundary. In this case, three surfaces, $G_{0,1,2}^2$ enclose the volume, however, only one surface, G_1^2 , contains nodes M_{surf}^0 . Instead, surfaces G_0^2 and G_2^2 are used only to *cap* the volume where it intersects the boundary. For our purposes, we designate G_1^2 an *interior* surface, and G_0^2 and G_2^2 as *capping* surfaces. Using the same criteria, in this example we designate the curves $G_{0,1,3}^1$ as *interior* and G_2^1 as *capping*.

With this information, we visit each capping entity $(G_i^{1,2})_{cap}$ and insert buffer edges or quads at its boundaries. For example, for curve G_2^1 in figure 18, two buffer mesh edges M_{buf}^1 are inserted into G_2^1 at vertices G_0^0 and G_1^0 . In a similar manner, we insert buffer quad elements M_{buf}^2 in surface G_0^2 only adjacent to its interior curve G_0^1 . Finally, the buffer hex elements M_{buf}^3 are constructed adjacent the quads on interior surface G_1^2 .

2.9 Smooth

We now turn our attention to improving the element quality by smoothing. We require that the subdomain boundaries not impose artificial constraints on the quality and location of the nodes. Furthermore we require parallel-serial consistency, or in other words, the same mesh must be generated regardless of the number of processors or the location of subdomain boundaries. To accomplish this we establish communication between Ω_M^p and neighbors, $\Omega_M^{p\pm}$.

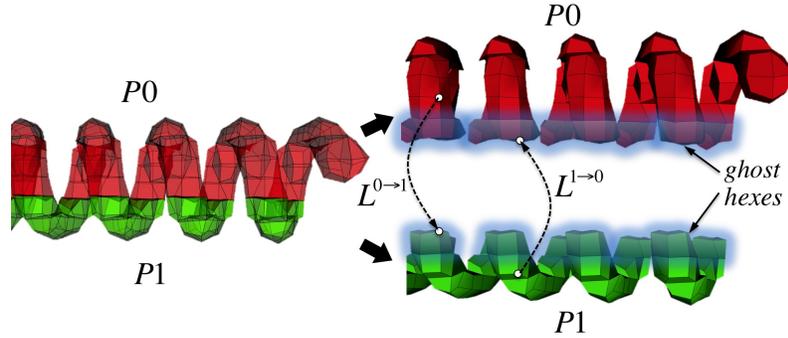


Fig. 19. Node locations on ghost hexes are communicated following each smoothing iteration

Following each iteration of smoothing, to ensure a continuous mesh with identical node locations computed at subdomain boundaries, a communication step must be performed. To do so, we require all nodes on processor p that are within its ghost layer, receive updated nodal coordinates from its neighbors $p\pm$. To facilitate this, we first identify lists $L(M_i^0)^{p \rightarrow p\pm}$ of all nodes M_i^0 in Ω_M^p that are part of the two ghost layers at its boundary, and that have been previously assigned to a geometry entity $G^{0,1,2,3}$. This includes any nodes generated to form the buffer layer from entities in the ghost layers. Each list, $L(M_i^0)^{p \rightarrow p\pm}$, will contain nodes that are ghosted from a unique neighboring processor, $p\pm$ to the host processor p . The locations of the nodes in $L(M_i^0)^{p \rightarrow p\pm}$ on processor p , are then sent to neighboring processors $p\pm$. Neighbor processors, $p\pm$ then determine the corresponding node in $\Omega_M^{p\pm}$ for each location sent to it from processor p and form corresponding lists $L(M_i^0)^{p\pm \rightarrow p}$. These lists comprise all nodes which must be sent to neighboring proces-

sors following each iteration of smoothing. Corresponding nodes M_i^0 on $\Omega_M^{p\pm}$ from locations received from Ω_M^p can be identified using a spatial tree search since locations should be identical on both processors. The lists $L(M_i^0)^{p\pm \rightarrow p}$ must be set up prior to any smoothing, since node locations will change once smoothing has begun. Figure 19 illustrates a simple two processor example where node locations in each of the lists $L^{0 \rightarrow 1}$ and $L^{1 \rightarrow 0}$ are communicated.

Node smoothing consists of several iterations of successively smoothing nodes M_j^0 on geometric entities starting with curves $G_i^1(M_j^0)$, surfaces $G_i^2(M_j^0)$, and then volumes $G_i^3(M_j^0)$. Rather than traditional Gauss-Seidel smoothing that relies on an order-dependent incremental update of node locations, we use a Jacobi approach that uses the initial locations of the nodes at the start of the iteration.

For most cases we can neglect curve smoothing, as most interior and capping curves reside in ghost regions which are updated by the parallel communication discussed previously. For curves at the absolute domain boundaries a simple one-dimensional Laplacian smooth is performed on a piecewise quadratic approximation of the nodes.

For interior surfaces, since an explicit representation of the surface is not available, we use a quadric approximation of the nodes M_{surf}^0 [13] that is centered at the node to be smoothed $P_k(x_k, y_k, z_k)$.

$$Q_k(x, y) = z_k + a_{k2}(x - x_k) + a_{k3}(y - y_k) + a_{k4}(x - x_k)^2 + a_{k5}(x - x_k)(y - y_k) + a_{k6}(y - y_k)^2 \tag{7}$$

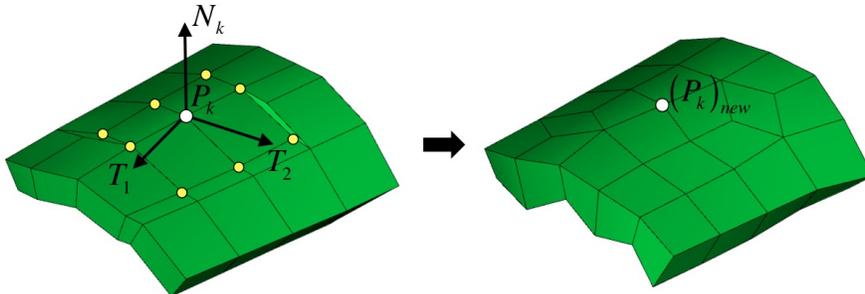


Fig. 20. Quadric approximation of surface from surrounding nodes at P_k is performed

We first transform nodes attached by edges to P_k to a local coordinate system centered at P_k with orientation defined by (N_k, T_1, T_2) as shown in figure 20, where N_k is the surface normal at P_k and (T_1, T_2) are orthogonal tangent vectors. Coefficients $a_{k2, k3, \dots, k6}$ for equation (7) can then be computed by solving the linear system:

$$\begin{bmatrix} \sum w_i x^2 & \sum w_i xy & \sum w_i x^3 & \sum w_i x^2 y & \sum w_i x^2 y^2 \\ \sum w_i xy & \sum w_i y^2 & \sum w_i x^2 y & \sum w_i xy^2 & \sum w_i xy^3 \\ \sum w_i x^3 & \sum w_i x^2 y & \sum w_i x^4 & \sum w_i x^3 y & \sum w_i x^2 y^2 \\ \sum w_i x^2 y & \sum w_i xy^2 & \sum w_i x^3 y & \sum w_i x^2 y^2 & \sum w_i xy^3 \\ \sum w_i xy^2 & \sum w_i y^3 & \sum w_i x^2 y^2 & \sum w_i x^2 y^2 & \sum w_i y^4 \end{bmatrix} \begin{pmatrix} a_{k2} \\ a_{k3} \\ a_{k4} \\ a_{k5} \\ a_{k6} \end{pmatrix} = \begin{pmatrix} \sum w_i xz \\ \sum w_i yz \\ \sum w_i x^2 z \\ \sum w_i xyz \\ \sum w_i y^2 z \end{pmatrix} \quad (8)$$

where $x = x_i - x_k$, $y = y_i - y_k$, $z = z_i - z_k$ and w_i is an inverse distance weight. A Laplacian smoothing operation can then be performed on node P_k to get a smoothed location P'_k in the local coordinate system. The point P'_k is then projected to the quadric surface, also in the local coordinate system using:

$$(P_k)_{local} = \begin{cases} (P'_k - P_k) \cdot T_1 \\ (P'_k - P_k) \cdot T_2 \\ a_{k2}x_k + a_{k3}y_k + a_{k4}x_k^2 + a_{k5}x_k y_k + a_{k6}y_k^2 \end{cases} \quad (9)$$

Finally, the new location $(P_k)_{new}$ in the original coordinate system is computed as:

$$(P_k)_{new} = P_k + (P_k)_{local}^T \begin{Bmatrix} T_1 \\ T_2 \\ N_k \end{Bmatrix} \quad (10)$$

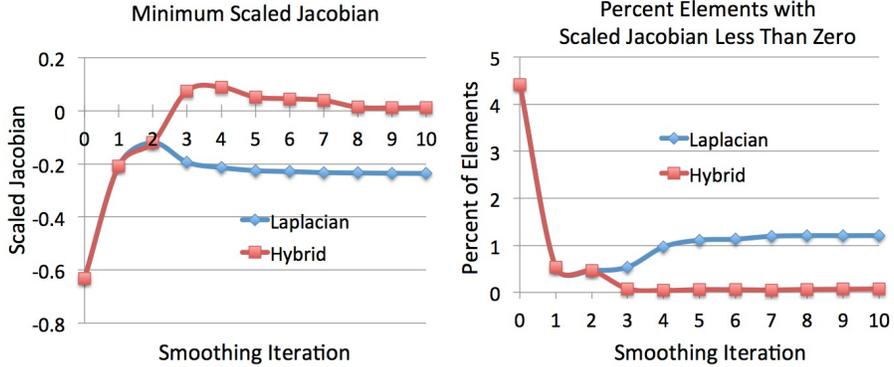


Fig. 21. Smoothing

For volumes, we also use a Jacobi Laplacian smoothing method to smooth interior nodes $G_i^3(M_j^0)$. In practice we have found that Laplacian smoothing alone is not sufficient to generate acceptable quality. Instead, after an initial two iterations of Laplacian smoothing, an optimization-based approach using the Mesquite toolkit [12] is used for subsequent iterations. This requires both untangling and mesh optimization which the ShapeImprovement tool in Mesquite is able to provide. Figure 21 shows typical results over 10 iterations

of Jacobi smoothing on a model with approximately 500,000 elements. The left graph compares minimum Scaled Jacobian using just Laplacian smoothing vs. using combined Laplacian and the Mesquite ShapeImprovement smoother after the second iteration. On the right, results from the same model are shown, except that the percent of elements with Scaled Jacobian less than zero is displayed. These results show that two iterations of Laplacian smoothing, combined with two iterations of ShapeImprovement optimization is sufficient to drive the elements to a computable range.

In practice, Laplacian smoothing is much faster than optimization-based smoothing and is able to make enormous improvements with very small cost. For this reason, we do not start with the Mesquite ShapeImprovement optimization. Also, to improve efficiency, we limit application of the ShapeImprovement procedure to only those patches of elements where the scaled Jacobian mesh quality falls below a threshold of 0.2. These small patches of elements must be consistently identified on all processors, therefore, an additional communication step is required to communicate element patches smoothed in ghost regions.

3 Examples

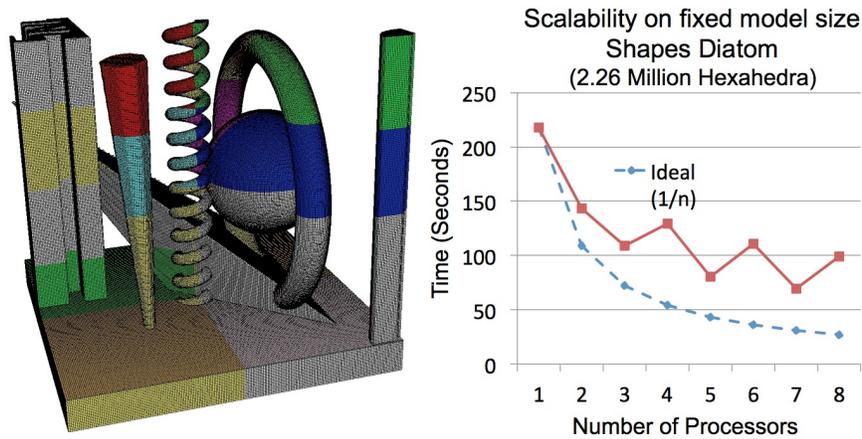


Fig. 22. Hex mesh constructed on eight processors and its associated timing data

We show several examples to illustrate the proposed capability. The first example shown in figures 22 and 23 illustrates several geometric primitives, or *shapes diatoms*, that have been converted to volume fractions on a Cartesian grid and then hex meshed. Scalability results, shown in figure 22 are still preliminary and illustrate up to an eight processor decomposition generating approximately 2.26 million elements. Timing results include I/O beginning with initial volume fraction data with one file per processor and resulting

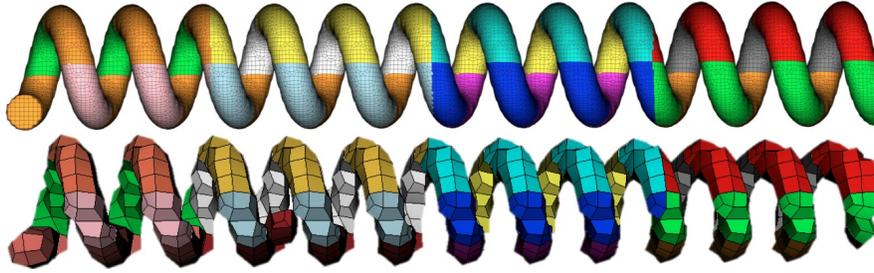


Fig. 23. Detail of helix in shapes diatom model in fig. 22 at two different resolutions

in a Lagrangian mesh file for each processor that contain conforming node locations at the subdomain interfaces. The oscillating pattern observed in the timing results is most likely an artifact of the domain decomposition of odd vs even configurations. Similar timing results were also observed in subsequent examples. The two helix images shown in figure 23 are an enlarged view of one of the objects in figure 22 modeled at two different resolutions where color represent different processors. Note the smoothness of the surfaces at higher resolution even across processor boundaries.

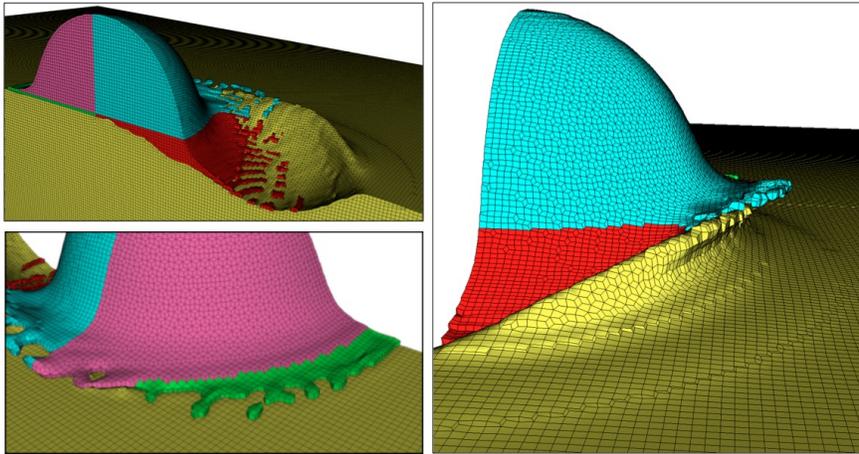


Fig. 24. Hex mesh of ball impact on plate at one selected time step.

In figure 24 a series of results at time step intervals have been computed with CTH of a sphere impacting a plate. The results have been exported as volume fraction data and processed at one selected time step using the proposed procedure. In figure 25 we illustrate a hex mesh at two time steps of a simulated pipe bomb explosion that was computed with CTH.

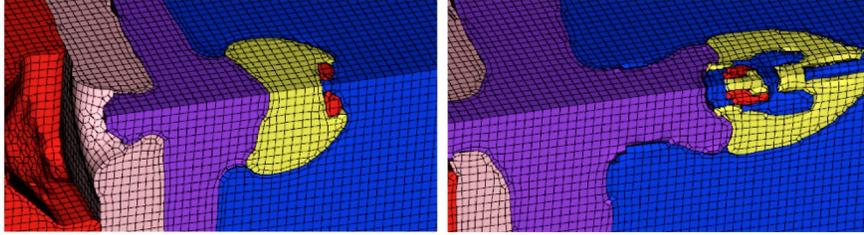


Fig. 25. Close-up view of hex meshes generated at two different time steps of a simulated pipe bomb explosion

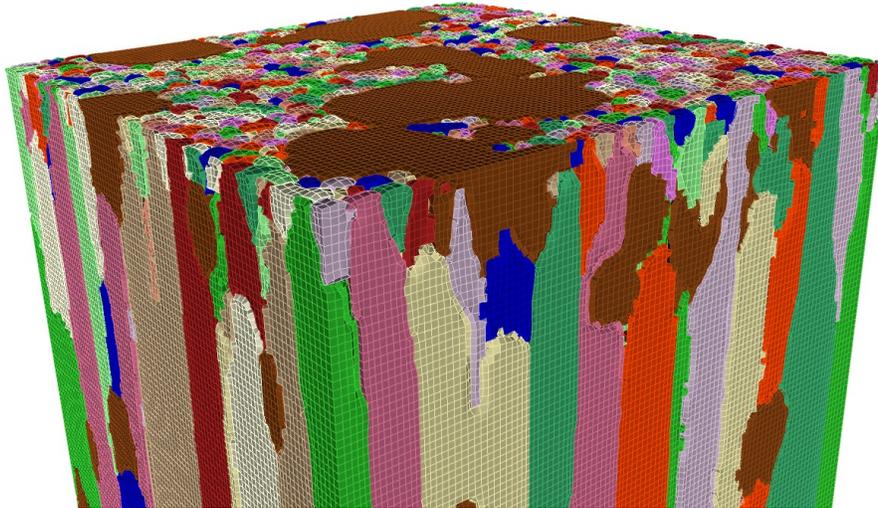


Fig. 26. Hex mesh of simulated grain microstructure with 15 different materials.

The final example, shown in figure 26 is a hex mesh generated from volume fraction data, also computed from CTH, representing a simulated microstructure of a material having columnar type grain structure with isolated porosity. This model contains 15 materials and 1.6 million elements generated on eight processors.

4 Conclusion

This work introduces a step-by-step procedure for generating a hexahedral mesh from volume fraction data defined on a Cartesian grid. Contributions include improved methods applicable for both serial and parallel processing including a new primal-contouring approach for defining multiple material domains, new methods for node smoothing, resolving non-manifold conditions as well as defining geometry for parallel subdomains. We recognize that we are still in the research phases of this project, and that there are many areas left

to explore. We would anticipate that the results of such research would move this technology towards a scalable tool that can be robustly used for coupling Eulerian and Lagrangian codes. We do however offer that these methods improve on existing techniques proposed in the literature particularly as they apply to parallel mesh generation using overlay grid methods.

References

1. Blacker TD, Meyers RJ (1993) Seams and Wedges in Plastering: A 3D Hexahedral Mesh Generation Algorithm, *Engineering with Computers*, 2(9):83–93
2. Tautges TJ, Blacker TD, Mitchell SA (1996) The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes, *International Journal for Numerical Methods in Engineering*, 39:3327–3349
3. Staten ML, Kerr RA, Kerr, Owen SJ, Blacker TD (2006) Unconstrained Paving and Plastering: Progress Update, In: *Proceedings, 15th International Meshing Roundtable* 469–486
4. Zhang Y, Bajaj CL (2006) Adaptive and Quality Quadrilateral/Hexahedral Meshing from Volumetric Data. *Computer Methods in Applied Mechanics and Engineering* 195: 942–960
5. Zhang Y, Hughes TJR, Bajaj CL (2007) Automatic 3D Mesh Generation for a Domain with Multiple Materials. *Proceedings of the 16th International Meshing Roundtable*: 367–386
6. Ito Y, Shih AM, Soni BK, (2009) Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates, *International Journal for Numerical Methods in Engineering*, 77(13):1809–1833
7. Schneiders R, Schindler F, Weiler F (1996) Octree-based Generation of Hexahedral Element Meshes, In: *Proceedings of the 5th International Meshing Roundtable*, 205–216
8. Kwak DY, Im YT (2002) Remeshing for metal forming simulations - Part II: Three-dimensional hexahedral mesh generation, *International Journal for Numerical Methods in Engineering*, 53:2501–2528
9. Tchon KF, Hirsch C, Schneiders R (1997) Octree-based Hexahedral Mesh Generation for Viscous Flow Simulations, *American Institute of Aeronautics and Astronautics A97-32470* 781–789
10. Zhang H, Zhao G (2007) Adaptive hexahedral mesh generation based on local domain curvature and thickness using a modified grid-based method, *Finite Elements in Analysis and Design*, 43:691–704
11. Yin J, Teodosiu (2008) Constrained mesh optimization on boundary, *Engineering with Computers* 24:231–240
12. Brewer M, Freitag-Diachin L, Knupp P, Leurent T and Melander D, (2003) The Mesquite Mesh Quality Improvement Toolkit, *Proceedings, 12th International Meshing Roundtable*, 239–250
13. Jones NL (1990) Solid Modeling of Earth Masses for Applications in Geotechnical Engineering, Dissertation, University of Texas, Austin