SANDIA REPORT

SAND2019-8967

Printed July 2019

Mesh Generation for Microstructures

Steven J. Owen, Corey D. Ernst, Judith A. Brown, Hojun Lim, Kevin N. Long, James W. Foulk III, Nathan W. Moore, Corbett Battaile, Theron Rodgers

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

A parallel, adaptive overlay grid procedure is proposed for use in generating all-hex and tetrahedral meshes for stochastic (SVE) and representative (RVE) volume elements in computational materials modeling. The mesh generation process is outlined including several new advancements such as data filtering to improve mesh quality from voxelated and 3D image sources, improvements to the primal contouring method for constructing material interfaces and pillowing to improve mesh quality at boundaries. We show specific examples in various applications that have benefitted from the proposed mesh generation procedure and illustrate results of the procedure with several practical mesh examples. We also include an extended Appendix with practical examples for using the proposed methods.

ACKNOWLEDGMENT

We wish to acknowledge the work of many people on the Cubit/Meshing team at Sandia that have contributed to the ongoing research and development effort that is Sculpt. We also acknowledge the many individuals who have requested capabilities that have led to further development.

Funding for Sculpt has been from a variety of sources including DOE Advanced Simulation Computing (ASC) Integrated Codes program, Advanced Technology, Development and Mitigation (ATDM), and Goodyear Inc.

This work was also supported by the Laboratory Directed Research & Development program at Sandia National Laboratories from the "Stochastic Shock in Advanced Materials." project.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Determining relationships between material behavior at different length scales is an active area of research and a key part of current integrated computational materials engineering initiatives [14][30]. Computational materials modeling efforts that explicitly resolve microstructure and/or meso-scale material features play an important role in understanding mechanical behavior, deformation and failure mechanisms that ultimately drive macroscale material behavior. However, most materials have features at this scale that are geometrically irregular and present many challenges for mesh generation. Furthermore, it is often necessary to simulate the response of many unique realizations of the material microstructure to capture stochastic effects of variability in feature arrangement and to ensure that the predicted response is not unique to local features in any particular realization. This is particularly important when smaller stochastic volume elements (SVEs) are used to mitigate computational expense if the required size of a representative volume element (RVE) is large. Although tetrahedral meshes may be acceptable, in some cases, hex meshes may be necessary to enable certain features of the analysis or maintain compatibility with other meshed components. The meshing step can quickly become prohibitively expensive unless the complex microstructural features can be meshed in an automated way. Thus, a robust meshing algorithm that can quickly generate meshes on hundreds of material volume elements (RVE or SVE) is an enabling feature to perform comprehensive computational studies of material microstructure behavior.

Current methods for automatic hexahedral mesh generation can be classified as *geometry-first*, or *mesh-first*. Where an explicit geometry representation such as a CAD design model is used, geometry-first approaches can be used to generate high quality, crafted meshes using block-structuring [3] [1] and pave and sweep [33] procedures. These methods begin with reference geometry that must be interactively cleaned-up and decomposed to admit a limited set of topologic meshing primitives. Geometry-first methods are impractical for computational materials modeling where methods must be completely automated and where the input can come from 3D image based data with highly complex material interactions. Instead, we utilize a mesh-first approach that begins with an overlay grid that is locally modified to incorporate microstructural features that will result in a conformal hexahedral mesh of the RVE.

A limited set of literature is available describing conformal hexahedral meshing procedures for computational materials modeling. Qian [32] and Zhang [39] propose procedures that build complex microstructures based upon the dual contouring approach introduced in [38]. Both hexahedral and tetrahedral approaches are described where meshes are generated for multiple material grain structures including procedures for identifying and resolving complex topological interactions as well as the ability to produce smooth interfaces using geometric flow-based smoothing. These works illuminate important aspects peculiar to microstructures, however reported mesh quality [32] can be marginal at grain interfaces and do not appear to provide for distributed algorithms for scalable applications.

For our purposes we seek both tetrahedral and all-hex meshing procedures for application to computational materials modeling that is fully automatic. Methods that require any user interaction to clean or decompose the geometry are impractical for this application. The ability to control the execution of the meshing tool via scripting is also desirable to facilitate rapid execution of hundreds of stochastic simulations. We note that the target analysis codes require computable quality hex or tet elements with smooth conformal interfaces between materials. The mesh generation method should also support multiple input sources including voxelated, volume fraction and analytic geometry. Topologic complexity with hundreds or even thousands of separate grain structures are also necessary for this application. Finally, the ability to rapidly run hundreds of stochastic models with meshes exceeding tens of millions of elements make distributed computing a vital requirement.

This work proposes an overlay grid procedure that builds on Sandia National Laboratories' Sculpt [21]. In [26, 27] we first introduced a parallel overlay grid procedure based on volume fractions for arbitrary geometry and later describe the smoothing procedures for this method in [19]. In [22, 23] we provide a validation of meshes produced from Sculpt in application with computational mechanics codes and in [24, 25], the method is extended to incorporate a new adaptive 2-refinement technique. This work extends and revises [20] which addresses the specific problems encountered in computational materials modeling. New procedures are introduced to ensure the preceding requirements are met and that the resulting meshing tool can be used for practical, robust and efficient computational materials modeling.

2. ALGORITHM

2.1. OVERVIEW

The proposed methodologies are based upon an overlay grid method where a Cartesian grid or any unstructured mesh is used as the basis for a finite element mesh. The proposed procedure works directly from a volume fraction representation where $\sum_{j=0}^{k} vf(i,j) = 1$, with i a cell index and j a material index. With no explicit geometric or topology representation from which to work, the procedure extracts a boundary representation (B-Rep) topology and recovers approximated geometric interfaces from the volume fraction data.

2.1.1. Meshing Procedure

In this work we propose a series of steps or procedures that have proven effective in building hexahedral and tetrahedral meshes for computational materials modeling on stochastic or representative volume elements. The following is an outline of the procedure used for generating hex and tet meshes with their corresponding section in this document:

The following methods are common to generating both hex and tet meshes

1. *Define overlay grid:* Define or import a grid or mesh overlapping the domain that will serve as the basis for the final hex or tet mesh. (fig. 2-1(a))(sec. 2.2)
2. *Process input data:* Import and convert material data to volume fractions if not already. (fig. 2-1(a))(sec. 2.3)
3. *Distribute data for parallelism:* Distribute Cartesian volume fraction data to multiple processors via MPI. (sec. 2.4)
4. *Refine or coarsen:* Adaptively refine or coarsen the Cartesian grid to build a conformal unstructured base grid on which to build the geometry and mesh. (sec. 2.5)
5. *Compute material gradients:* A gradient field is approximated in order to establish normal vectors at material interfaces. (fig. 2-1(b))(sec. 2.6)
6. *Filter input data:* Assign each cell of the base grid to its dominant material. Modify the assignment to eliminate non-manifold conditions and reduce potential mesh quality issues.(fig. 2-1(c))(sec. 2.7)
7. *Construct B-Rep topology graph:* Construct volumes, surfaces, curves and vertices and their mesh entity associations. (fig. 2-1(g))(sec. 2.8)

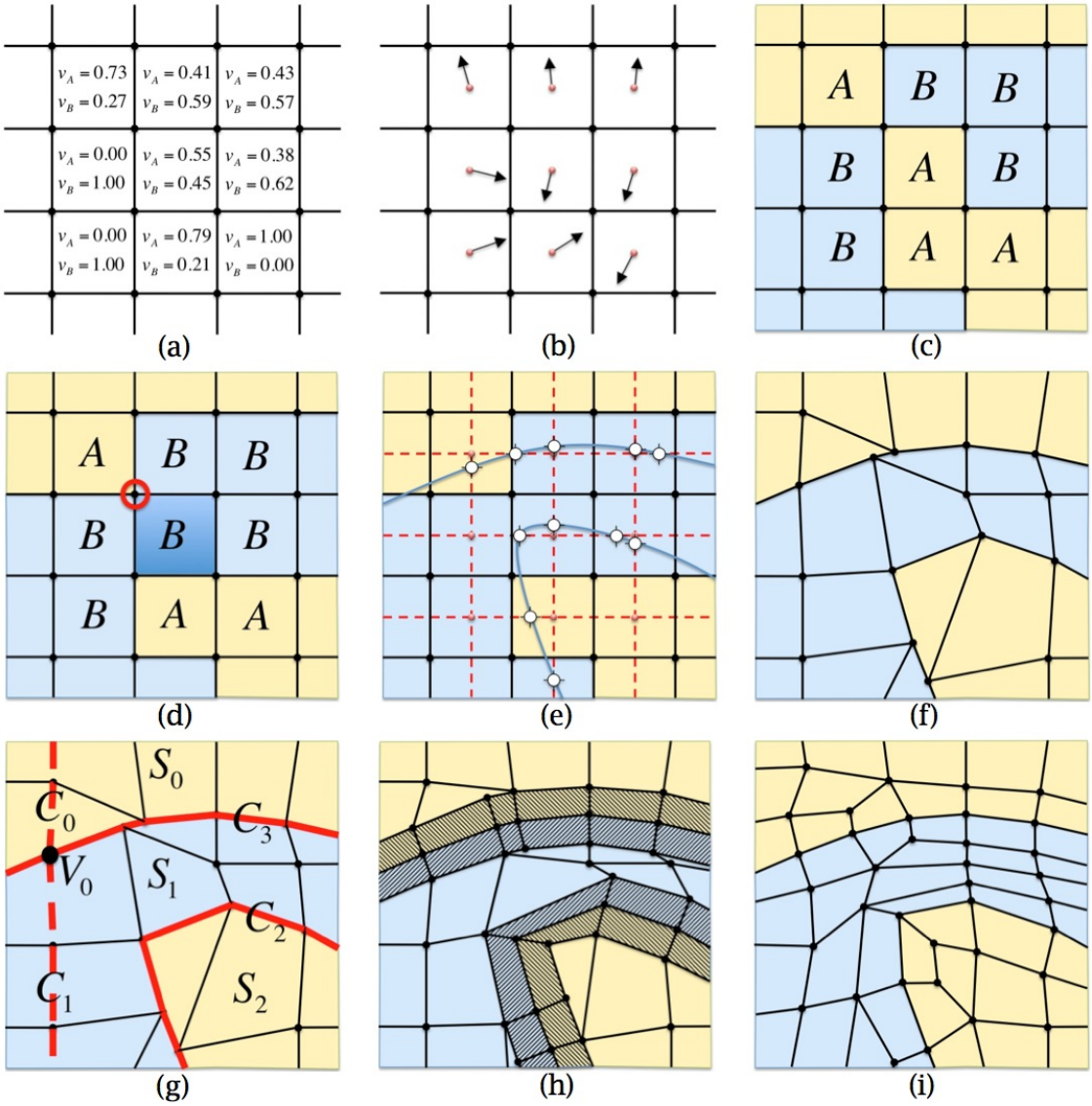


Figure 2-1. A visual representation of the steps used for generating a hex mesh from volume fraction data: (a) Establish parallel Cartesian grid. (b) Estimate gradients at cell centers. (c) Assign materials to cells. (d) Resolve non-manifold cases. (e) Compute dual edge crossings. (f) Move grid points to iso-surface. (g) Create geometry definition. (h) Insert hex buffer layer. (i) Smooth.

8. *Recover geometric interfaces:* Compute initial placement of nodes to approximate material interfaces. (fig. 2-1(e-f))(sec. 2.9)

For tet mesh generation: (sec. 2.10)

9. *Split surface quads:* Split quadrilaterals defining the interface surfaces into triangles, ensuring acceptable quality triangles are maintained. (sec. 2.10.1)

10. *Collapse small edge and surfaces:* Collapse small curves and surfaces and update geometric topology to avoid poor quality tets. (sec. 2.10.2)
11. *Build tet mesh:* Export faceted representation of interfaces and their topology. Tet mesh is generated in Cubit with third party meshing solution. (sec. 2.10.3)

For hex mesh generation: (sec. 2.11)

9. *Build interior hex elements:* Construct topology for hex elements from the overlay grid definition. (sec. 2.11.1)
10. *Insert pillow layers:* Insert layers of hexes to improve topology of hexes at interfaces. (fig. 2-1(h))(sec. 2.11.2)
11. *Perform smoothing:* Perform smoothing on nodes assigned to curves, surfaces and volumes to optimize mesh quality. (fig. 2-1(i))(sec. 2.12)

A description of some of the procedures and algorithms involved in the preceding steps has been outlined in [19]–[27], and for brevity are not included in this work. Instead we focus on the aspects of the work flow that are unique to computational materials modeling or that improve upon known methods in the literature.

2.2. DEFINE OVERLAY GRID

Overlay grid or *mesh-first* methods require an initial mesh that overlaps the geometry. This mesh serves as the basis for meshing procedures described in this report. The regular, structured overlay grid is often defined in terms of minimum and maximum coordinates of an axis-aligned Cartesian grid along with interval counts in each dimension. While a Cartesian grid is perhaps the simplest method for prescribing the domain, any hexahedral mesh may be used as the base overlay grid. A general hex mesh is usually generated in an external application such as Cubit [33], so that it conforms to a particular geometry. For example figure 2-2(a) shows an unstructured mesh of a cylindrical shape that was generated using a sweep scheme in Cubit. The resulting microstructure mesh based on the cylindrical overlay grid is shown in 2-2(b). Overlay grids from simple primitive geometry can also be generated inline using Pamgen[11], a parallel tool for generating simple hex meshes from a prescribed analytic description.

2.3. PROCESS INPUT DATA

The data for microstructure computational models can come from a variety of sources. Three principle sources are considered, illustrated in figure 2-3 are analytic geometry, voxel data, and volume fraction data.

Analytic geometry may be given by a series of geometric primitive specifications. For example, in figure 2-3(a), analytic definitions of concentric spheres are used to represent microballoons. Voxel data, illustrated in 2-3(b) is provided as a dense set of integers on a Cartesian grid where the integer represents the dominant material in each cell. Figure 2-3(c) illustrates the phase-field representation of a crystalline

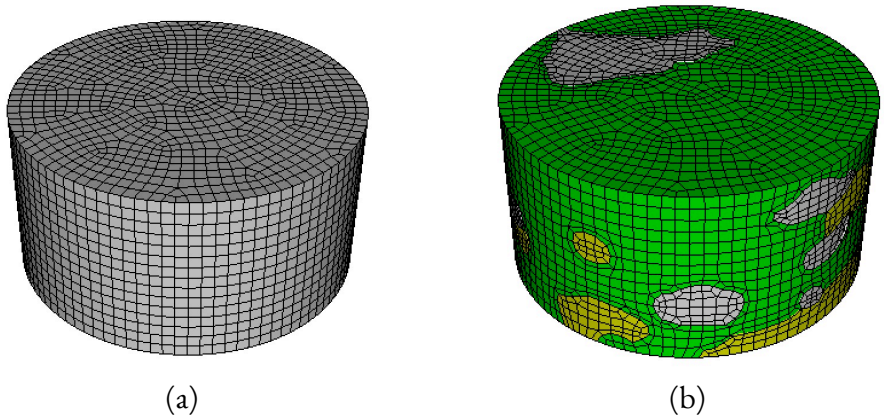


Figure 2-2. (a) Overlay grid defined by an unstructured mesh. (b) Resulting microstructure mesh generated on unstructured grid in (a)

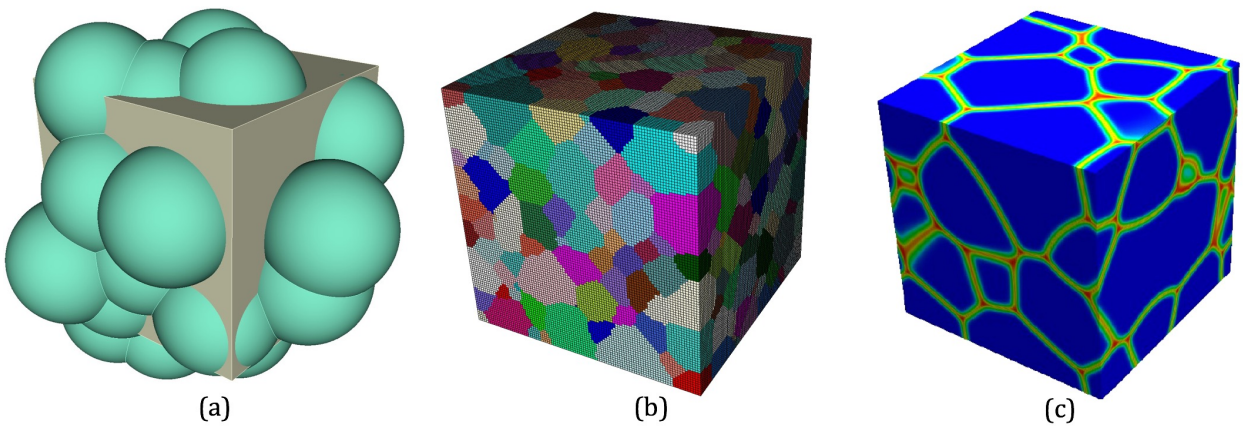


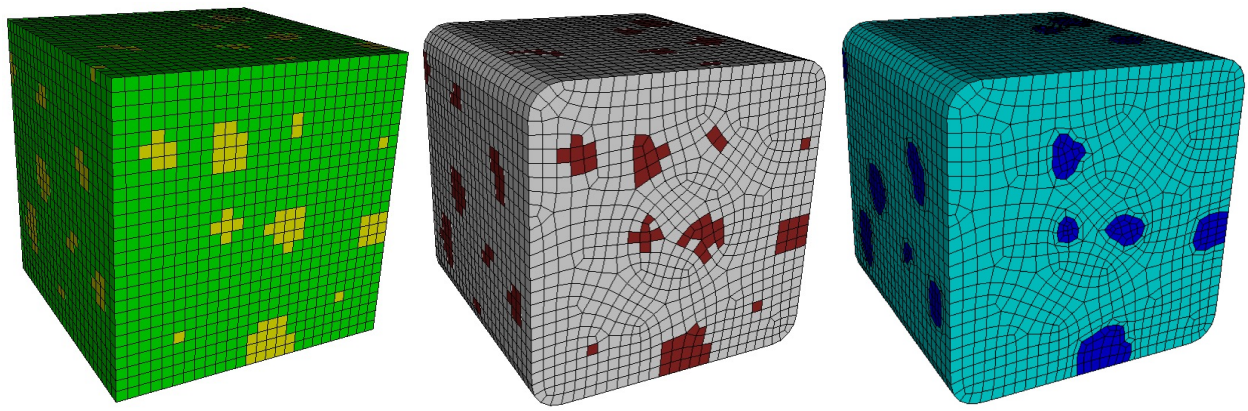
Figure 2-3. Types of input data used for microstructures mesh generation. (a) analytic geometry, (b) voxel data and (c) volume fraction data

material which is given as volume fractions on a Cartesian grid. For analytic geometry and voxel data the input must first be converted into a volume fraction representation to be processed. For analytic geometry, a series of inside-outside tests are performed on a uniform grid of sampled locations within each cell to approximate the volume fraction of each material present. For voxel data, the volume fraction is simply set to 1.0 for the cell's dominant material and 0.0 for all others.

In many cases, the geometry for the overlay grid is implied by the input data. For example, input data in the form of $N \times M \times L$ integers would result in a final mesh with resolution $N \times M \times L$ where each cell is assumed to be of unit size. Where the overlay grid is prescribed, such as that shown in figure 2-2(a), the data may be associated with each element of the unstructured mesh by including it as element data in an Exodus mesh file.

Figure 2-4 illustrates another example of input data. Figure 2-4(a) represents an ascii file with integers defining materials for each cell of a Cartesian grid. In this example, the overlay grid is defined in terms of

the unstructured mesh shown in (b). The data for this example, must be interpolated using an inverse distance weighting method from the Cartesian ascii representation to the unstructured mesh. With the data on the unstructured overlay grid, the proposed meshing procedures can be run resulting in the mesh in 2-4(c).



(a) (b) (c)

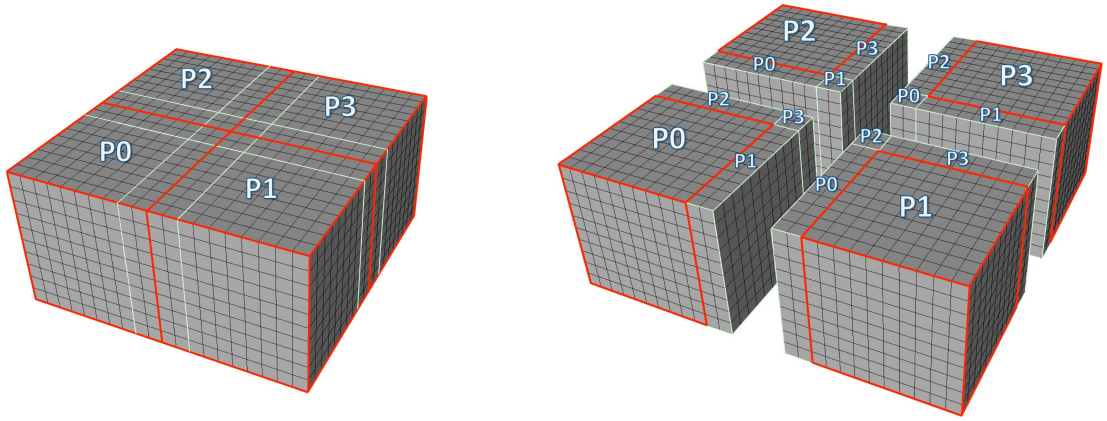
Figure 2-4. Interpolation of input data to unstructured grid. (a) Material data is provided as an ascii file with one integer per cell on a Cartesian grid (b) Data is interpolated to the hexes of an unstructured overlay grid (c) final mesh generated with Sculpt.

2.4. DISTRIBUTE DATA FOR PARALLELISM

The tools described in this report are intended to be scalable and parallel. To accomplish this, the base mesh is distributed among multiple processors with each processor generating a separate mesh for its portion of the domain. Communication is accomplished through the Message Passing Interface (MPI) to maintain parallel consistent, conforming hex meshes.

For Cartesian grids, the distribution is accomplished via a trivial decomposition as shown in figure 2-5. 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, as illustrated in figure 2-6. The location of grid nodes and size of cells of Ω_M^p is established by defining three independent arrays in each coordinate direction:
$X_\Omega = \{x_0, x_1, x_2, \cdots, x_{nx+1}\}, Y_\Omega = \{y_0, y_1, y_2, \cdots, y_{ny+1}\}, Z_\Omega = \{z_0, z_1, z_2, \cdots, z_{nz+1}\}$,
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.

In the proposed primal contouring approach, the nodes of the grid M^0, become the actual nodes in the final FEM hexahedral mesh. This has the consequence of the need to establish *dual nodes, edges, faces* and *cells* (shown in figure 2-6) when defining our algorithms and procedures. Rather than explicitly defining these dual entities we can define a correspondence between primal and dual entities and utilize only the primal definition for implementation purposes.



(a) (b)

Figure 2-5. (a) Cartesian grid with a 4 processor decomposition. (b) Exploded view of processor decomposition showing ghosted cells on each processor.

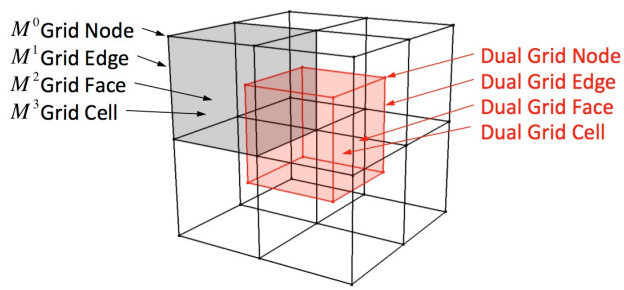


Figure 2-6. The primal grid entities are illustrated and identified by $M^r|r = 0, 1, 2, 3$. Corresponding entities in the dual are also illustrated.

Ghosted cells are set up at the boundaries of processors where they meet a neighboring processor. For our purposes we utilize 2 layers of ghost cells which duplicate geometry and data from neighboring processors. Figure 2-7 illustrates ghosting on a domain composed of two processors. The cells in region A on processor 0, shown in (a), are duplicated and added to the cells at the boundary of processor 1 shown in (b). In a similar manner, the cells in region B on processor 1 are duplicated and added to processor 0.

2.4.1. Unstructured Decomposition

For unstructured meshes, the mesh must first be decomposed into roughly equivalent regions. To accomplish this we use the `decomp` application which is part of the SEACAS [35] tool suite. The `decomp` tool incorporates Sandia's Zoltan [5] software for distribution and load balancing. To ensure conforming hexes, ghost cells are extracted and used as a mechanism for communication.

Table 2-1. Correspondence between primal and dual entities

M^i	Primal	Dual
M^0	node	cell
M^1	edge	face
M^2	face	edge
M^3	cell	node

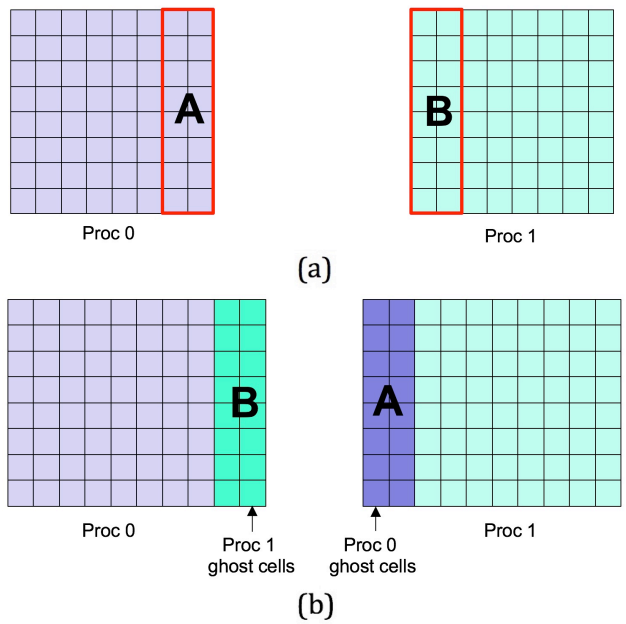


Figure 2-7. (a) Domain defined on two separate processors. Cells in regions A and B are owned by processors 0 and 1 respectively. (b) Ghost cells in regions A and B are copied and added to to processors 0 and 1 respectively.

2.4.2. Periodic Ghosting

Periodic meshes can be generated by ensuring that nodes and faces on opposite sides of the RVE match exactly. This can be accomplished by using standard ghosting procedures with MPI. Without periodicity, when multiple processors are used, cells are duplicated only on neighboring processors that share common nodes and faces. Periodic meshes also share common nodes and faces on opposite sides of the RVE domain. Periodic ghosting can be set up by copying cells from one side of the mesh domain to the opposite side, applying a translation of one period to the nodes. As an example, Figure 2-8(a) shows the same two processor domain shown in figure 2-7. For the periodic case, we define regions of cells on all boundaries and corners of the domain that will be duplicated to opposite sides. We note that ghosting between neighboring processors in regions A and B are handled the same as the non-periodic case. For all other boundaries, cells are duplicated to opposite sides which may be on the same processor, as in regions E and D or another processor, as in regions C and F

We note that periodicity defined in this manner assumes that the underlying geometry is also periodic. While this approach will work, even if the geometry is not periodic, true periodicity requires that the geometry is periodic.

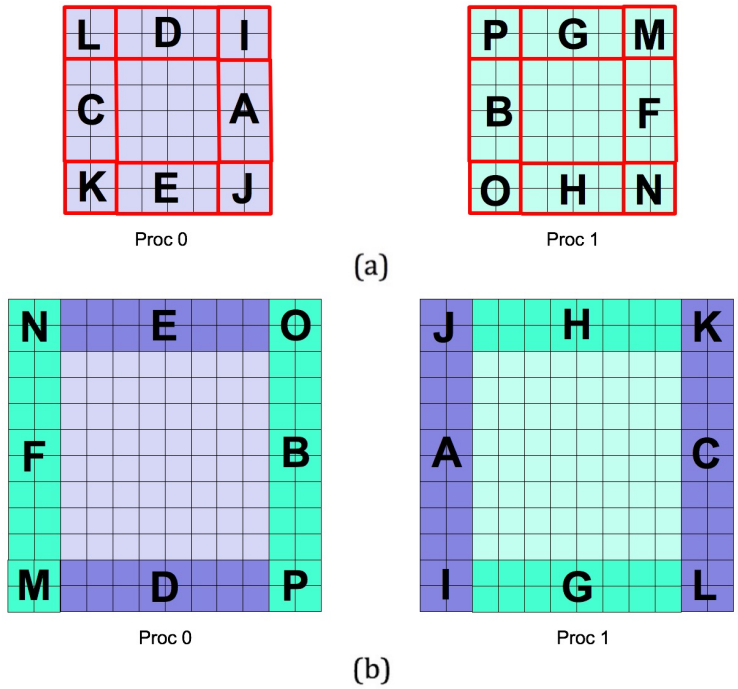


Figure 2-8. Periodic ghosting: (a) two processor example showing cells ghosted on the same or neighboring processor. (b) Cells added to boundary of each processor that are copied from either the opposite side of the same processor or from the neighboring processor.

2.5. MESH ADAPTIVITY

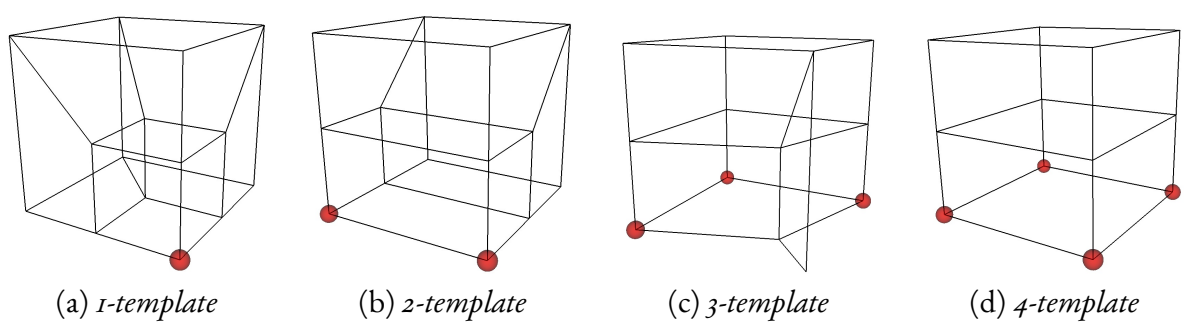
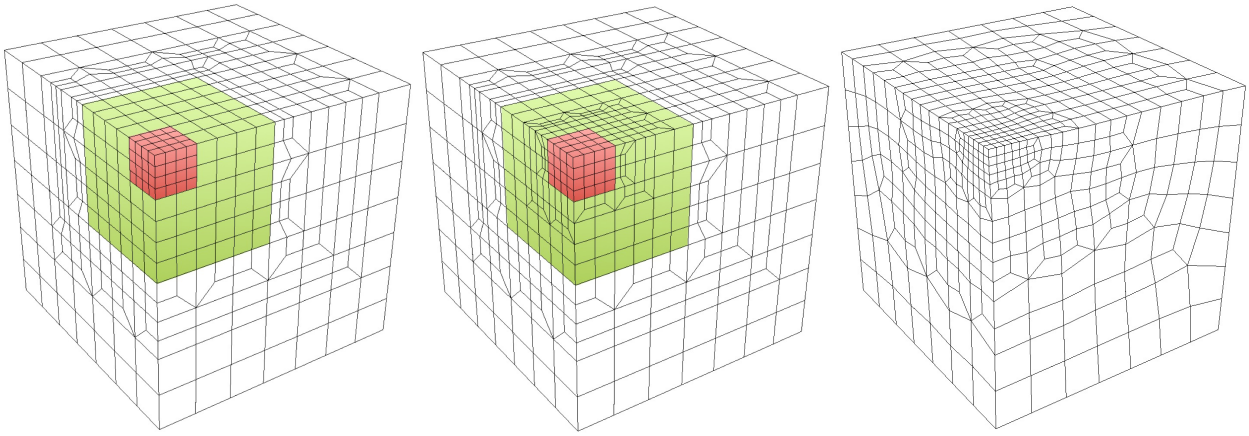


Figure 2-9. Complete set of 3D 2-refinement templates. Templates are identified based on the number of marked nodes on a hex.

2.5.1. Refinement

The input overlay grid provides the initial base resolution for the mesh. This initial resolution can be modified using the 2-refinement method described in [24, 25], which uses a template-based approach for subdividing cells of the overlay grid. Figure 2-9 illustrates the templates used for subdivision. Since we require a conformal mesh where no hanging nodes are present, the templates provide a solution for transitioning from the fine resolution to coarse. Reference [25] describes methods for marking nodes based on geometric criteria. One of the 4 templates shown in figure 2-9 are then applied successively in the three Cartesian directions based upon the marked nodes.



(a) (b) (c)

Figure 2-10. (a) Example 3D refinement showing first level of refinement from green elements. Green elements will serve as transition zone for second level of refinement from uniformly refined red elements. (b) Transition elements for second level of refinement added. (c) Smoothing applied.

Refinement can also be applied successively, to subdivide cells multiple times. Figure 2-10 illustrates two levels of refinement. In this example, the green cells in (a) show the first level of refinement with the transition templates applied surrounding them within the white cells. In (b) the red cells illustrate a second level of uniform refinement. These elements also require transition templates which are placed within the green cells. (c) shows the same mesh after it has been smoothed.

We note that the 2-refinement technique outlined in [24, 25] is limited to a structured arrangement of cells in the overlay grid such as a Cartesian grid or mapped mesh. This method relies on pairing of adjacent layers of cells for the templates shown in figure 2-9. As a result, general unstructured overlay grids will not admit a 2-refinement procedure. We note however that 3-refinement procedures may be applied using an external tool [33] prior to import.

2.5.2. Coarsening

For some cases, the volume fraction data provided on the overlay grid can be very high resolution. There is currently no general method for coarsening a hexahedral mesh. Instead, when using a structured

overlay grid, we can first redefine the grid in terms of a coarse resolution and refine cells only where needed.

For coarsening from dense Cartesian data, we begin with a grid resolution $N \times M \times L$, where the intervals in each coordinate direction can be defined as $N = 2^{c_{max}} N_0$, and where N_0 is the coarsest allowable cell dimension and c_{max} is the user-defined maximum number of coarsening levels permitted. Based on this relationship, the coarsest dimensions $N_0 \times M_0 \times L_0$ are determined. This serves as the initial grid resolution, where one cell of the base grid contains $2^{3c_{max}}$ cells of the original dense grid. To ensure all cells in the dense grid are used, the dimensions N, M and L should be evenly divisible by $2^{c_{max}}$. If not, then the remaining cells at the boundary of the dense grid will be discarded. Figure ?? illustrates this procedure on a 2D grid with two levels of coarsening ($c_{max} = 2$).

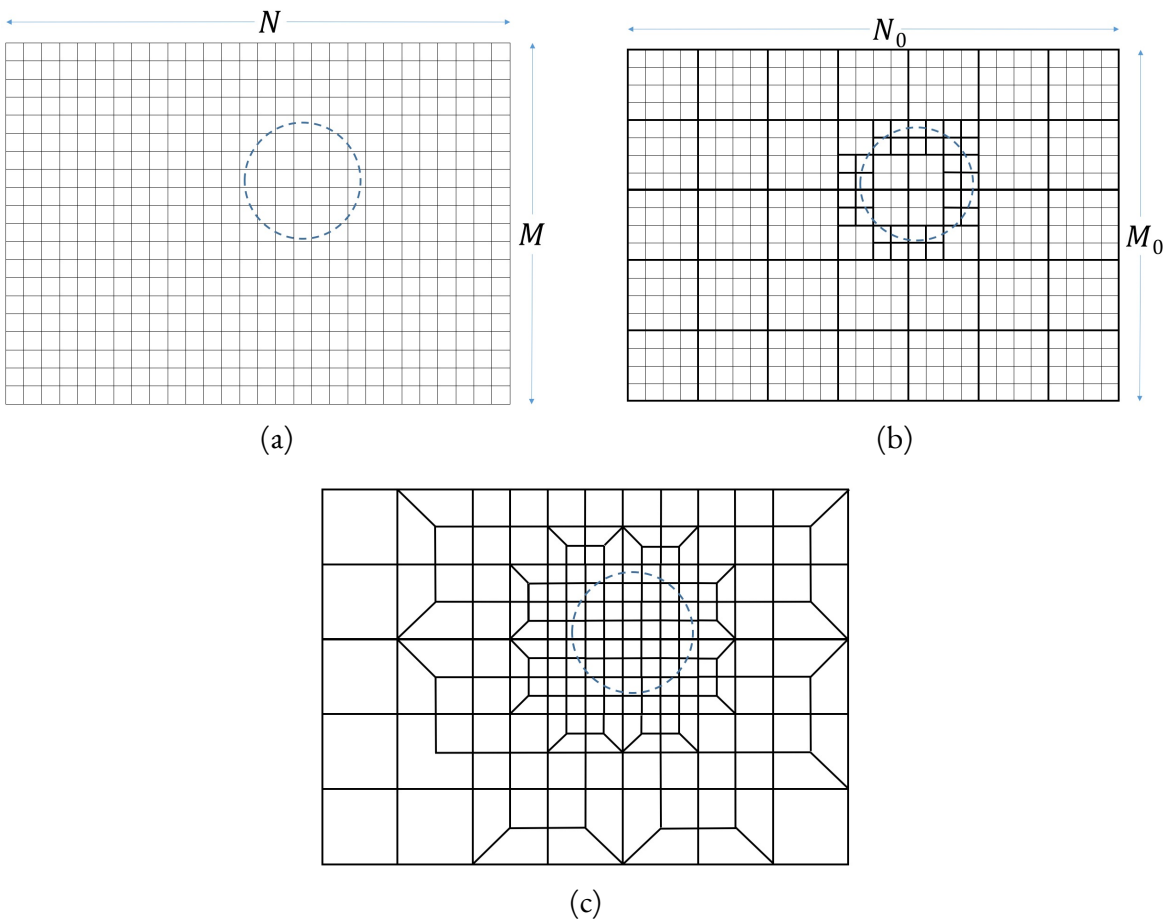


Figure 2-11. (a) Initial dense resolution Cartesian grid, (b) Cells of resolution $2^{c_{max}}$ consolidated where data does not change. (c) Transitions inserted to maintain conformal mesh

A cell in the coarse base grid is marked for uniform refinement if the children contained within the parent cell contain volume fractions that differ by less than a user-defined threshold δ. This procedure continues, adaptively applying uniform refinement to child cells up to c_{max} times.

An example of coarsening is shown in figure 2-12. In this case, the initial grid resolution was $456 \times 456 \times 166$ or approximately $34.5 million$ cells. In this example, a high resolution scanned

representation of apparatus and microstructure resulted in a resolution that would be too dense for practical use. With a coarsening value $c_{max} = 3$ the initial coarsened resolution was reduced to $N_0 \times M_0 \times L_0 = 57 \times 57 \times 20$ or $64,980$ cells. Once refinement is applied to the coarse grid, the maximum resolution for the microstructure is recovered, but leaves most of the solid apparatus structure at the coarser resolution.

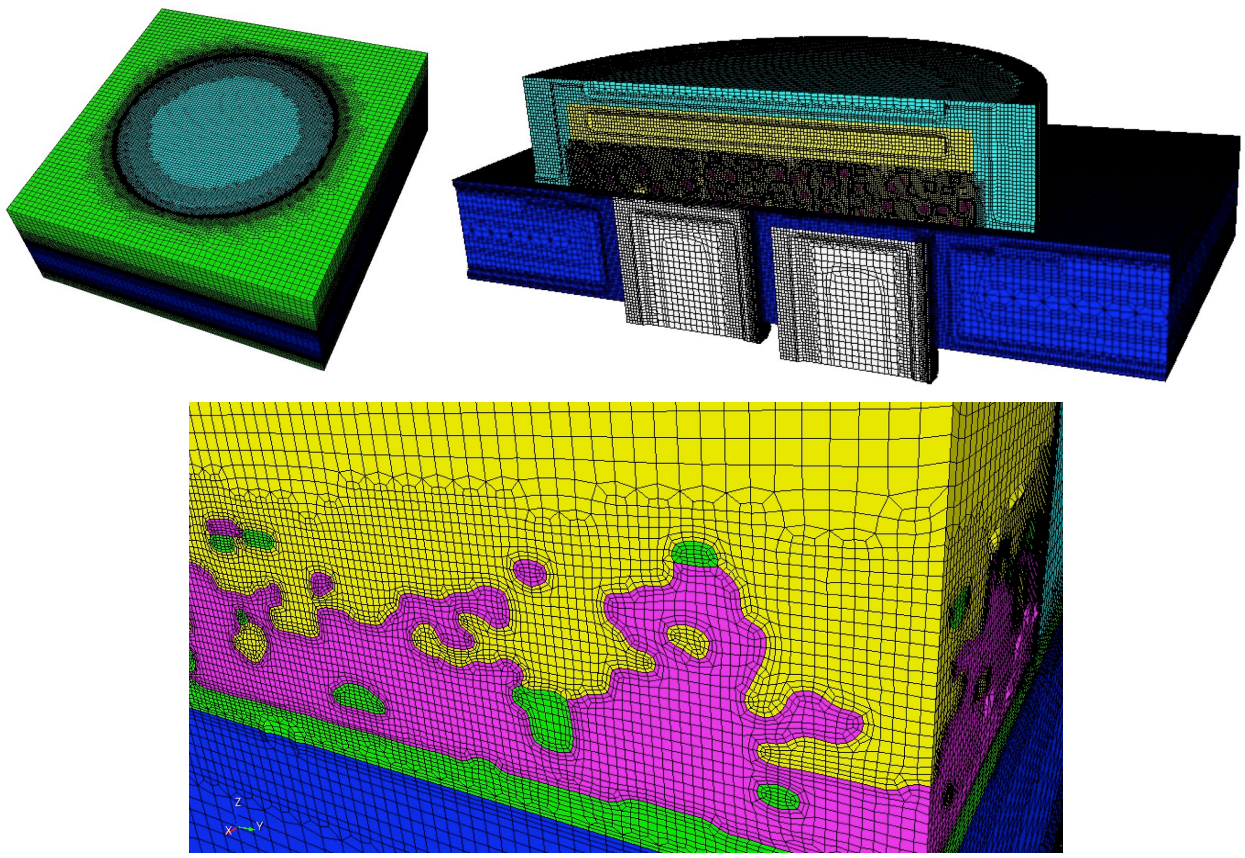


Figure 2-12. Example of mesh coarsening. Initial data resolution is approximately 34.5 million cells. With 3 coarsening levels ($c_{max} = 3$), and refinement, final hex mesh is approximately 4.6 million elements

2.6. GRADIENT ESTIMATION

As initial data is provided as pure cells or volume fractions, we must convert interfaces between materials into a smooth representation. The key step of this procedure is the approximation of the normals at material interfaces. This can be done by using a finite difference approach to first approximate the gradients of the materials throughout the model. We begin by approximating the volume fraction gradient at cell centers for each material defined on the domain. For each cell, exactly N_{mat} scalar values $v_n = v_0, v_1, ...v_{N_{mat}-1}$ are provided. We can represent the gradient for material n at cell center (i, j, k) as $\bigtriangledown v_n(i, j, k) = \left(\frac{\partial v_n}{\partial x}, \frac{\partial v_n}{\partial y}, \frac{\partial v_n}{\partial z}\right)$. For each cell $M^3_{i,j,k}$ in the grid, the differences of

26 neighboring values Δv_n, and cell center coordinate locations $(\Delta x, \Delta y, \Delta z)$ at $M^3_{i\pm,j\pm,k\pm}$ 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 (2.1) produces the least squares approximation to the gradient for material n.

$$\begin{bmatrix} \sum \Delta x_i^2 & \sum \Delta x_i \Delta y_i & \sum \Delta x_i \Delta z_i \\ \sum \Delta x_i \Delta y_i & \sum \Delta y_i^2 & \sum \Delta y_i \Delta z_i \\ \sum \Delta x_i \Delta z_i & \sum \Delta y_i \Delta z_i & \sum \Delta z_i^2 \end{bmatrix} \begin{Bmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial z} \end{Bmatrix} = \begin{bmatrix} \sum \Delta x_i \Delta(v_n)_i \\ \sum \Delta y_i \Delta(v_n)_i \\ \sum \Delta z_i \Delta(v_n)_i \end{bmatrix} \tag{2.1}$$

Note that the finite difference calculation $\Delta(v_n)_i$ in equation 2.1 for any cell center, requires the difference of volume fractions between neighboring cells. As a consequence, the required primal node offset for any node M^0, will require volume fraction data from two layers of neighboring cells to contribute to its final location. This fact motivates the need to establish two layers of ghost cells at processor boundaries, effectively reducing the amount of required interprocessor communication.

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 $|\Delta v_n| > 0$, we can ignore cases where the gradient is undefined. In practice we compute gradients only where $|\Delta v_n| > 1.0e - 6$.

To compute gradients, we simply loop through all cells of the grid computing the required $\bigtriangledown v_n(i, j, k)$ for every cell. Note also that for cells at the boundary of the grid, fewer than 26 adjacent cells are available for the summations in equation (2.1). 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. Since we have established two layers of ghost cells, these effects are normally minimal, but can effect the final node positions. 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.

2.7. DATA FILTERING

As an initial approximation of the geometry, each cell is assigned a material ID based on its dominant material. This initial approximation can include conditions that can make hex meshing impossible or result in less than acceptable quality. Here we identify five specific classes of problems and propose solutions for filtering the data by modifying the initial material assignment for a small number of cells. These solutions include:

1. *Non-manifold resolution:* Identify and resolve cases where a common material is connected by a single node or edge.
2. *Defeaturing:* Identify and remove small material volumes, protrusions and isthmus conditions.
3. *Thickening:* Add volume fraction to material boundaries to reduce noise in input data.
4. *Reversal correction:* Identify and resolve case at nodes where adjacent faces have opposite normals.
5. *Expansion layer insertion:* Add additional layer(s) to RVE boundaries to improve quality.

The following sections provide additional details for each of these data filtering solutions.

2.7.1. Non-manifold resolution

Raw data describing material arrangement within an RVE can often result in local non-manifold conditions. Figure 2-13 illustrates two simple cases where a non-manifold condition exists in an overlay grid. In this case a cell containing blue material is connected to an adjacent cell with blue material by only an edge (a) or a node (b). The proposed mesh generation method requires that all materials assigned to cells in the overlay grid maintain a manifold condition. In other words, all adjacent cells with the same material must have at least one face in common. To achieve manifold conditions, it is usually necessary to re-assign the material at nodes and edges exhibiting a non-manifold condition.

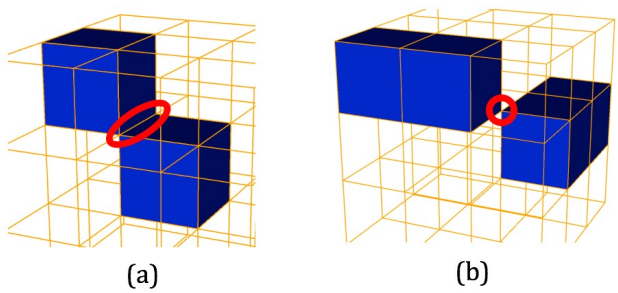


Figure 2-13. (a) Non manifold condition at an edge. (b) Non-manifold condition at a node.

In [26, 27] we describe a technique for eliminating non-manifold conditions within a structured grid. Where all interior nodes in the base grid have a valence of eight, a limited set of seven non-manifold cases, illustrated in figure 2-14 have been identified. For each non-manifold case, a corresponding set of resolution options, where cells are either added or removed from the non-manifold set are identified. The option resulting in the smallest change to the target volume fraction is selected.

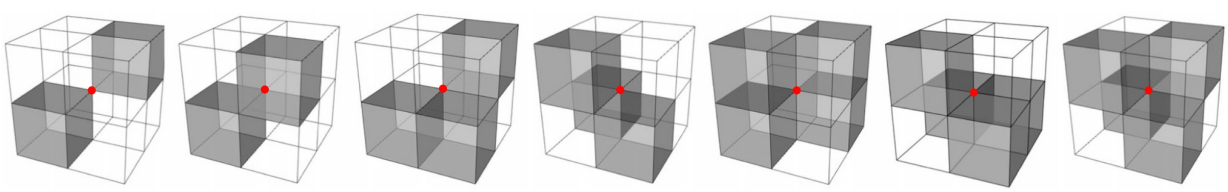


Figure 2-14. Seven cases for non-manifold conditions at a node are illustrated.

With a relatively small set of non-manifold conditions possible for structured grids, it is a manageable task to implement a set of hard-coded resolution strategies. Following conformal refinement of the structured Cartesian grid, the valence at any node in the grid may be variable. Figure 2-15 illustrates a few examples of non-manifold conditions that can exist in an unstructured grid. To identify non-manifold conditions for the general case, we utilize the Euler number E, given by equation 2.2, a standard measure for characterizing the completeness of a mesh based on a count of nodes, edges and faces.

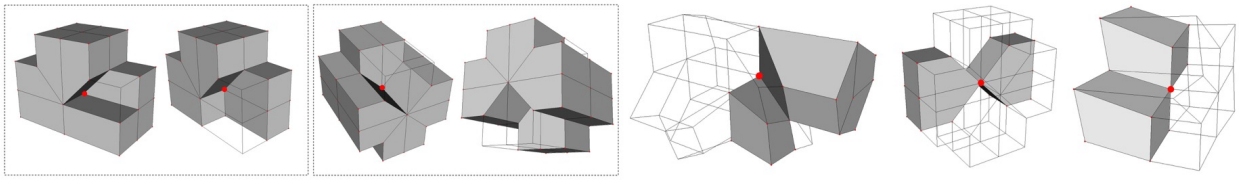


Figure 2-15. Examples of unstructured non-manifold cases at a node

$$E = N_{nodes} - N_{edges} + N_{faces} \tag{2.2}$$

where N_{nodes}, N_{edges} and N_{faces} are the boundary nodes, edges and faces respectively of a three-dimensional mesh. For a hexahedral mesh, to be free of non-manifold conditions, the Euler number must be exactly equal to 2 for every material present at a node. The Euler number is computed for sets of cells at a node i that share a common material j. Non-manifold conditions at node i are then identified where $E(i)_j \neq 2$ for any material j.

To resolve non-manifold connections, we identify a series of potential options that will resolve the non-manifold condition by compiling a list of options where material IDs are reassigned at node i to ensure $E(i)_j = 2$. With multiple options identified, we choose the option that will result in the minimum deviation from the underlying volume fraction data.

The Euler number computed at a node for each material present, will be a valid measure of manifoldness whether structured or unstructured. In practice however, since the structured case is usually most common, for efficiency we distinguish the structured case and utilize the hard-coded resolution strategies outlined in [26, 27]. For the unstructured cases, we compute the Euler number and a series of modifications that can be tested to ensure $E(i)_j = 2$ is met for each material at a node.

We also observe that a solution that will resolve the non-manifold condition at one node, may otherwise introduce a new non-manifold condition at a neighboring node. Similarly, neighboring nodes, both with non-manifold conditions, may select conflicting solutions. To avoid these cases we utilize a coloring approach that effectively isolates the kernel of cells that will be involved in a single non-manifold resolution operation. This is accomplished by first identifying all possible non-manifold resolution cases along with their kernel of surrounding cells and their corresponding resolution strategy to minimize deviation from the underlying volume fractions. For cells that would otherwise be involved in more than a single non-manifold resolution operation, the operation with the minimum volume fraction deviation is selected and the others discarded for the current iteration. Once a unique set of independent kernels have been identified, the actual modifications to the cell material assignments are made. In a subsequent iteration, the non-manifold nodes that were discarded are again identified and included in the coloring selection procedure.

Although the proposed coloring procedure succeeds in isolating individual non-manifold resolution operations, it is possible that some amount of oscillation may occur where the same cells may be added or removed multiple times. To avoid this condition we introduce a small delta (δ) volume fraction value to cells that have been changed in the current iteration. For example, if the volume fraction of material A in a cell is x, the new volume fraction for material A in the cell will either be $x + \delta$ or $x - \delta$ depending on whether the cell was added or removed from material A. The δ value is progressively

increased for each iteration for which the cell's material assignment is changed. The result is that the oscillation will be inhibited resulting in most models converging within four or five iterations.

We note that the proposed coloring procedure is effective in providing a reproducible and parallel consistent result that minimizes deviation from the underlying volume fraction data. To ensure parallel consistency, communication of the proposed resolution kernels must be effected for cells at processor boundaries. This is to ensure that cases that involve material regions spanning neighboring processors will be identified and selected in a consistent manner.

2.7.2. Defeaturing

The resolution of all non-manifold conditions is a prerequisite for the proposed primal contouring hex mesh generation method as subsequent geometry and smoothing operations will otherwise fail. We have noted however that computable quality hex elements can be difficult or impossible to produce based on some cases of material cell assignment. The cases described in figure 2-16 illustrate conditions that can otherwise result in poor quality along with their proposed resolutions.

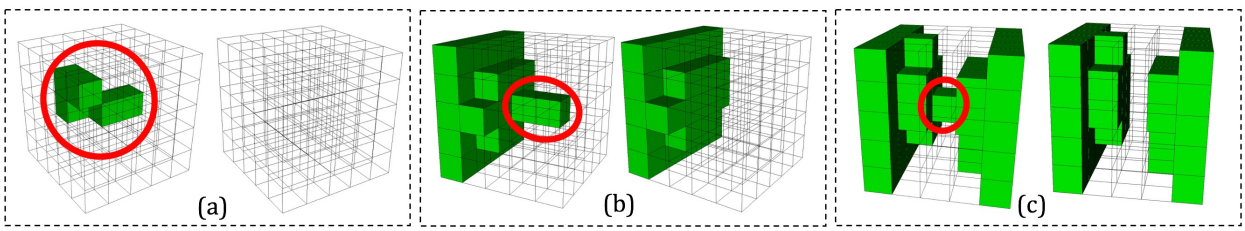
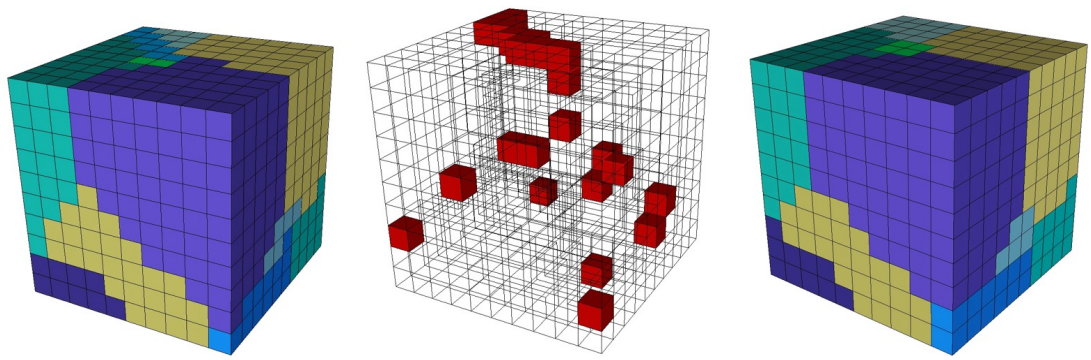


Figure 2-16. Three cases of cell material assignment that can result in poor quality elements and their proposed resolutions: (a) Island: small number of cells grouped together, (b) Protrusion: one or more cells surrounded by another material on at least 4 sides, (c) Isthmus: single cell connecting two larger groupings of cells of the same material.

For small volumes, shown in figure 2-16(a), a user defined minimum number of cells per volume, S_{min} is used. Small volumes are identified by collecting groupings of the same material through a recursive search of surrounding cells. Parallel communication is also required where volumes with less than S_{min} cells are at the boundary of a processor domain.

Peninsula and isthmus conditions can be identified by enumerating the cells of a different material at the six faces of a cell. Cells with at least 4 faces with adjacent cells of a different material are selected for modification. Update of the material assignment for defeaturing criteria is effected in the same manner as used for non-manifold resolution. A coloring procedure, where each cell identified as a potential change is included in a kernel defined by its immediate neighbors. Defeaturing options are discarded from the current iteration when its kernel cells are included in more than one option. In these cases the option with the minimum change in volume fraction will be selected and the others discarded for the current iteration.



(a) (b) (c)

Figure 2-17. Example of defeaturing operations on a microstructure model. (a) Initial microstructure with colors representing different grains. (b) Red cells indicate those that satisfy one of the criteria shown in figure 2-16. (c) Resulting microstructure after reassignment of materials.

Figure 2-17(a) illustrates a simple microstructure prior to defeaturing. The red cells in 2-17(b) indicate the cells that have been identified for defeaturing that meet one of the criteria shown in figure 2-16. The microstructure following defeaturing is shown in figure 2-17(c).

We note that it is possible that criteria used to resolve defeaturing conditions may introduce non-manifold conditions. Likewise non-manifold resolution may introduce conditions that will need to be defeatured. To ensure all cases are effectively resolved, the defeaturing and non-manifold resolution are done in a loop and continue until no further changes are necessary.

2.7.3. Resolving Reversals

An additional case that presents a problem for some forms of smoothing is illustrated in figure 2-18(a). In order to insert a buffer layer at the interface between materials, an approximate normal is computed for the surface. A normal vector at an interface node can be computed by averaging the face normals of surrounding interface faces. Averaging face normals for the configuration of cells illustrated in figure 2-18 (a) can produce negative quality buffer layer hexes.

To avoid the reversal case, we can re-assign materials at the node in a similar manner we correct for non-manifold conditions. Figures 2-18 (b)-(e) illustrate the options used for the structured case for modifying the reversal case in (a). Similar to the non-manifold resolution criteria, the option selected attempts to minimize the change in local volume fraction for cells attached to the node.

For the unstructured case, we can identify a reversal by comparing local face normals at a node using the following criteria:

$$\mathbf{N_i} \cdot \mathbf{N_j} < -(1 - \delta) \tag{2.3}$$

where $\mathbf{N_i}$ and $\mathbf{N_j}$ are a pair of face normals at the same node. Equation 2.3 identifies faces that have normals facing in nearly opposite directions where δ defines a small threshold. When a reversal is encountered, local changes to materials are identified and updated in a similar manner to the unstructured non-fold resolution case described in section 2.7.1.

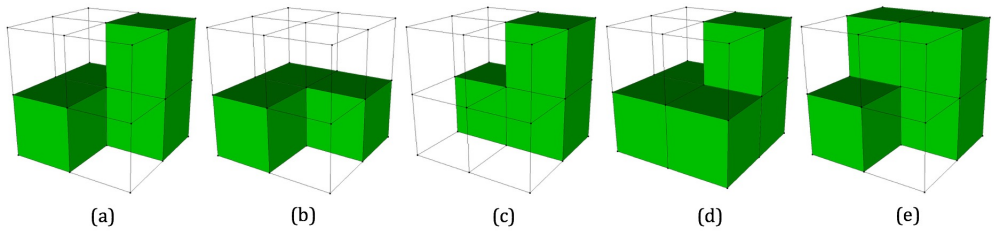


Figure 2-18. (a) Example reversal case where adjacent faces have opposite facing normals. (b)-(e) shows possible resolution states for reversal.

2.7.4. Thickening

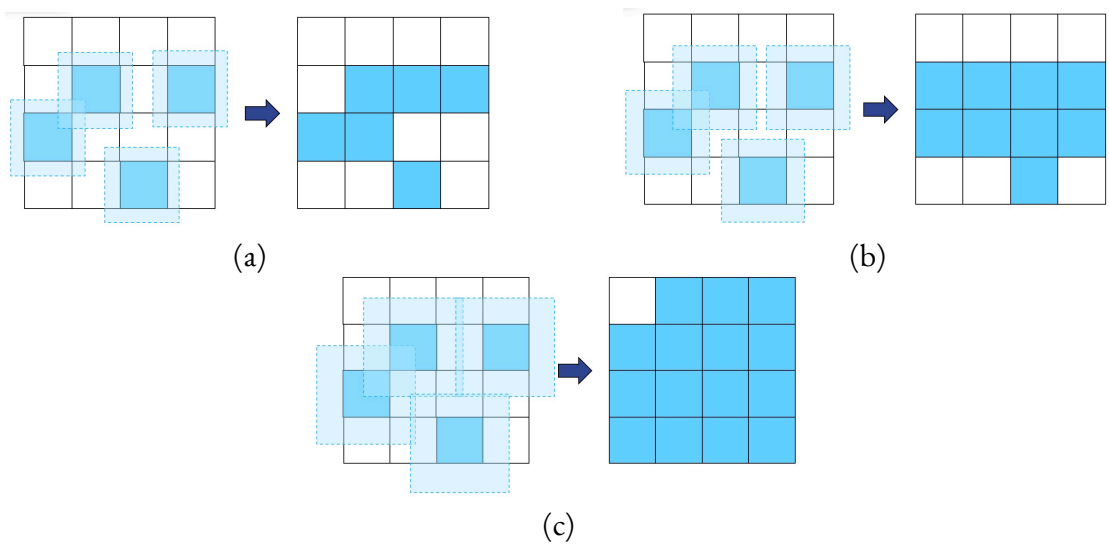


Figure 2-19. Effect of applying thickening. Light blue illustrates the relative amount of thickening applied to each cell. The resulting material assignment is shown at right for each case. Larger values of thickening (c) result in more neighboring cells reassigned and a more continuous material definition

For particularly noisy data, the thickening process has the effect of filling in gaps to create more continuity in the material interfaces. While the defeaturing procedure tends to remove cells from material, the thickening process will add volume fraction at material boundaries.

To thicken a material, a target volume fraction value, vf_i is specified that will be added at all boundaries of material $\mathbf{M_i}$. Figure 2-19 illustrates how thickening is applied at the cell level and its effect. The

relative sizes of the light blue boxes indicate the relative size of vf_i and the figures on the right indicate the change in material assignment as a result. Each neighboring cell to those with material assignment $\mathbf{M_i}$ adds some portion of vf_i to its volume fraction data. Since we must maintain $\sum_{j=0}^{k} vf(i,j) = 1$ for the sum of volume fractions in the cells, other materials present in the cell must also be reduced by the same amount. Cells may then reassigned to a new material based upon the dominant material present in the cell after addition of vf_i.

Figure 2-20 illustrates the effect of thickening over a range of thickening values combined with defeaturing on an example 3D data set. Normally thickening is applied before defeaturing. Images on the top row show the effect of changing vf_i ranging from $vf_i = 0$ (no thickening) to $vf_i = 1$ (maximum thickening). The second row of images illustrate the effect of defeaturing the thickened cells and the bottom row shows the final mesh.

Thickening can also be applied to multiple materials in a user specified order. This can result in competing thickening operations where materials are adjacent by removing or adding material to the same cells. Some trial and error may be necessary to determine which materials and values for vf_i should be used.

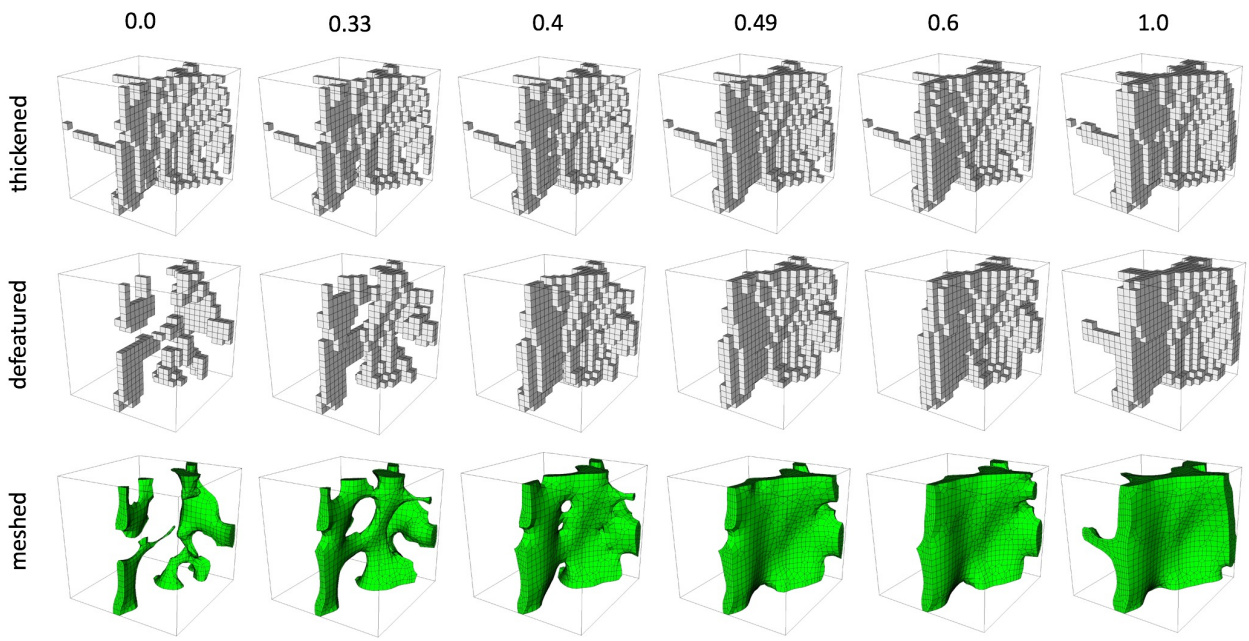
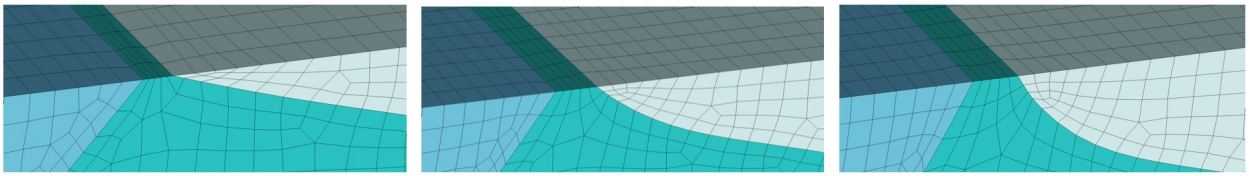


Figure 2-20. Effect of thickening on an example 3D data set. Top row shows the effect of different values of thickening applied to the cells of one material. Middle row illustrates the effect of defeaturing on the thickened cells. Bottom row shows the resulting hex mesh based on the thickened cells.

2.7.5. Expansion Layers

When observing meshes that have been generated with the proposed method, we often observe cases where material interfaces are almost tangent or form very small angles with the RVE boundary. An

example of this condition is illustrated in figure 2-21(a) where the quality of the hexes is constrained at the boundary.



(a) (b) (c)

Figure 2-21. Example of introducing expansion layers to control element quality. (a) Resulting hex mesh where material interface intersects RVE boundary at small angle without expansion layers (b) Single expansion layer has been inserted. (c) Two expansion layers inserted.

To resolve this, we can optionally introduce one or more expansion layers to the boundary of the RVE. Each expansion layer adds an additional two cells to the intervals of the RVE in each Cartesian direction. The new cells in the expansion layers are assigned the volume fractions from their immediate neighbors in the grid. For cells at the corners and vertices of the RVE, the volume fraction data is repeated based on its closest cell in the base grid.

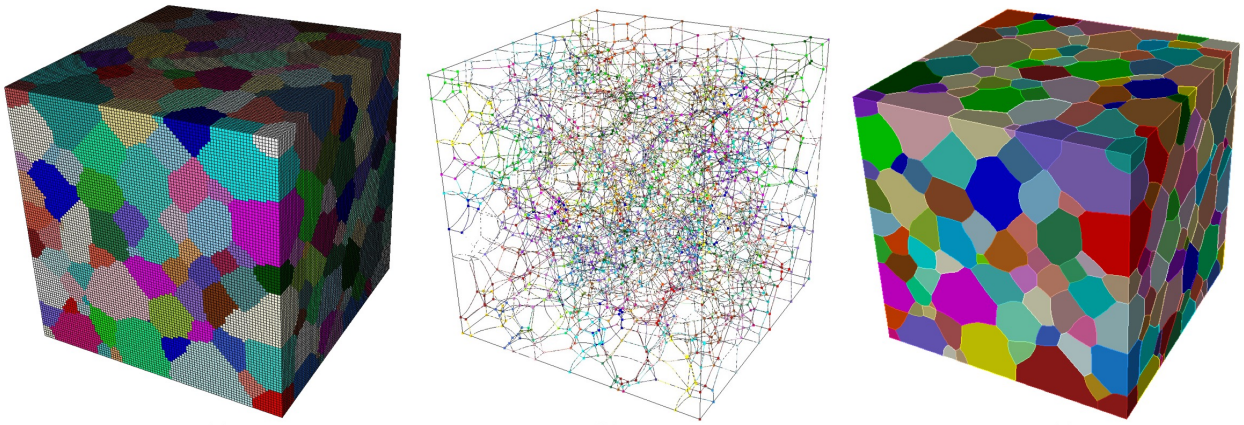
By adding the expansion layers, material interfaces where acute intersections occur at the RVE boundaries will effectively be modified to exit the RVE at close to an orthogonal angle, after smoothing has been completed. The effect of inserting one and two expansion layers are illustrated in figure 2-21(b) and (c) respectively with the orthogonality increasing with the number of inserted layers.

2.7.6. Data Filtering Accuracy

We note that when using the data filtering tools as described above, significant changes to the underlying data may be necessary in order to achieve acceptable results. A tool that compares input data with the final mesh is provided to check for accuracy. The volume of each material present upon input and the volume of elements in each material in the final mesh are computed and compared. A relative error measure can then be determined. In many cases, modifying the thickening values for different materials can greatly effect the error measures.

2.8. BREP CONSTRUCTION

With the cell material assignment completed, it is now possible to construct a BREP topology graph of the model. A BREP is necessary to help facilitate subsequent pillowing and smoothing operations. The BREP includes a set of connected volumes, surfaces, curves and vertices which each serve as containers for mesh entities. BREP construction is done by selectively grouping and assigning cells, faces, edges and nodes of the grid to an owner volume, surface, curve and vertex respectively.



(a) (b) (c)

Figure 2-22. Example BREP extracted from a 100 x 100 x 100 cell microstructure voxel model. (a) shows the initial Cartesian grid with cells colored according to their initial material assignment, (b) 6395 curves, 3086 vertices, (c) 3946 surfaces and 636 volumes extracted from the voxel data.

One volume is created for each contiguous set of cells assigned a common material ID. Surfaces can then be identified and assigned faces that have common pairs of adjacent cells that have been assigned to different volumes. Curves are then identified and assigned edges that have a common set of unique surfaces. Finally, vertices are created where adjacent edges have been assigned to different curves.

Topology containers must also be distinguished by their location with respect to one of the following classifications: 1. interior of the RVE, 2. the RVE domain boundary or 3. processor boundary. Distinct pillowing and smoothing operations are performed based on its classification with respect to one of these three types.

An example BREP is illustrated in figure 2-22 for a microstructure with 636 grains . In 2-22(a), the initial 100 x 100 x 100 Cartesian grid with each cell colored by its material assignment is shown. 2-22(b) the curves and vertices, 2-22(b) surfaces and volumes extracted from the voxel data are illustrated.

2.9. PRIMAL CONTOURING

The primal contouring procedure is a method for approximating the interfaces between materials. Reference [26, 27] describes a procedure for computing a minimizer location for cell nodes between two materials. This method is similar to the dual contouring method described in [38], however instead of forming elements from the dual of the base grid, leaving the underlying grid unchanged, we instead modify the base grid itself, adjusting locations of the primal nodes of the cells of the grid. This approach better lends itself to parallel distribution, where cells can be distributed onto different processors without having to define elements across processor boundaries.

As outlined in 2.8, a B-Rep is constructed based on the topology of the material assignments in each of the cells. With this construction the set of curves and surfaces are assigned their appropriate groupings

of edges and faces. The primal contouring procedure consists of moving the nodes on these entities to define a smooth interface approximation that best maintains the underlying volume fraction data. Here we propose an improved and generalized procedure over that described in [26, 27], for assigning locations for nodes on curves and surfaces built from an unstructured base grid.

2.9.1. Locating nodes on surfaces

Figures 2-23(a) and (b) illustrate the basic approach for locating a node p_{ab} that lies on a surface between materials a (yellow) and b (blue). We assume that volume fractions for materials a and b are provided for each adjacent cell, and their approximated normals. In [26, 27] we describe a method for computing normals for materials in each cell based upon a finite difference approximation between volume fractions and a least squares fit from the 26 neighbors of an interior cell. This method can also be generalized for unstructured grids where any number of neighboring cells can be used to approximate normals for a given cell. With this information we approximate a set of planes that intersect the edges of a polygon that is defined by the centroids, C_i, of the neighboring cells to p_{ab}. In figure 2-23(a) we show two planes intersecting the virtual edges $C_1 - C_2$ and $C_3 - C_4$ respectively. The intersecting planes are defined by locations $x(ab)_{12}$ and $x(ab)_{34}$ and normals which have been interpolated from volume fractions at the cells. Node p_{ab} can then be projected to these planes to produce points p_{12} and p_{34}. Finally, as illustrated in 2-23(b), node location p'_{ab} is computed as the weighted average of locations p_{12} and p_{34}. For the general unstructured case, any number of locations p_{ij} can contribute to locating p_{ab}. The set of projected points p_{ij} will be computed for all dual edges corresponding to adjacent cell faces to p_{ab} with different materials on either side.

2.9.2. Improved edge-cross locations

Of critical importance to the interface approximation is accurately approximating the edge-cross locations. For the example in figure 2-23(a) the edge-cross locations corresponds to points $x(ab)_{12}$ and $x(ab)_{34}$. Figure 2-24(a) illustrates the case where two adjacent cells have been assigned to different materials a and b. In this example Cell C_1 contains volume fractions $v(a_1) = 0.6$ and $v(b_1) = 0.4$. Cell C_2 contains volume fractions $v(a_2) = 0.3$ and $v(b_2) = 0.7$. Our objective in this example is to define a parametric location t_{12} on a line segment connecting the centroids of C_1 and C_2 where we expect the interface of materials a and b to cross the line segment.

We observe that linear curves with slope $v(a_2) - v(a_1)$ and $v(b_2) - v(b_1)$ can be defined describing the approximate change in volume fractions between C_1 and C_2 for materials a and b. For this example, the curves are illustrated in figure 2-24(b). We then compute the intersection of these curves at t_{12} as the most probable parametric location of the interface ab along line segment $C_1 - C_2$ which is given by equation 2.4.

$$t_{12} = \frac{v(b_1) - v(a_1)}{v(a_2) - v(a_1) - v(b_2) + v(b_1)} \tag{2.4}$$

We can improve upon this approximation by incorporating the normal information at cells C_1 and C_2. Rather than linear line segments associated with materials a and b between cell centroids C_1 and C_2 we

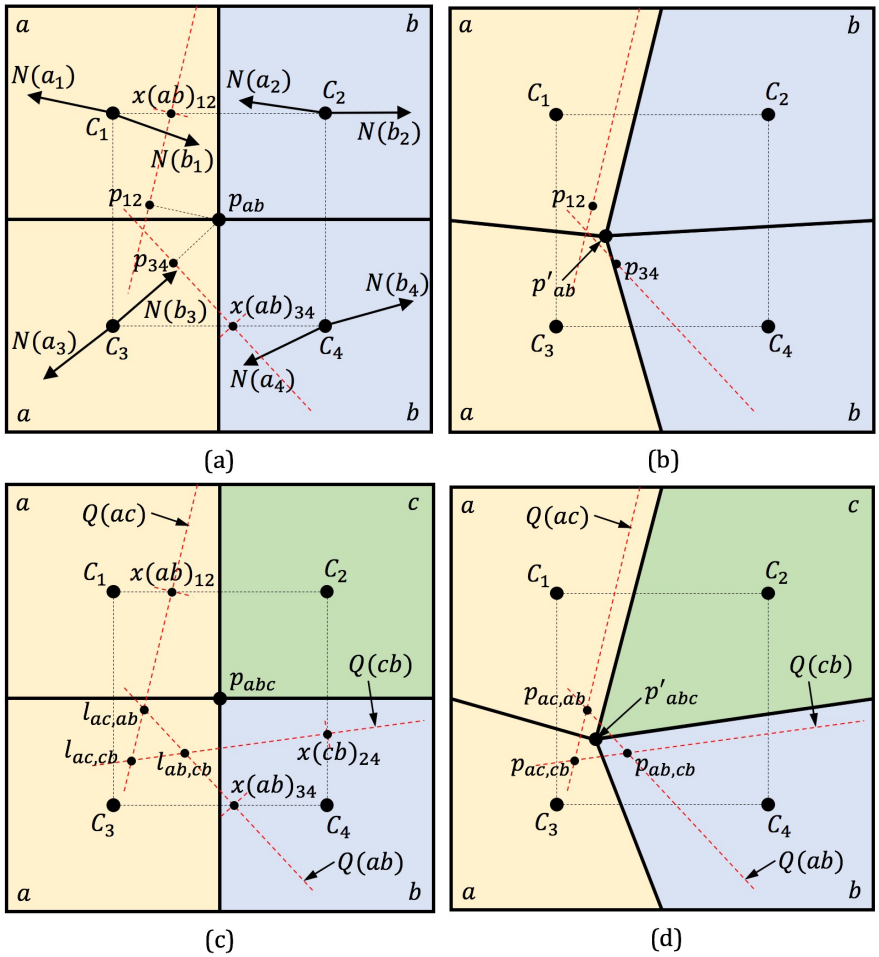


Figure 2-23. Examples of relocating nodes p_{ab} and p_{abc} on surfaces and curves. (a) and (b) describe relocating surface node p_{ab} at the interface between materials a and b by projecting to planes defined by edge cross locations $x(ab)_{ij}$. (c) and (d) describe relocating node p_{abc} on a curve at the interface between materials a, b and c. Edge cross locations define multiple planes that are intersected to form linear curves to which p_{abc} is projected.

can incorporate normal information to represent two hermite curves $a(t)$, $b(t)$ as illustrated in figure 2-24(c) and given by equations 2.5 and 2.6.

$$a(t) = v(a_1)(2t^3 - 3t^2 + 1) + v(a_2)(-2t^3 + 3t^2) + n(a_1)(t^3 - 2t^2 + t) + n(a_2)(t^3 - t^2) \quad (2.5)$$

$$b(t) = v(b_1)(2t^3 - 3t^2 + 1) + v(b_2)(-2t^3 + 3t^2) + n(b_1)(t^3 - 2t^2 + t) + n(b_2)(t^3 - t^2) \quad (2.6)$$

where $n(a_i)$ and $n(b_i)$ are the scalar components of normals $N(a_i)$ and $N(b_i)$ in the direction of the segment $C_1 - C_2$. The intersection of curves $a(t)$ and $b(t)$ will yield an improved interface location t'_{12}. Because the solution of the intersection of the cubic equations in 2.5 and 2.6 is non-trivial, we instead

employ a numerical root finding method to efficiently compute location t'_{12} using t_{12} as an initial guess.

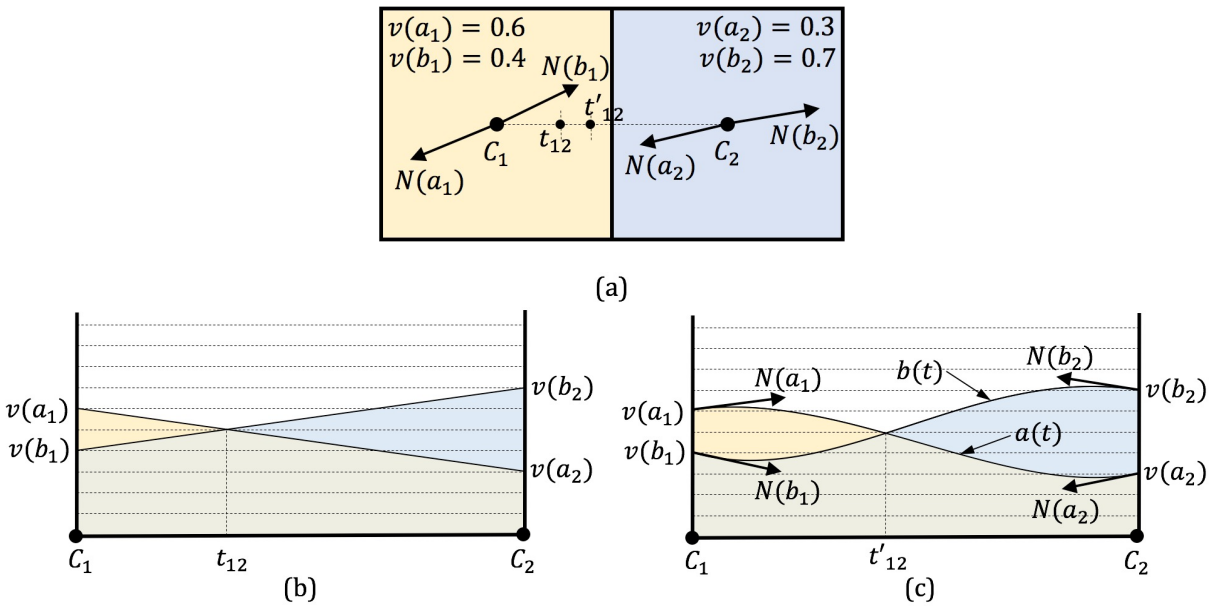


Figure 2-24. Example of computing parametric edge cross location t_{12} on dual edge $C_1 - C_2$. (a) Two adjacent cells with different materials a and b with their associated volume fraction data $v(a_i), v(b_i)$ and normals $N(a_i), N(b_i)$. (b) Line segments defined by slopes $v(i_2) - v(i_1)$ are intersected to define parametric edge cross location t_{12} using equation 2.4. (c) Improved parametric edge cross location t'_{12} computed by intersecting hermite curves $a(t)$ and $b(t)$ defined by equations 2.5 and 2.6.

2.9.3. Locating nodes on curves

To locate nodes on curves we must take into account the interfaces between three or more materials. Figures 2-23(c) and (d) illustrate the placement of node p_{abc} which lies at the interface between materials a (yellow), b (blue) and c (green). We note in figure 2-23(c) that three edge-cross locations on dual edges surrounding p_{abc} are computed. For dual edge $C_1 - C_2$, the edge cross location $x(ab)_{12}$ is defined. This serves as the basis for a plane $Q(ac)$ which is an approximation of the interface between materials a and c. Similarly, edge-cross location $x(cb)_{24}$ is located on dual edge $C_2 - C_4$ which serves as the basis for plane $Q(cb)$. Finally a third plane, $Q(ab)$ is computed from location $x(ab)_{34}$ on dual edge $C_3 - C_4$. The three planes, $Q(ac)$, $Q(cb)$ and $Q(ab)$ now form the basis for relocating node p_{abc}. To do this we first intersect pairs of surfaces to define linear curves. For example, figure 2-23(c) shows $l_{ac,cb}$ as the intersection between plane $Q(ac)$ and $Q(cb)$. In the 2D figure 2-23(c), $l_{ac,cb}$ is represented as a point, however in 3D the intersection of planes is in fact a linear curve. We can then project point p_{abc} to linear curve $l_{ac,cb}$ to get location $p_{ac,cb}$. Similarly we can intersect planes $Q(ac) - Q(ab)$ and $Q(ab) - Q(cb)$ to arrive at locations $p_{ac,ab}$ and $p_{ab,cb}$. A weighted average of these three points will yield location p'_{abc}.

Generalizing this procedure for an arbitrary number of adjacent materials, we form n planes $Q(ij)(i = 1...n, j = 1...n, i \neq j)$ where n is the number of dual edges defined by faces at p_{abc} and where adjacent material $i \neq j$. Intersecting unique pairs of planes will define k linear curves where $k = \sum_{i=1}^{n-1} i$. The weighted average of the projection of point $p_{i,j,...n}$ onto the k linear curves will yield point $p'_{i,j,...n}$.

We note that the procedures for relocating nodes on curves and surfaces described above and illustrated in figures 2-23 and 2-24 are relevant for nodes interior to the RVE boundary. Nodes on the RVE domain boundary are subsequently projected to one of the six planes, twelve edges or eight vertices of the bounding domain depending upon their classification.

2.10. TET MESHING

The Sculpt application does not currently provide the ability to generate tetrahedral meshes directly. Instead, the proposed methods are used to extract and simplify geometry based on volume fraction data to generate a facet-based geometry that can be meshed in an external tool. For our purposes we utilize Cubit [33] which integrates the third party tools CAD-Surf [8] for surface meshing and Tetra-Meshgems [9] for volume meshing. This allows us to take advantage of the full features of Cubit for assigning and controlling sizes and mesh quality, while using Sculpt as means for generating geometry and topology of the microstructures.

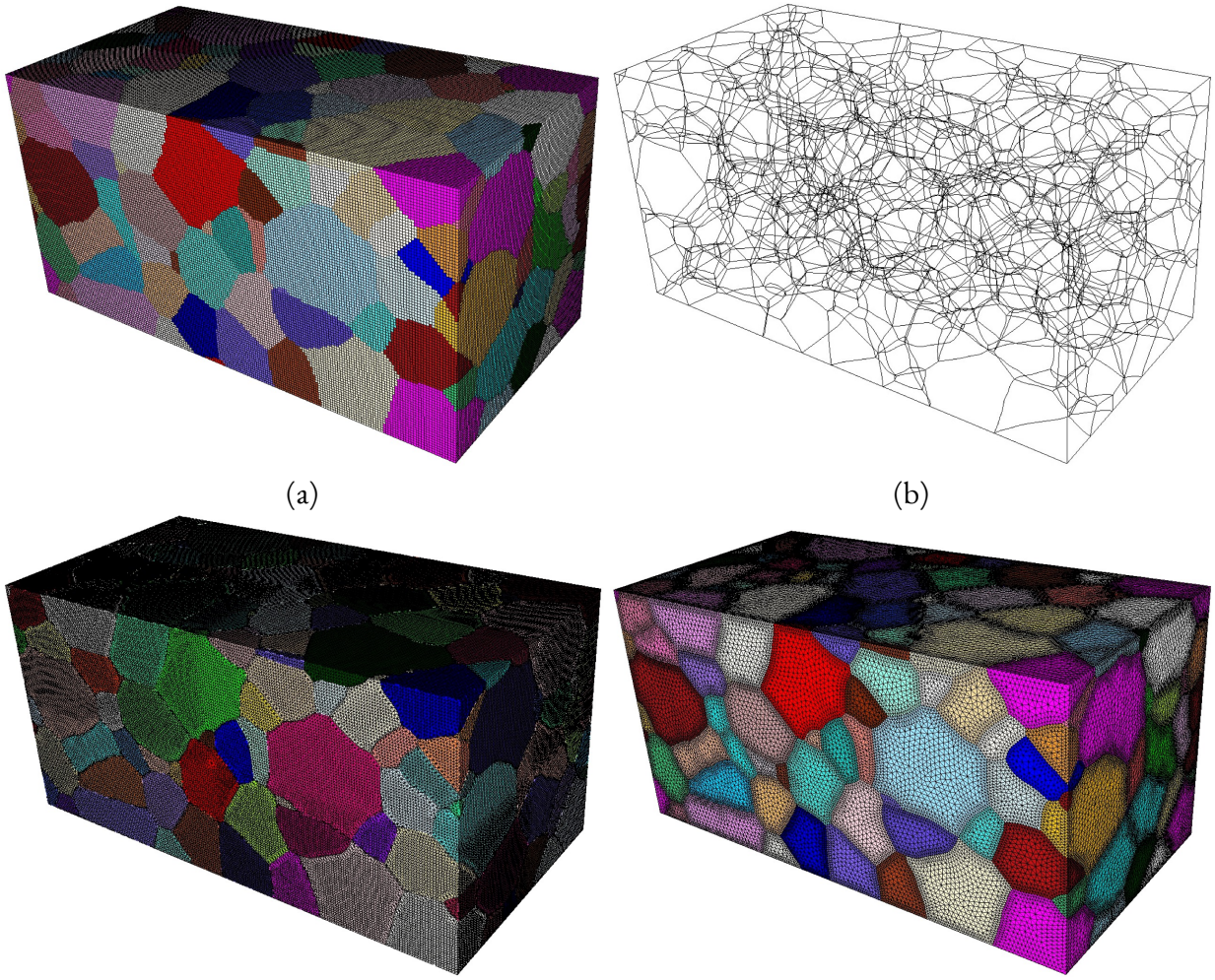
The tet meshing procedure involves the same methods used in hex meshing, however interior hexes and pillowing operations are not performed. To generate the facet-based surfaces, the cell faces contained by each of the surfaces are first split into triangles, geometry is simplified and then exported for use in Cubit. An overview of the procedure is illustrated in figures 2-25 and 2-26.

2.10.1. Surface Triangulation

After a boundary representation (see sec. 2.8) has been defined, a set of surfaces and curves will exist at the interfaces of materials. Surfaces, composed of quad faces, can be split to define triangles that are used for the geometry description. We note that these triangles are not used directly in the final tet mesh, but are used only to describe the geometric definition of the material interfaces. While this reduces the requirement for triangle mesh quality, all triangles must however must be non-inverted. For quads that have nodes projected to curves near the boundaries, this involves first checking for potential triangle inversion cases and splitting a quad at the *best* diagonal to avoid inversion. In rare cases where three or more nodes of the same quad are associated with the same curve, the quad can be split into three or four triangles to avoid inversion.

2.10.2. Triangle Collapsing

In practice, the curves and surfaces generated by this procedure can be problematic for tet meshing. Small curves and surfaces can result in poor quality tetrahedra. To address this issue, small curves and surfaces can be automatically collapsed resulting in an improved geometric representation. To effect



(a) (b) (c) (d)

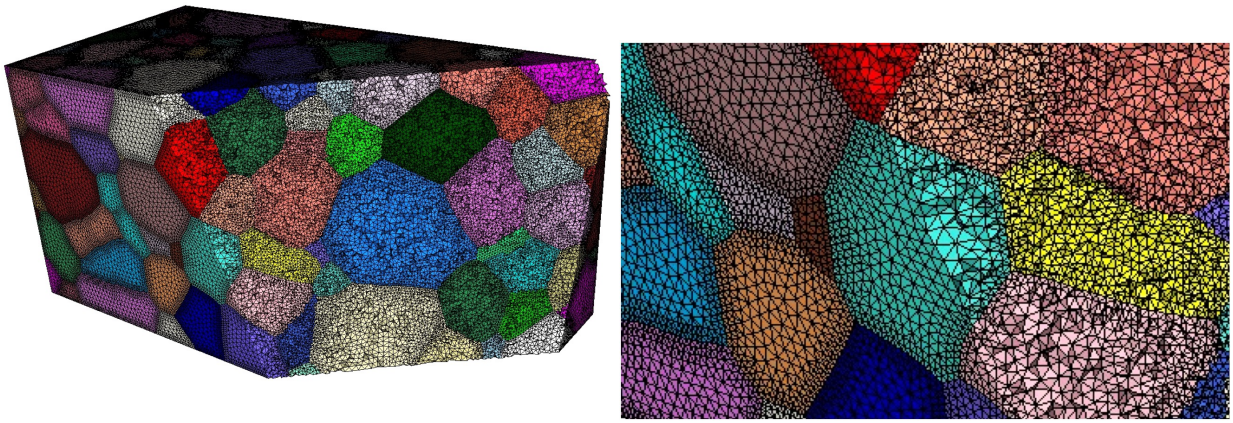
Figure 2-25. Example generation of tetrahedral mesh (a) Initial Cartesian grid with pure cell material assignment (b) Curves defined by Sculpt boundary representation. (c) Facet mesh defined by Sculpt (exported to file) (d) Surface and tet mesh generated in Cubit.

these collapse operations, curves containing a single triangle edge are collapsed into a vertex and surfaces containing less than three triangles are collapsed into a curve. Figure 2-27 shows the microstructure geometry where an automatic curve collapse has been performed.

2.10.3. Facet Export

Smoothing operations are also performed on the surface facets. A simple laplacian smoothing procedure is used for the triangles which results in reasonably smooth surfaces.

Once an acceptable geometry representation is developed in Sculpt, the resulting facet-based



(a) (b)

Figure 2-26. Cut-away and closeup of Tet mesh shown in figure 2-25

representation can be exported as mesh file containing triangles only. To assist Cubit in geometry construction, an additional file is also exported that defines the geometry entities and their association to the facets. An example workflow for generating tetrahedral meshes from volume fraction input data for an RVE is described in Appendix J.

2.11. HEX MESHING

To generate hexahedral elements, we first construct hexahedral elements based on the overlay grid definition followed by a pillowing procedure that inserts layers of hexes at the geometric interfaces.

2.11.1. Generate Interior Hexes

The overlay grid definition provides the interior definition for the hexahedral elements. After the BREP definition has been defined (see sec. 2.8), each cell, will have been assigned to a specific material and volume. These cells are used to construct interior hexahedral elements for use in the final mesh and are assigned materials based on their enclosing volume.

2.11.2. Pillowing

After relocating nodes to curves and surfaces, while cells interior to volumes are generally of good quality, those hexes with nodes on surfaces and curves can often be unacceptable for analysis. To improve mesh quality around surfaces and curves we introduce three separate pillowing operations: volume pillowing, surface pillowing and boundary pillowing. For non-manifold multi-material CAD models, two stage pillowing techniques are introduced in [22] and [31]. This work extends and applies this technique for complex non-manifold grain geometries in an RVE.

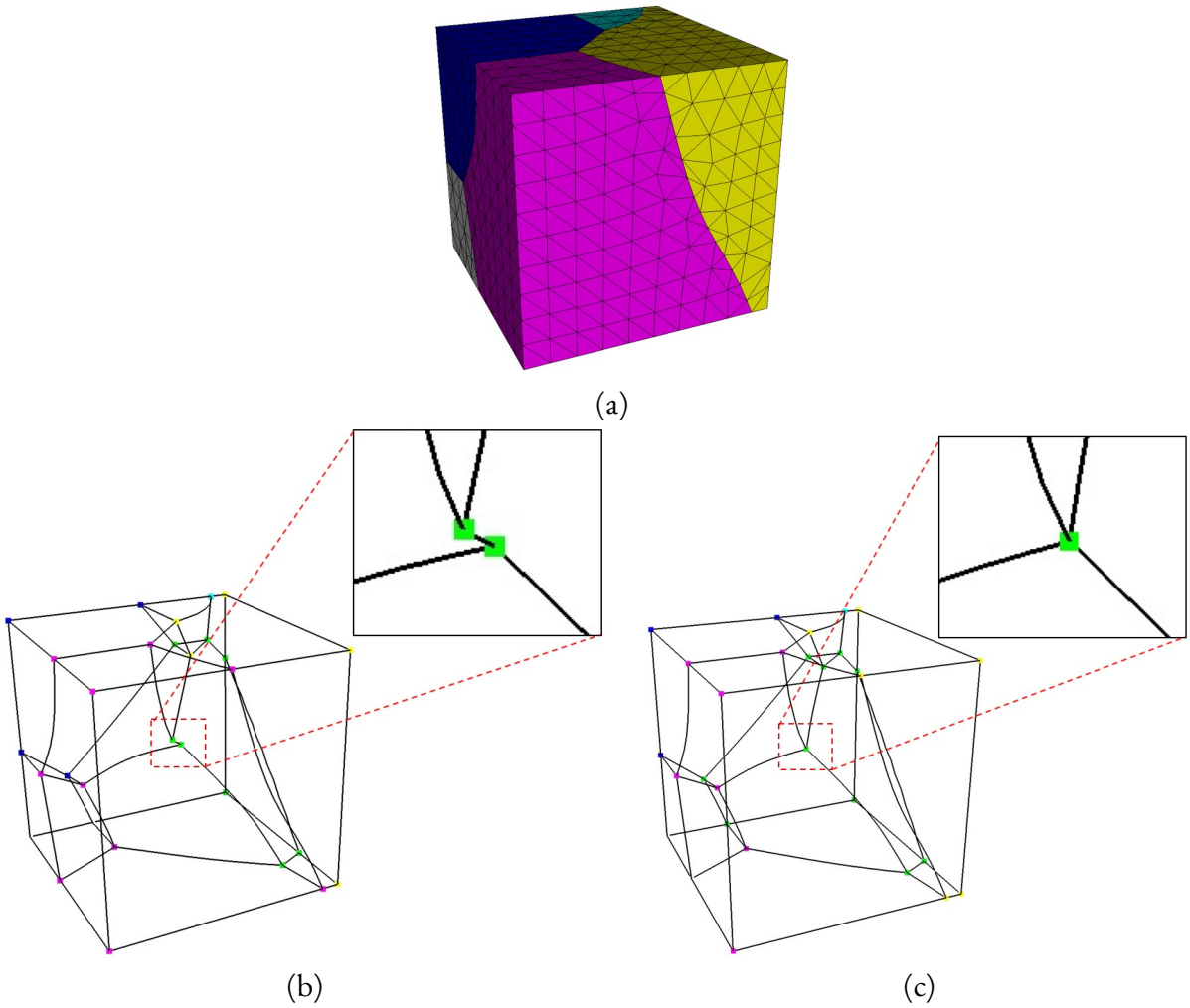


Figure 2-27. Example of curve collapse operation (a) Initial microstructure geometry defined from triangles. (b) Wireframe geometry showing close-up of small curve detected in geometry. (c) Close-up showing small curve collapsed into a vertex.

Pillowing is a technique for inserting a layer of hexes into an existing mesh with the objective of modifying the local hex topology to allow more freedom for smoothing and optimization to improve the mesh quality. This is done by first selecting a contiguous set of hexes as a shrink set. These hexes are effectively pulled or shrunk away from their neighboring hexes not in the shrink set. This duplicates the nodes and faces where they are pulled apart creating two identical and parallel discrete surfaces. Each node and face on these parallel surfaces has a corresponding node and face on the other surface, but separated by a small distance orthogonal to their local tangent planes. Connecting the paired nodes and faces on the parallel surfaces is done to form hexes that define a continuous pillow layer. Subsequent smoothing operations are also done to improve local element quality of the hexes in the pillow layer and those hexes surrounding it.

2.11.3. Volume pillowing

Upon relocating nodes to surfaces, it is likely that multiple faces of the same hex lie on the same surface resulting in dihedral angles close to 180 degrees. When this occurs, it is impossible for smoothing to improve the mesh quality. The volume pillowing operation can remedy this condition. As illustrated in figure 2-28, pillowing operations are performed on each volume independently, selecting all of the hexes in the volume as the shrink set. The new pillow layer will form a new hex from each of the faces that lie on the surface, ensuring that no more than one face from a hex lies on the same surface. Figure 2-28(a) shows an example of a B-Rep topology extracted from a base grid. In figure 2-28(b) the pillow layers generated at the volume boundaries for this model are illustrated, with figure 2-28(c) displaying the full mesh following pillowing. We note that hexes are only inserted at interior surfaces, allowing the pillow layer to terminate at the RVE boundary.

2.11.4. Surface pillowing

Following the volume pillowing operations, although hexes attached to faces on the interior of surfaces will be improved, it is likely that two or more edges of the same hex will lie on a common curve bounding the surfaces. Figure 2-28(d) illustrates the volume pillow layer for a single grain geometry. In this case, a curve separates two of its surfaces A and B. Here we note that two edges of some of the hex faces lie on this curve resulting in angles close to 180 degrees. Once again, this makes it impossible for smoothing operations to correct mesh quality at these locations. To improve this condition we introduce the surface pillowing operation, shown in figures 2-28(e) and (f). This is done by visiting each interior surface in the model. All hexes that share a face on the surface are selected as the shrink set. This will include hexes in the two volumes immediately adjacent to the surface. This introduces a layer of hexes that completely wraps and encloses each interior surface of the model as illustrated in figure 2-28(e). This has two positive effects on the mesh. Most importantly it introduces a layer of quads surrounding each curve on the interior of the RVE and ensures no hex will share more than a single edge on a curve. Secondly it introduces an additional orthogonal layer of good quality hexes at each interior surface which provides an effective boundary layer between grains as illustrated in 2-28(c).

2.11.5. RVE Domain boundary pillowing

One additional option for pillowing is the introduction of a single layer of hexes at each of the six faces of the RVE. This may be necessary as a result of refinement operations performed on the initial Cartesian grid. Reference [24, 25] describes the 3-template as an operation that can manage refinement of convex regions. When a 3-template is performed at the boundary of the RVE it may result in a hex with two of its faces on a common surface, resulting in a case that will not improve with smoothing. Introducing an additional pillow layer at each of the six faces of the RVE, as illustrated in figure 2-29, will correct for this condition and allow smoothing to generate quality elements at the boundaries.

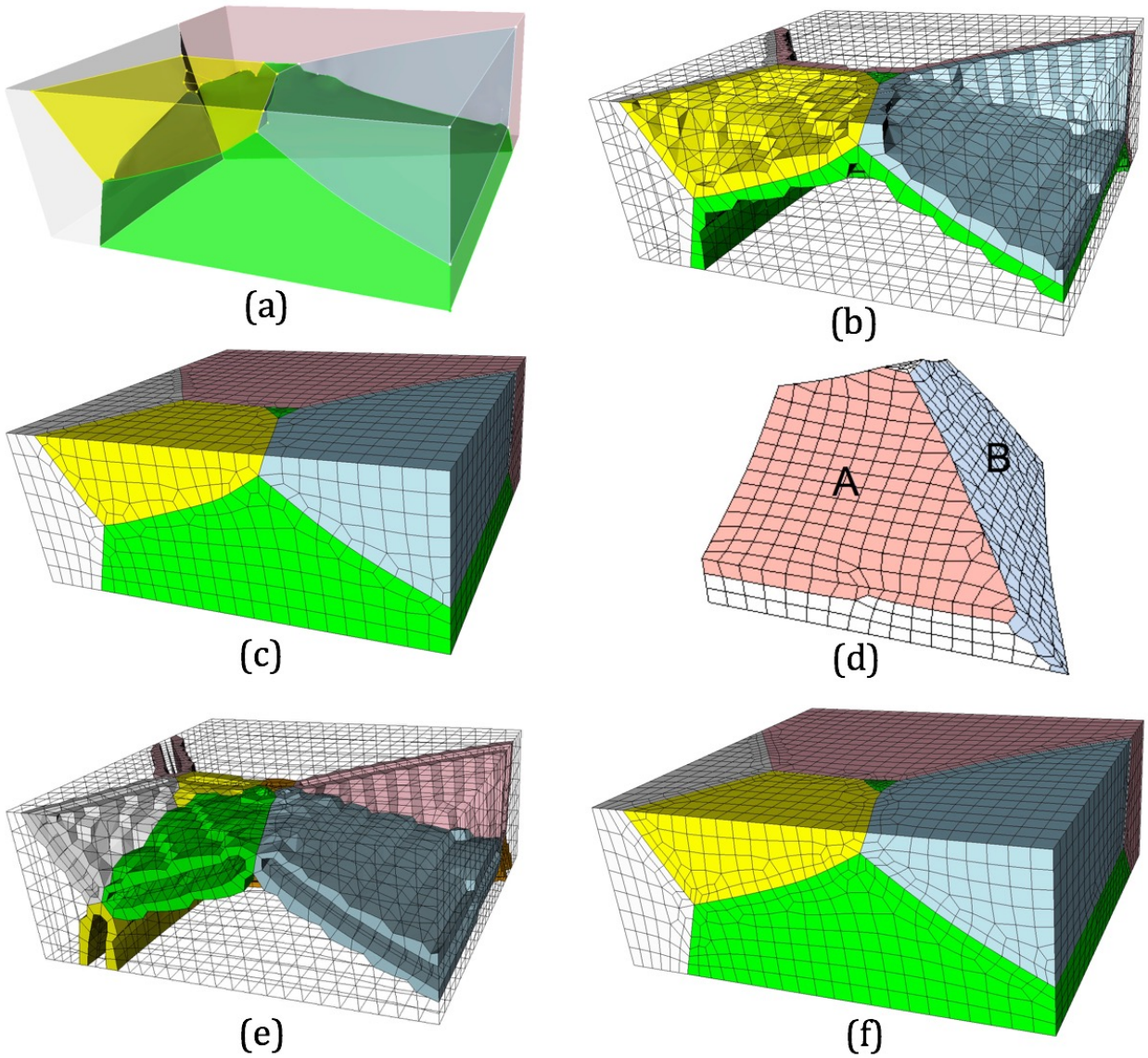


Figure 2-28. Example of volume and surface pillowing operations (a) Initial topology of microstructure grains, (b) Pillow layers displayed at interface boundaries for all materials. (c) Solid view with volume pillows inserted. (d) Single grain following volume pillowing. Hexes at curve interface between surfaces A and B contain faces with 3 nodes on the curve. (e) Pillow layers shown following surface pillow operation. (f) Solid view following surface pillowing

2.12. SMOOTHING

Among the challenges faced with grid-based hex meshing methods is the critical mesh quality issue. Although grid-based methods can be general purpose and fully automatic, typical results can often yield

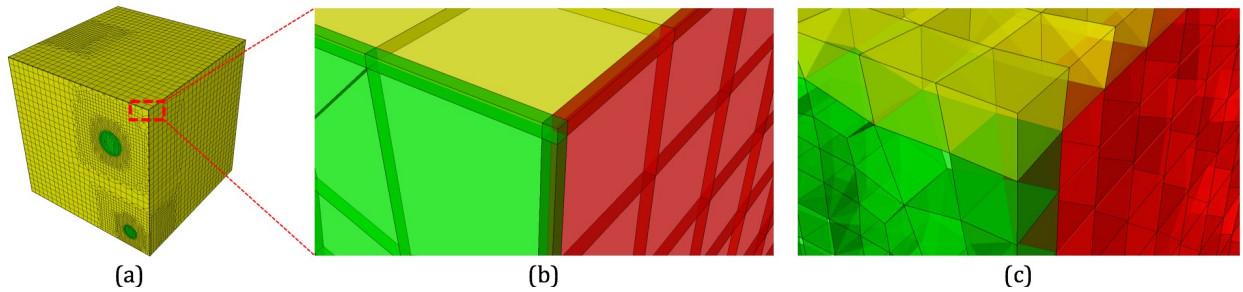


Figure 2-29. Illustration of domain boundary pillowing. (a) Pillows inserted at 6 faces of RVE. (b) Close-up view of pillow layers before smoothing. (c) Close-up view of pillow layers after smoothing.

unusable elements near the boundaries. In this report 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 limit our application of smoothing to volumetric domains bounded by implicit surface representations common to computational materials models. 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 is expected, regardless of the number of processors used or decomposition strategy employed. This work also advocates and leverages the work of prior authors and practitioners in this field [6, 15, 10, 16, 17].

2.12.1. Scaled Jacobian Metric

For our purposes, we define acceptable quality 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 2-30 and equation (2.7). The scaled Jacobian metric for a hex is then taken as the minimum of the eight determinant calculations as in equation (2.8).

$$(J_s)_I = det\left\{\hat{E}_i\hat{E}_j\hat{E}_k\right\}^\top \tag{2.7}$$

$$J_s = min\left((J_s)_I, I = 0, 1, ...7\right) \tag{2.8}$$

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.

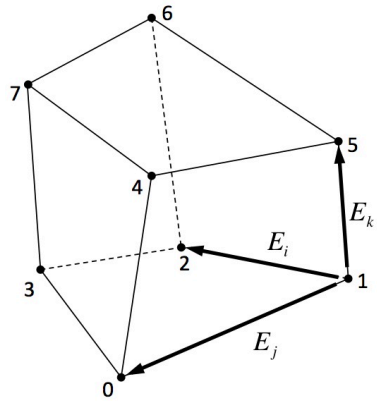


Figure 2-30. Ordered edges E_i, E_j, and E_k are used to compute Scaled Jacobian at 1

While scaled Jacobian is often used as an objective for mesh quality improvement, it does not take into account aspect ratio or target mesh size. Since edge vectors $\hat{E}_i\hat{E}_j\hat{E}_k$ are normalized, a high aspect ratio parallelepiped may yield the same result as an equilateral cube. In the same manner, a tiny isotropic element with respect to a target mesh size, will yield the same result as a large element provided all edges are proportionally scaled.

To control for aspect ratio and target mesh size, we propose a size scale factor S_f on the scaled Jacobian as shown in equation 2.9.

$$(J_s)_I = S_f det\left\{\hat{E}_i\hat{E}_j\hat{E}_k\right\}^\top \tag{2.9}$$

$$S_f = \begin{Bmatrix} e_s \leq S_t, & \frac{e_s}{S_t} \\ e_s > S_t, & \frac{S_t}{e_s} \end{Bmatrix} \tag{2.10}$$

$$e_s = min(\|E_i\|, \|E_j\|, \|E_k\|) \tag{2.11}$$

where S_t is a target mesh size.

2.12.2. Smoothing Overview

Our method includes a tiered approach to smoothing to improve element quality. It starts by applying smoothing to all nodes in the mesh and progressively restricts the smoothing operations to only those nodes that fall below a user-defined scaled Jacobian threshold. The three smoothing phases include:

- *Laplacian Smoothing:* Applied to all elements. Very inexpensive fast approach to improve quality, but can result in degraded element quality if applied to excess. Laplacian smoothing is normally done in combination with optimization smoothing.

- *Optimization Smoothing:* Applied only to elements who's scaled Jacobian falls below a threshold parameter (default 0.6) and their surrounding elements. This approach uses a more expensive optimization technique to improve regions of elements simultaneously. Because this method optimizes node locations simultaneously, neighboring nodes with competing optimum can sometimes limit mesh quality.
- *Parallel Color Smoothing:* Applied only to elements who's element quality falls below a threshold parameter (default 0.2). This technique is the most expensive of the techniques, but focusses only on nodes that are immediately adjacent to poor quality hexes. Each node is smoothed independently of its neighbors, and may require a high number of iterations to achieve desired results.

We provide details for each of these methods in the sections that follow.

2.12.2.1. Parallel Considerations

Smoothing on domains that are distributed across multiple processors, present some interesting challenges. 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 first utilize a Jacobi-based approach for smoothing followed by parallel coloring. Both approaches are used to ensure parallel consistency.

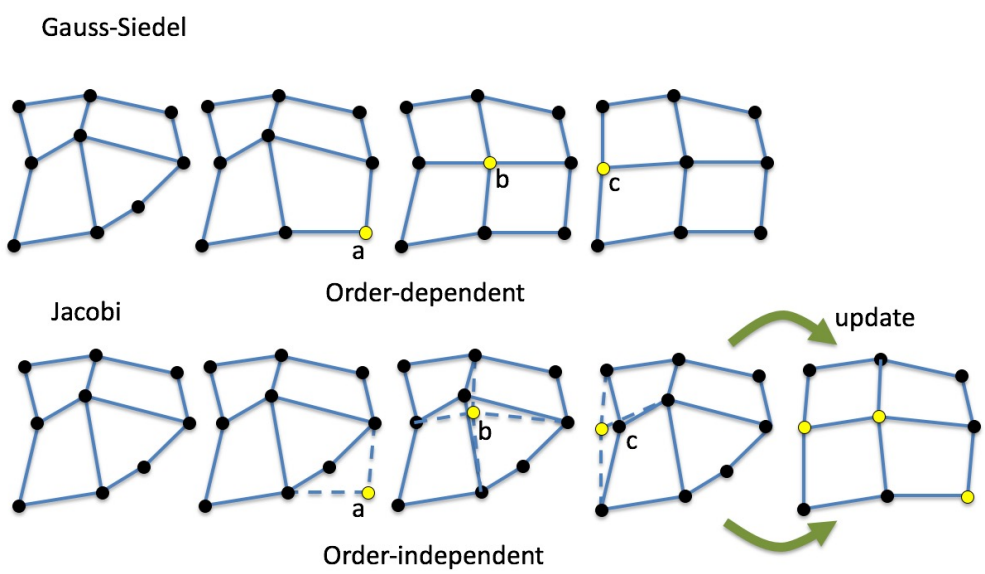


Figure 2-31. Illustration of the difference between Jacobi and Gauss-Siedel smoothing.

In most serial applications, a Gauss-Seidel approach is normally used. Figure 2-31 illustrates the difference between Gauss-Siedel and Jacobi-vbased smoothing. For Gauss-Seidel smoothing, mesh quality improvement is normally performed through multiple iterations of visiting each node of the mesh. New node locations are computed as a function of neighboring node locations, themselves having been potentially moved during the same iteration. The traditional Gauss-Seidel approach

unfortunately results in order-dependency that cannot be guaranteed for different distributions. We note that in figure 2-31 that smoothed locations for nodes B and C for the Gauss-Seidel case are based on having previously moved node A.

Jacobi-based methods, however rely only on the initial state of the nodes at the beginning of an iteration. Once node locations are determined, a single update is performed to change node locations in preparation for a subsequent iteration. Locations for nodes B and C in figure 2-31 for the Jacobi case, are based only on the initial locations of all surrounding nodes allowing them to be order-independent.

Gauss-Seidel approaches will normally convergence faster since improved node locations can be achieved within a single iteration based on neighboring nodes that have already been improved and updated. In practice, however, the Jacobi procedure we present appears to improve mesh quality to acceptable levels within very few iterations.

Parallel coloring is also an attempt to avoid order dependency needed for our application. Coloring is a common technique used in parallel computing [4][10] that attempts to isolate independent sets of items in a graph-based domain for separate treatment.

2.12.3. Combined Laplacian and Optimization Smoothing

Figure 2-32 summarizes the Jacobi-based procedure used for combined Laplacian/Optimization based smoothing. 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 [26, 27].

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.

2.12.3.1. Laplacian Smoothing

As outlined in figure 2-32, this work depends heavily on initial Laplacian smoothing to improve node locations. Laplacian smoothing is an iterative method that averages the node locations from surrounding elements. In practice, applying only two iterations of Jacobi-based Laplacian smoothing, appears to correct the majority of inverted elements.

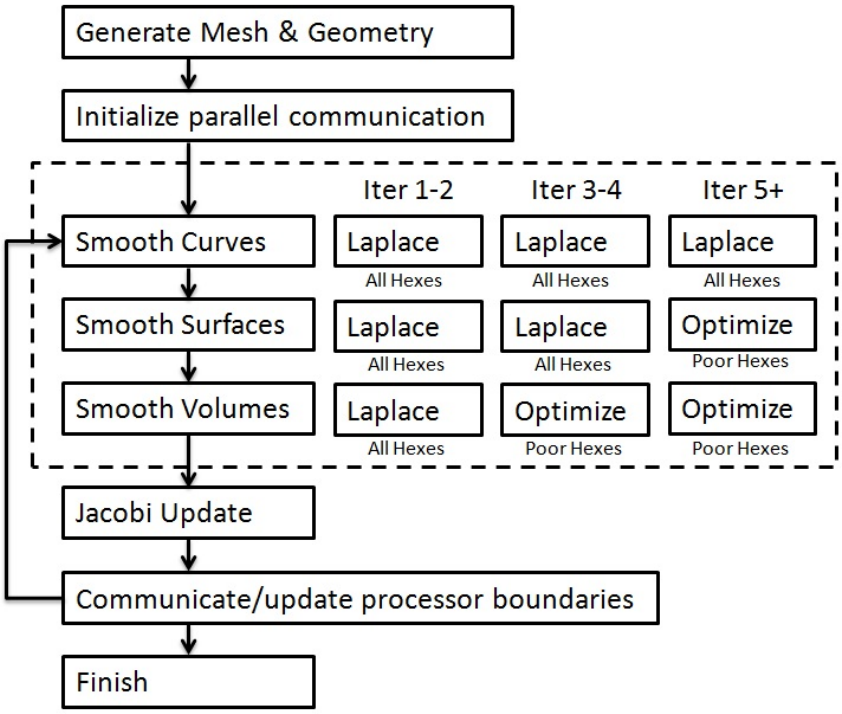


Figure 2-32. Procedure used for combined Laplacian and Optimization smoothing

2.12.3.2. Optimization Smoothing

Following Laplacian smoothing, we utilize a targeted optimization to those nodes near elements that fall below a threshold quality metric. We choose $J_s \leq 0.6$ as the threshold criteria and expand one layer of elements to include additional nodes in the smoothing domain.

For optimization, we propose dual objective functions. When all elements attached to a node have positive J_s, then maximizing the local Scaled Jacobian value at a node can be used as the objective. However, if $J_s < 0$, then maximization of the local negative volume V_n at the node must be used. We note that the Scaled Jacobian value is well behaved for $J_s > 0$. However, for $J_s < 0$, with standard optimization techniques, the gradient of J_s is not guaranteed to be monotonically increasing. As a result, we switch to maximizing V_n when elements become inverted.

The technique of maximizing the local negative volume at a node is also referred to as *untangling* as it tends to un-invert or untangle severely distorted elements to a point where the Scaled Jacobian objective can take over.

To facilitate parallel-consistent Jacobi-based smoothing, we propose a simple node-by-node optimization method as outlined below:

1. Compute the minimum scaled Jacobian J_s from the hexes attached to node M_i^0. If $J_s > 0$ for all adjacent hexes, then use Scaled Jacobian as the objective, otherwise use the Minimum Volume V_n at the node as the objective.

2. Compute the numerical gradient, ∇J_s or ∇V_n 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 or V_n by incrementally moving M_i^0 in the direction ∇J_s or ∇V_n

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 for most cases. We also note that the objective function may switch between optimizing V_n to J_s should the quality improve sufficiently such that all hexes exhibit $J_s > 0$. In a similar manner, the objective function may switch to optimizing negative volume, should hex quality drop below zero.

When all adjacent hexes have positive volume, the objective function is the minimum scaled Jacobian at the node from equation 2.7. While only the node we are optimizing will be in motion, the angle at its three attached nodes may also be affected by its movement. Therefore we use the minimum scaled Jacobian from the node and its three attached nodes. For example in figure 2-30, the scaled Jacobian at nodes 1, 0, 2 and 5 contribute to J_s at node 1. Since other nodes on the hex do not move, we can neglect the scaled Jacobian at other nodes on the hex.

For the untangling case, negative volume at a node is computed as the oriented volume of the tetrahedron defined by the node and its 3 attached nodes. We compute V_n by summing the negative volume at the node and all of its attached nodes in the hex. Nodes where local volume at the node is positive, are neglected in the calculation of V_n, so the maximum value for V_n will be zero.

The objective function is therefore one of either:

$$J_s = min\left((J_s)_I, I = 0, 1, ...n_{hex}\right) \tag{2.12}$$

$$V_n = min\left((V_n)_I, I = 0, 1, ...n_{hex}\right) \tag{2.13}$$

where $(J_s)_I$ and $(V_n)_I$ is the scaled Jacobian and negative volume respectively at the node for the Ith adjacent hex and n_{hex} is the number of adjacent hexes.

2.12.3.3. Volume Optimization

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

$$(\nabla J_s)_x = \left\{\frac{J_s(x_0 + \varepsilon_x) - J_s(x_0)}{\varepsilon}\right\} \tag{2.14}$$

$$(\nabla V_n)_x = \left\{\frac{V_n(x_0 + \varepsilon_x) - V_n(x_0)}{\varepsilon}\right\} \tag{2.15}$$

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

2.12.3.4. Surface Optimization

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.

$$(\nabla J_s)_u = \left\{ \frac{J_s(x_0 + \varepsilon_u) - J_s(x_0)}{\varepsilon} \right\} \tag{2.16}$$

$$(\nabla J_s)_v = \left\{ \frac{J_s(x_0 + \varepsilon_v) - J_s(x_0)}{\varepsilon} \right\} \tag{2.17}$$

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, described below, the location is updated by projecting to the surface definition prior to computing $J_s(x_n)$.

Line Search: Once a vector gradient ∇J_s or ∇V_n is established, we can begin searching for a new location x_n that provides an improved value for J_s or V_n along the gradient direction.

$$x_n = x_0 + \alpha \nabla \hat{J}_s \tag{2.18}$$

We note that the same procedure is used for untangling, but use maximizing scaled Jacobian as our example. 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 zero in on an improved value for J_s. We do so by computing at each location x_n, a new $J_s(x_n)$ and compare to the previous $J_s(x_{n-1})$. If $J_s(x_n) > J_s(x_{n-1})$, then we update the location of the node. Otherwise the previous value at x_{n-1} is maintained. 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.

In practice, since we are not necessarily searching for an optimum, but rather an improved or computable value for J_s, it was not necessary to iterate on the α extensively. For our application, 4 iterations was sufficient, in general, to achieve an acceptable mesh.

2.12.3.5. Damping

Jacobi smoothing methods can sometimes result in oscillations, where the optimal location for a node conflicts with the optimal location for an immediate neighbor node. To reduce this effect we can employ a damping factor, d_f to the final smoothed locations. For example, if the smoothing current iteration is n of n_{max} possible iterations, we can compute the damping factor as:

$$d_f = \frac{n}{n_{max}} \tag{2.19}$$

and the damped location $\mathbf{X}$ as:

$$\mathbf{X} = \mathbf{X}_i + d_f(\mathbf{X}'_i - \mathbf{X}_i) \tag{2.20}$$

where $\mathbf{X}_i$ is the initial location for Node N_i prior to optimization and $\mathbf{X}'_i$ is the optimized location. Applying a damping factor has the effect of moving the node slowly to its destination over multiple iterations, reducing the chance for oscillation.

2.12.3.6. Conjugate Gradient

For most optimization applications, a conjugate gradient update, that will modify the direction of the gradient, following an initial line search, can improve accuracy and efficiency. As we are moving the location of the node, however, it is likely that the actual hex computed as the minimum $(J_s)_I$ will change based on our choice of α or the objective function may change from untangling to maximizing scaled Jacobian. This can cause the actual gradient field to be discontinuous. This did not seem to adversely affect our method, however, as a consequence, applying conjugate gradient updates of the line search direction did not improve accuracy or reduce computational time. In practice, increasing the total number of iterations beyond 2 or 3 made only minor improvements in mesh quality. As a consequence, setting the maximum number of iterations to 2 proved sufficient for most cases. We also limited the iterations, based upon the minimum $(J_s)_I$ at the node. If the value for J_s exceeded our threshold of 0.2, no additional optimization would be performed.

2.12.4. Parallel Coloring Smoothing

The application of combined Laplacian and optimization-based smoothing including damping and untangling proved successful for most cases, however a small number poor quality elements can persist. As a result, parallel coloring smoothing is invoked on nodes where adjacent hexes fall below a user defined threshold which is defaulted to 0.2.

Parallel coloring smoothing uses at its core the same optimization-based methods described above in 2.12.3.2. What differentiates the two procedures is its application to parallel coloring. The combined Laplacian-optimization described above utilizes a Jacobi-based method that will update all node locations after a single iteration. We note that updating all nodes simultaneously, can result in

neighboring nodes conflicting or oscillating. While damping can reduce this effect, it does not eliminate it. Figure 2-33 shows an overview of the method proposed in this work.

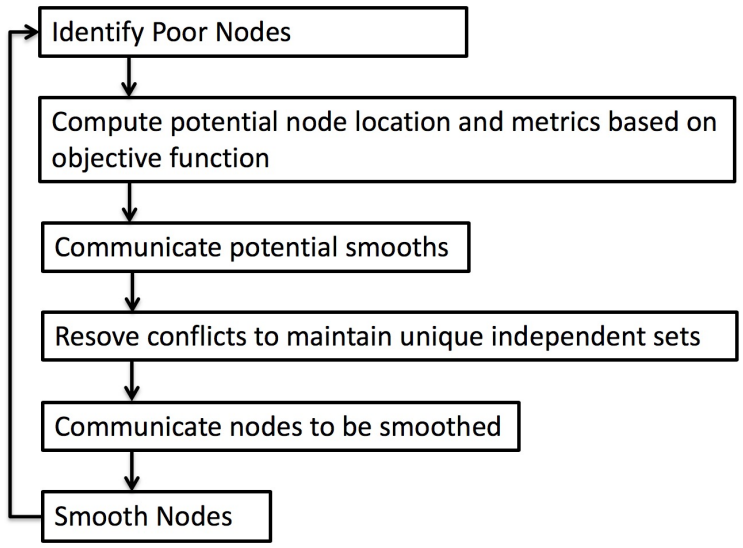


Figure 2-33. Smoothing procedure used for paralleling color method

The set of nodes identified as poor, $\mathbf{N}$, defined as nodes with adjacent elements with $J_s \leq 0.2$ are considered for smoothing. For each node in $\mathbf{N}$ with initial locations $\mathbf{X}_i$ and quality q_i, we compute a potential optimal location $\mathbf{X}'_i$ and potential mesh quality q'_i by using the optimization methods described in section 2.12.3.2. We note that q_i and q'_i may be either a scaled Jacobian or negative volume metric.

Following a communication step to ensure parallel consistency, independent sets are determined. For our purposes, an independent set includes the set of adjacent hexes to a node where no hex is used more than once for any individual set. To ensure nodes are independent, we eliminate nodes with conflicting hexes based upon the following prioritization where node N_i is compared with node N_j that share at least one common hex:

1. *Initial mesh quality:* if $q_i < q_j$ then N_i is selected and N_j is eliminated.
2. *Potential mesh quality:* if $q_i \approx q_j$ then choose on the basis of q'_i and q'_j. if $q'_i > q'_j$ then N_i is selected and N_j is eliminated.
3. *Location:* if $q_i \approx q_j$ and $q'_i \approx q'_j$ then choose based on the magnitude of coordinates $\mathbf{X}_i$ and $\mathbf{X}_j$ at N_i and N_j. if $\|\mathbf{X}_i\| \leq \|\mathbf{X}_j\|$ then N_i is selected and N_j is eliminated.

Once conflicts are resolved to maintain unique independent sets, a further communication step is performed. Node locations defined by the surviving independent sets are then updated to the previously computed locations $\mathbf{X}'_i$.

This procedure then repeats until a maximum number of iterations is reached or no further progress is detected. We define progress as at least one node increasing its mesh quality, q_i during a single smoothing iteration.

2.12.5. Geometry Considerations

Node smoothing and projection consists of several iterations of successively smoothing and projecting nodes M_j^0 on geometric entities starting with vertices $G_i^0(M_j^0)$, curves $G_i^1(M_j^0)$, surfaces $G_i^2(M_j^0)$, and then volumes $G_i^3(M_j^0)$.

Vertices, curves and surfaces can be identified as either interior, G^I or exterior G^E. It is necessary to distinguish between G^I and G^E so that projection operations for every node M_j^0 can be accurately established based upon an underlying geometric definition. For interior entities, the geometry is described implicitly using the methods described in section 2.9 or from a local quadratic approximation, while exterior entities are described by analytic or mesh-based methods.

2.12.5.1. Exterior Geometry

For an RVE represented by an overlay Cartesian grid the geometric representation can be described by analytically defined planes, curves and vertices. The exterior surfaces, G^{2E} can be defined by the six axis-aligned planes of the Cartesian grid boundaries. Similarly, exterior curves, G^{1E} and vertices, G^{0E} can be represented by its 12 linear curves and 8 vertices.

For the unstructured case, however, where the geometry is defined by an arbitrary hexahedral mesh, a boundary geometry must first be extracted consisting of entities G^{0E}, G^{1E}, and G^{2E}. In this case, A mesh-based geometry description is first derived where the boundary curves and surfaces are represented as a set of piecewise linear triangles and segments. Curve and vertex topology is also extracted in this procedure based upon a prescribed feature angle. Reference [28] describes a similar procedure for extracting a mesh-based geometry representation from an input mesh.

2.12.5.2. Interior Geometry

Nodes on interior geometric entities G^{0I}, G^{1I}, and G^{2I} are normally smoothed by either a Laplacian or optimization-based procedure described above. Projection to an implicitly defined curve or surface can then be performed using the methods described in section 2.9. We note however, that this can sometimes result in *noisy* or *bumpy* material interfaces. To reduce this effect, we utilize a quadratic approximation of the local surface nodes.

For interior surfaces, since an explicit representation of the surface is not available, the quadric approximation of the nodes M_{surf}^0 similar to reference [13] is used. For this approach we seek a projected node location defined by $P_k(x_k, y_k, z_k)$.

$$\begin{aligned} Q_k(x,y) &= z_k + a_{k2}(x-x_k) + a_{k3}(y-y_k) + a_{k4}(x-x_k)^2 \\ &\quad + a_{k5}(x-x_k)(y-y_k) + a_{k6}(y-y_k)^2 \end{aligned} \tag{2.21}$$

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 2-34, where N_k is the surface normal at P_k and (T_1, T_2) are orthogonal tangent vectors. Coefficients $a_{k2,k3,\ldots,k6}$ for equation (2.21) can then be computed by solving the linear system:

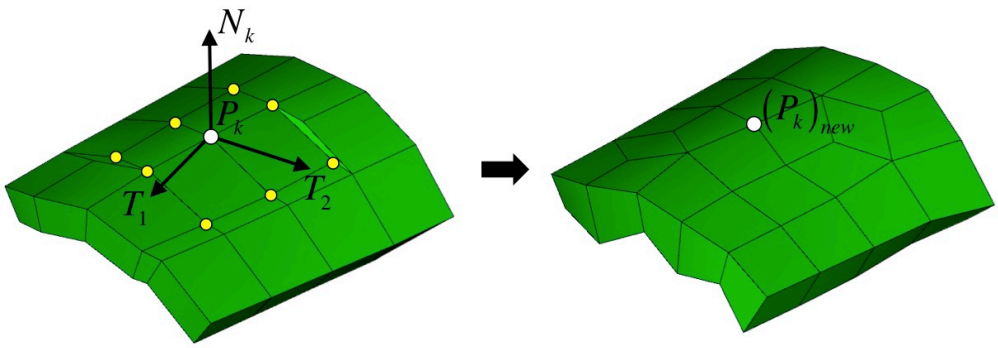


Figure 2-34. Quadric approximation of surface from surrounding nodes at P_k is performed

$$\begin{bmatrix} \sum w_i x^2 & \sum w_i xy & \sum w_i x^3 & \sum w_i x^2 y & \sum w_i x^2 y^2 \\ \sum w_i xy & \sum w_i y^2 & \sum w_i x^2 y & \sum w_i xy^2 & \sum w_i xy^3 \\ \sum w_i x^3 & \sum w_i x^2 y & \sum w_i x^4 & \sum w_i x^3 y & \sum w_i x^2 y^2 \\ \sum w_i x^2 y & \sum w_i xy^2 & \sum w_i x^3 y & \sum w_i x^2 y^2 & \sum w_i xy^3 \\ \sum w_i xy^2 & \sum w_i y^3 & \sum w_i x^2 y^2 & \sum w_i x^2 y^2 & \sum w_i y^4 \end{bmatrix} \begin{Bmatrix} a_{k2} \\ a_{k3} \\ a_{k4} \\ a_{k5} \\ a_{k6} \end{Bmatrix} = \begin{Bmatrix} \sum w_i xz \\ \sum w_i yz \\ \sum w_i x^2 z \\ \sum w_i xyz \\ \sum w_i y^2 z \end{Bmatrix} \tag{2.22}$$

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 or optimization-based 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:

$$(P_k)_{local} = \begin{Bmatrix} (P'_k - P_k) \cdot T_1 \\ (P'_k - P_k) \cdot T_2 \\ a_{k2} x_k + a_{k3} y_k + a_{k4} x_k^2 + a_{k5} x_k y_k + a_{k6} y_k^2 \end{Bmatrix} \tag{2.23}$$

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

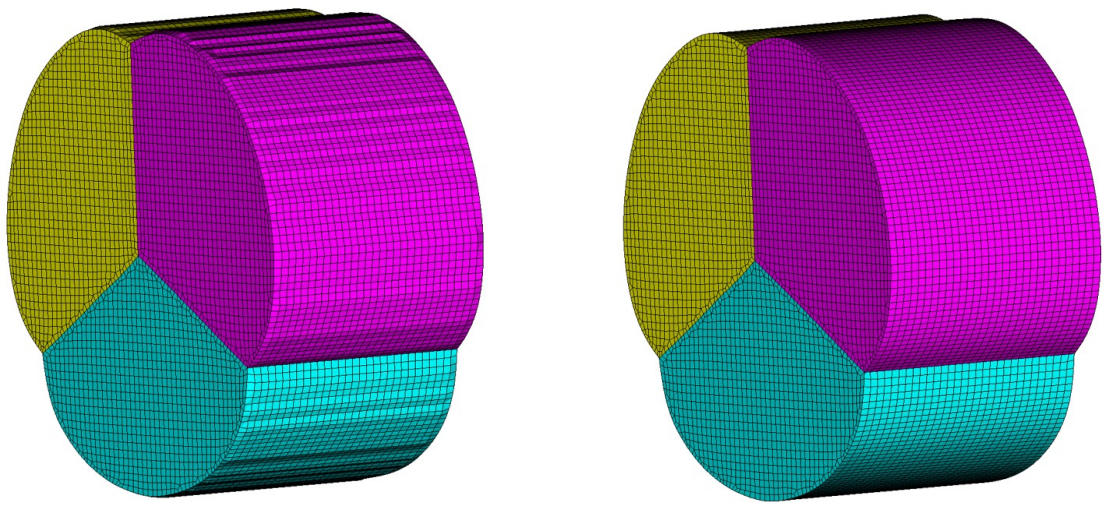
$$(P_k)_{new} = P_k + (P_k)^T_{local} \begin{Bmatrix} T_1 \\ T_2 \\ N_k \end{Bmatrix} \tag{2.24}$$

2.12.6. Geometry-Free Smoothing

In most cases, we have found that projection of nodes to an underlying geometry representation as described in section 2.12.5 provides good results. For some cases, however, the geometry description built from the material data will tend to produce noisy or non-smooth surfaces. In addition, the geometry can also constrain element shapes such that mesh quality improvement is impossible at interior curves and vertices. By relaxing requirements for strict conformance to geometry, smoothness and mesh quality can be improved.

2.12.6.1. Material Interface Smoothness

A consequence of extracting voxelized or coarse volume fraction data to use as the basis for geometry can be noisy or non-smooth surfaces. While there are many options for smoothing noisy surfaces [36, 37], one of the simplest and most effective is to take advantage of the Laplacian smoothing operation without a subsequent projection operation. Figure 2-35 illustrates the effect of removing the projections step following Laplacian smoothing.



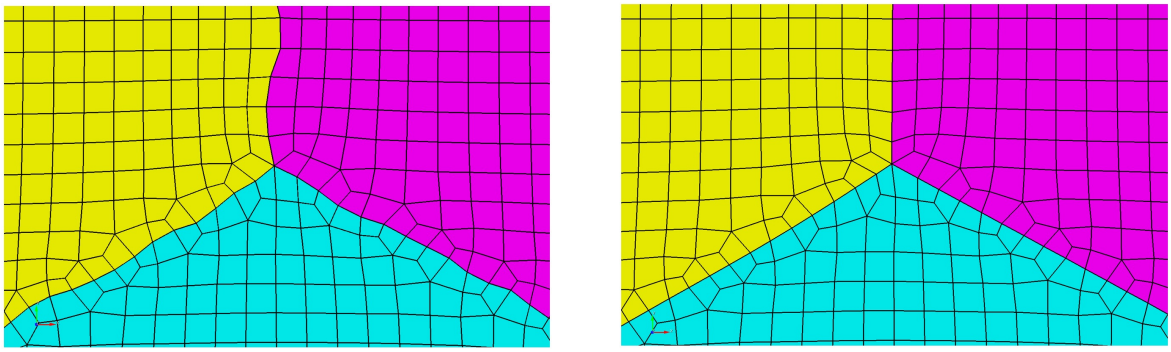
(a) (b)

Figure 2-35. Effect of smoothing without surface projections. (a) Surfaces smoothed with projections using implicit surface definition. (b) Surfaces smoothed without surface projections.

While effective, because Laplacian smoothing will average neighboring node locations, it can have the consequence of reducing or shrinking volume. In the case illustrated in figure 2-35 where surface curvature is minimal or where curvature is limited to one parametric direction, Laplacian smoothing is most effective in removing noise without adversely effecting volume. We note that limiting the number of user-defined Laplacian iterations applied can also limit the effects of shrinking, while still maintaining acceptable results.

To improve curve smoothness, it is advantageous to use a hermite interpolation method to smooth the curve definition rather than projecting to the implicit curve definition outlined in section 2.9.3. For a node, N associated with a curve with coordinates $\mathbf{X}$ we would like to find a location $\mathbf{X}'$ that improves smoothness of the curve. We can identify the two immediately adjacent nodes N_a and N_b on the curve with coordinates $\mathbf{X}_a$ and $\mathbf{X}_b$ respectively. We first define tangent vector at N_a and N_b by averaging the adjacent line segment vectors to compute $\delta\mathbf{X}_a$ and $\delta\mathbf{X}_b$. We can then find a location for N from:

$$\mathbf{X}' = \frac{\mathbf{X}_a + \mathbf{X}_b}{2} + \frac{\delta\mathbf{X}_a + \delta\mathbf{X}_b}{8} \tag{2.25}$$



(a) (b)

Figure 2-36. Effect of hermite smoothing on curves. (a) Curve smoothing using default curve projections. (b) Resulting curves following curve hermite smoothing

The effect of curve hermite smoothing is illustrated in figure 2-36. Note that without hermite smoothing, the curve interfaces can produce a wavy characteristic as in 2-36(a). For this example, 2-36(b) shows improved smoothness of the curves.

2.12.6.2. Mesh Quality Improvement at Curves and Vertices

As noted, it can sometimes be effective to loosen the geometric constraints on some nodes in order to improve mesh quality. This is especially significant at material interfaces where 3 or more materials meet. Pillowing can provide a good solution for these cases by inserting an additional layer of hexes at curves to allow additional freedom for smoothing. In practice, however, for very complex geometry arrangements, insertion of an additional layer of hexes can introduce additional complexity and constraints that cannot be easily be resolved with smoothing.

One solution employed in these cases, rather than inserting pillows to resolve curve interfaces, is to relax the geometric constraints on curves to allow nodes to *float* off of the curves and vertices to improve mesh quality. A curve optimization threshold, C_{OT} can be defined by the user. C_{OT} indicates the value for scaled Jacobian where if a node that falls on a curve that has neighboring quads with J_s less than this value, then the smoothing will no longer honor the curve definition. Instead the optimization smoother will attempt to place the node to optimize the neighboring mesh quality without regard for its placement on its owning curve.

Normally C_{OT} is set at 0.1 to avoid too many nodes from floating off of their owning curves, however, if mesh quality is constrained by curve geometry, setting this value higher can help to avoid bad or poor quality elements. For most stochastic models where material interfaces are statistically determined, minor deviations in curve representations caused by increasing the value for C_{OT} are usually acceptable and can improve mesh quality considerably.

2.12.7. Smoothing Results

To evaluate the effectiveness of the proposed smoothing procedures, an informal study was conducted using a total of 52 geometry models. A representative sampling of the models are shown in figure 2-37. For this study, we evaluate the effectiveness and performance of Laplacian, Optimization and Parallel Coloring Options as well as the use of Damping.

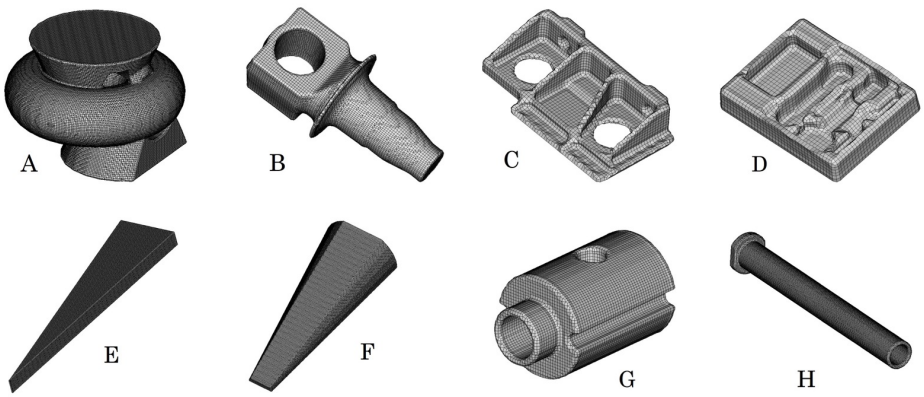


Figure 2-37. A sample of the 52 models used for smoothing tests

Table 2.12.7 and figure 2-38 illustrate the results from four different test cases. For each test, all 52 models were meshed and smoothed using one of four different options:

1. *No Damping:* Combined Laplacian and optimization-based smoothing was run using default options, however damping was not applied.
2. *Damping:* Same as the first test, except damping as described in section 2.12.3.5.
3. *No Damping & Color:* Combined Laplacian, optimization-based and parallel-coloring smoothing were used, except damping was turned off.
4. *Damping & Color:* Same as third test except damping was turned on.

	$J_s < 0$	$0 < J_s < 0.1$	$0.1 < J_s < 0.2$	$J_s > 0.2$	average J_s
No Damping	10	14	11	20	0.083
Damping	6	9	14	26	0.144
No Damping & Color	2	1	3	49	0.217
Damping & Color	2	1	3	49	0.223

Table 2-2. Results from smoothing tests. Four different tests performed using different smoothing options. Table shows number of tests out of 52 that had elements with minimum scaled Jacobian, J_s within each range. Results are also illustrated in figure 2-38

The results indicate a significant improvement in mesh quality when using damping when combined only with Jacobi-based Laplacian and optimization-based smoothing. A similar improvement is also

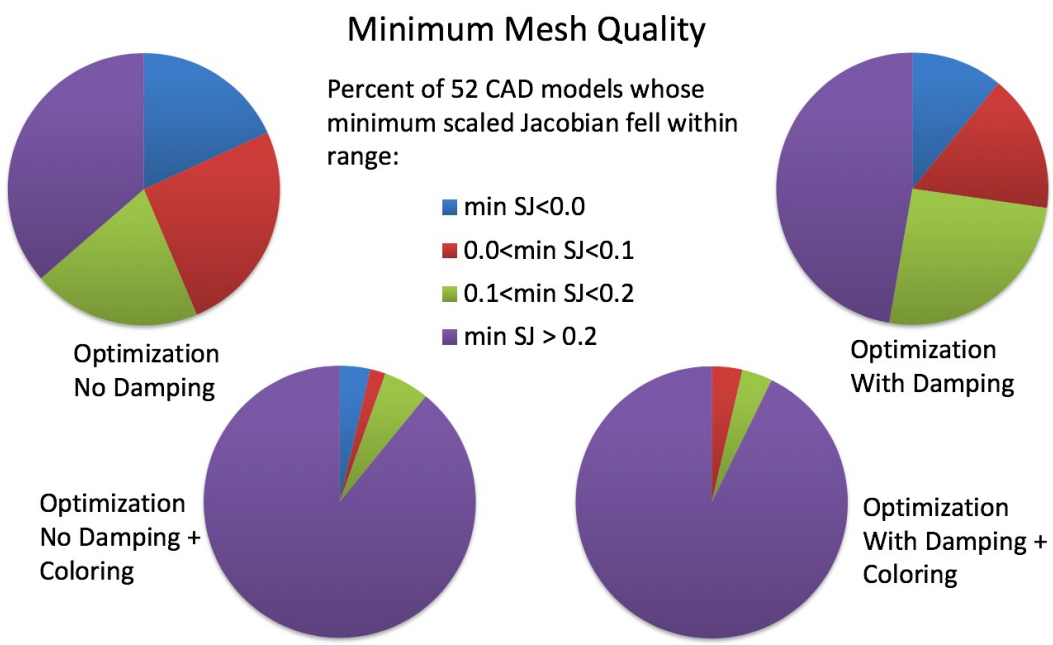


Figure 2-38. Results from smoothing tests illustrating expected minimum mesh quality when using optimization, damping and parallel coloring to smooth meshes

noted when adding parallel coloring even without damping. This would suggest that both parallel coloring and damping provide similar improvement characteristics. Adding both damping and coloring provided a minor increase in average minimum quality as illustrated in figure 2-39(a). We note however that for our tests, once the threshold of $J_s \geq 0.2$ was achieved, that no further smoothing is performed. Modifying the threshold may result in additional improvement.

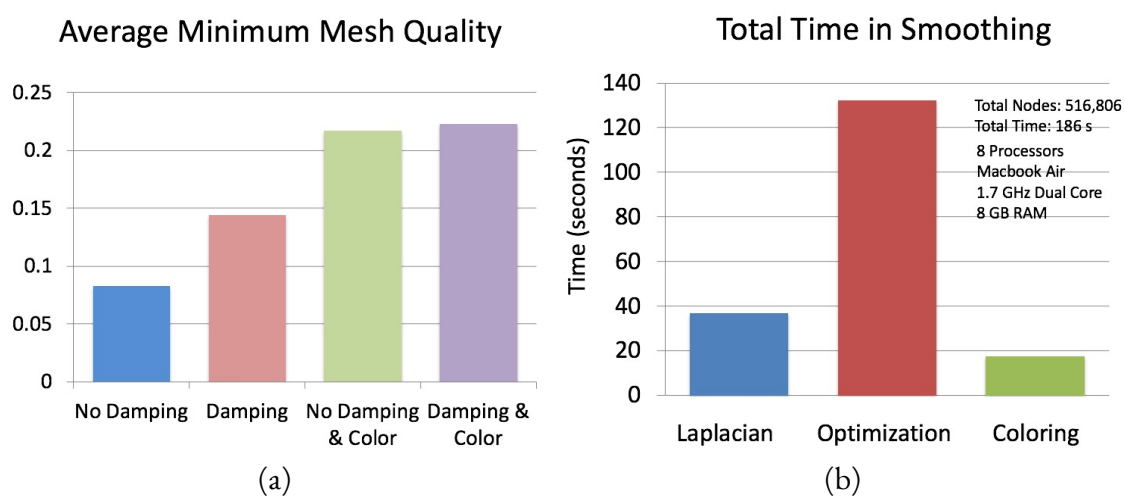


Figure 2-39. (a) Comparison of average minimum mesh quality using different smoothing options. (b) Time Comparison for different smoothing methods

We also illustrate in figure 2-39(b) the relative performance of the different smoothing methods on the test models. The tests measured the total cumulative clock time taken in each smoothing procedure to execute all 52 models. We observed that the Jacobi-based optimization procedures took the majority of time with the parallel coloring procedure taking the least. We note that modifying default thresholds for optimization and parallel coloring will change the performance metrics, and some trial and error is worthwhile to increase overall mesh quality.

3. EXAMPLES

For purposes of this study we present a few representative examples of computational materials modeling research for which the proposed meshing methods have been successfully applied.

3.1. CRYSTAL PLASTICITY

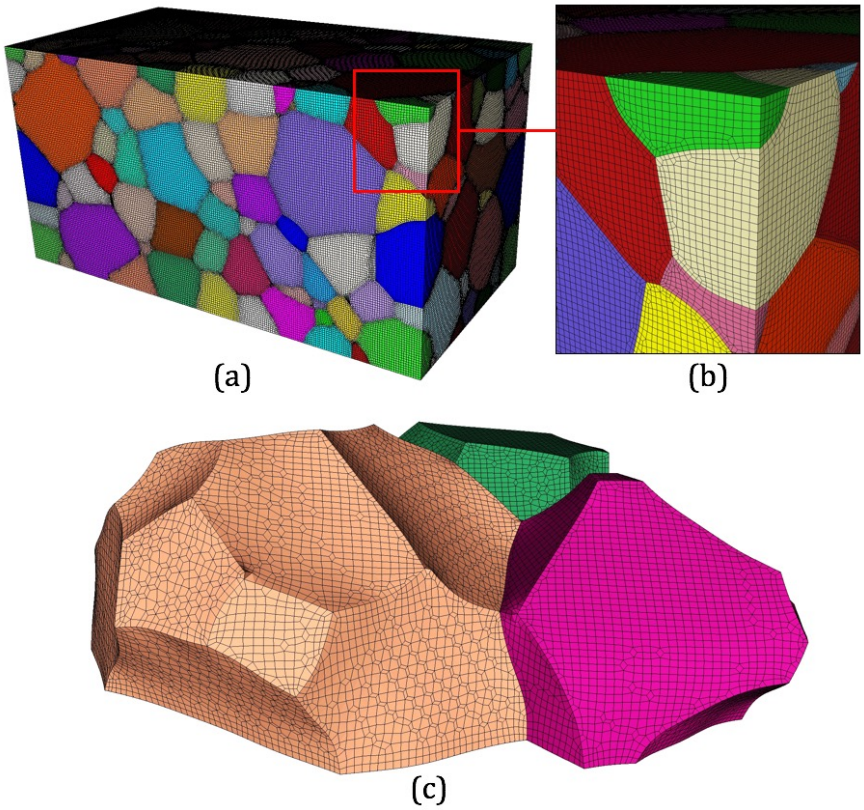


Figure 3-1. FE model containing 143 grains used for crystal plasticity analysis generated from voxelated data. (a) Full FEA mesh generated from 200x100x100 grid, (b) close-up of mesh, (c) view of three grains in FEA mesh.

The proposed methodology provides accurate representation and discretization of microstructural features, such as grains and grain boundaries [18]. Meso-scale computational models that incorporate these microstructural features such as crystal plasticity finite element (CP-FE), use a polycrystalline RVE

to model mechanical behaviors of grain aggregates. Most three-dimensional RVEs are generated by voxelated, or stair-step hexahedral finite elements which are difficult to accurately resolve curved grain boundaries and triple junctions.

Figure 3-1 shows the finite element discretization of 143 grains using an initial voxelization of $200 \times 100 \times 100$. Here, the three-dimensional microstructure is generated by the Monte Carlo Potts grain growth model in the Stochastic Parallel PARticle Kinetic Simulator (SPPARKS) developed at Sandia National Laboratories [12][34]. It is shown that the reconstructed hex mesh in figure 3-1 from SPPARKS simulations accurately reproduces smooth interfaces and junctions between grains. Lim et al. [18] showed that the FE mesh generated with the proposed methods is able to significantly reduce errors at grain boundaries and triple junctions in mechanical simulations when compared with an equivalent stair-step representation.

Figure 3-2 shows the FE meshes generated from experimental data created from electron backscatter diffraction (EBSD) measurement using a Scanning Electron Microscope (SEM) as illustrated in figure 3-2a. An EBSD technique characterizes microstructural features such as crystallography and phase of materials within a scanning grid. Crystallographic information from EBSD data is processed to have unique grain ids for each cell of a Cartesian grid that is used as input for the FE mesh generation procedure. Simulation results also showed that the smooth interfaces (figure 3-2c) were able to mitigate artificial stress localization commonly observed in stair-step interfaces.

A mesh generation example using Sculpt for generating a similar mesh shown in figures 3-2 and 3-1 is illustrated in Appending G. In this example, volume fraction data at each cell of a Cartesian grid is provided in a *.tex* file.

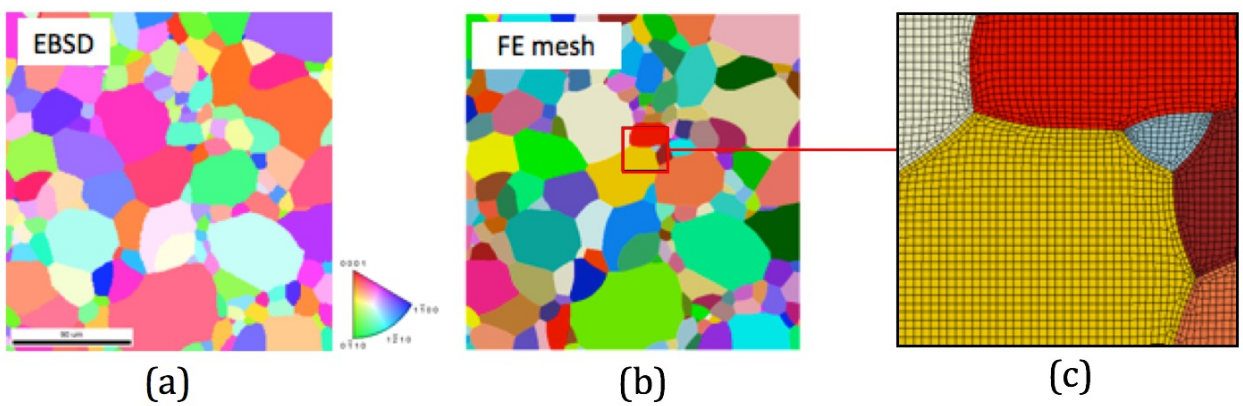


Figure 3-2. EBSD and FE discretizations of experimental data for modeling crystal plasticity.

3.2. SYNTACTIC FOAM MATERIALS

Syntactic foam materials, which consist of hollow spherical particles embedded in a matrix material, constitute another example of materials with complex microstructural features that present many challenges for 3-D meshing algorithms. Key features of the microstructure are the thin wall thickness of

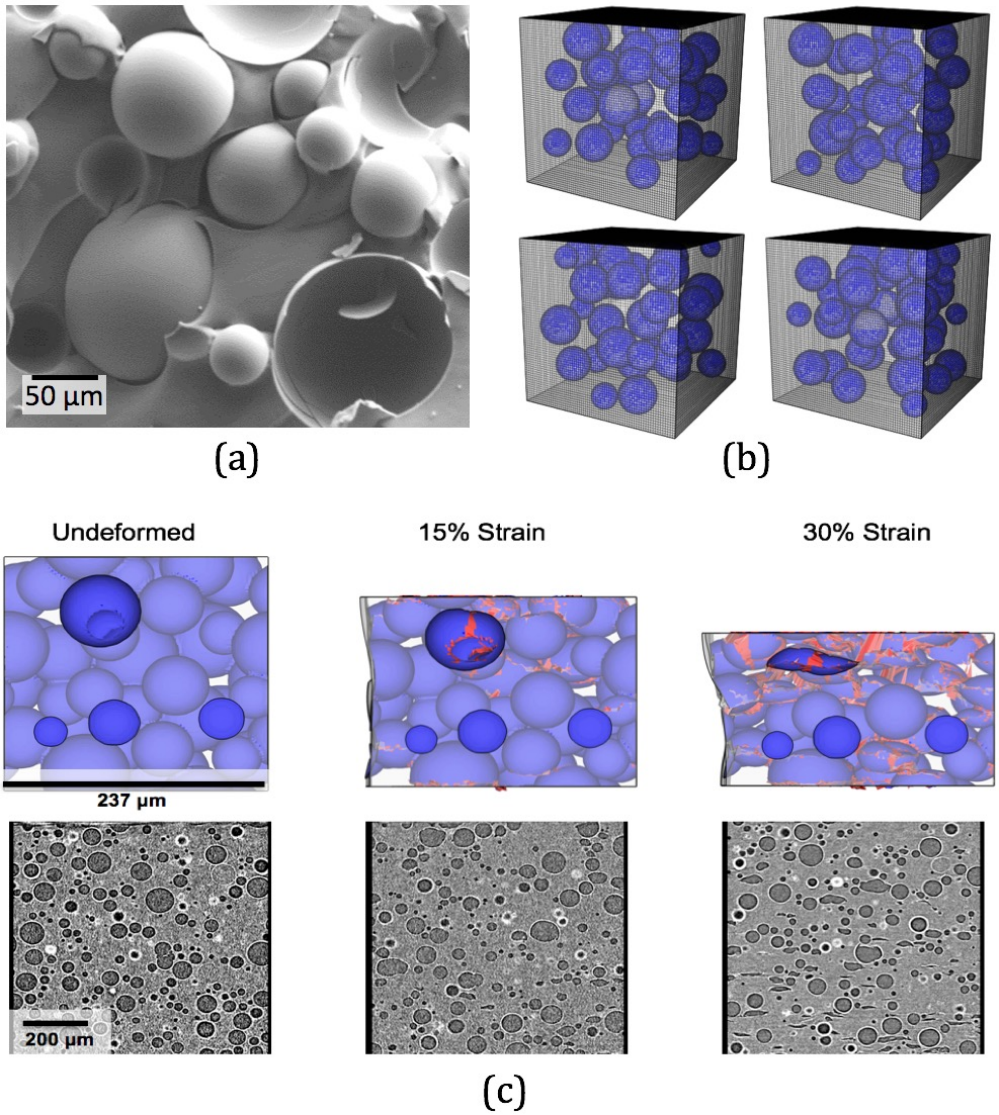


Figure 3-3. (a) SEM image of elastomeric syntactic foam microstructure showing narrow, irregular shaped regions in the matrix where many microballoons are clustered together. (b) Different SVE realizations of syntactic foam synthetic microstructures, (c) : Comparison of the large deformation uniaxial compression behavior of elastomeric syntactic foams for (Top) simulated synthetic microstructure with damaged elements shown in red and (Bottom) X-Ray Computed Tomography.

the hollow particles and irregularly shaped regions of the matrix where multiple particles are located close together. Such features are illustrated in Figure 3-3(a), which shows a SEM image of elastomeric syntactic foam filled with hollow glass microballoons

For synthetic microstructures based on this material, the matrix mesh must be fine enough to adequately resolve these irregular areas, and consist of hex elements to maintain a conformal mesh interface with

quadrilateral shell elements used to represent the hollow particles. This requires the matrix to have an element size smaller than the minimum spacing present between particles and can lead to large numbers of elements needed to mesh the entire SVE model unless meshing adaptivity features are enabled.

Additionally, many realizations of individual stochastic volume elements are needed for any analysis to capture the stochastic effects of variability in particle spacing and arrangement. This requires a meshing algorithm which can robustly generate hexahedral meshes for many different realizations of synthetic microstructure geometry without requiring any user interactions such as manual geometry decomposition. Figure 3-3(b) shows several different SVE realizations of synthetic syntactic foam microstructures meshed with Sculpt. For this example, the shell elements used to represent the hollow microballoons are colored blue, and hexahedral matrix elements are grey. Each SVE realization in Figure 3-3(b) has a unique geometry with randomized microballoon locations that were generated by a separate algorithm and passed to Sculpt as analytic geometry inputs.

Element quality analysis of the resulting meshes showed minimum scaled Jacobian values of approximately 0.2 over realization of 100 different SVE models. The location of shell elements representing microballoons in the meshes were also compared with the input geometric sphere locations, and resulted in differences of less than 0.2% when the background element size was 25 times less than the sphere radius.

This capability to rapidly generate and mesh synthetic microstructures with physically representative features enables high-fidelity microstructural simulations that can be used to study complex behaviors in these materials. Figure 3-3(c) shows a comparison of balloon breakage in elastomeric syntactic foams with increasing strain.

Mesh generation for this application is illustrated in Appendix K. In this case, the geometry is represented by a a set of analytic sphere definitions described in a *.diatom* file.

3.3. ENERGETIC MATERIALS

This example, illustrated in figure 3-4, demonstrates mesoscale modeling to represent thermal damage evolution in an ammonium perchlorate (AP) based rocket propellant. Studies have shown that exposure to elevated temperature produces thermally induced mechanical damage, including connected porosity. It is believed that this connected porosity can allow rapid flame spread and subsequent violent reaction in an accidental cook-off scenario (munition in a fire). Sculpt was used to generate a conforming all-hex mesh of a collection of AP particles within a polymer binder to simulate behavior at the mesoscale. A temperature-dependent reaction model was used to determine gas generation rates from the surface of the particles, which induces a local pressure at the particle-binder interfaces. A viscoelastic constitutive model was used for the polymer binder and a cohesive zone model was used to represent the bonded interface between each particle and the binder. During the simulation, void spaces around the particles grew asymmetrically, based on variations of the local stress state resulting from proximity of neighboring particles. Two-dimensional slices of the model showed crescent-shaped voids which are similar to those observed by X-Ray Computed Tomography (CT) scans of thermally damaged AP propellant samples. These techniques should be applicable to other materials such as plastic-bonded explosives as well.

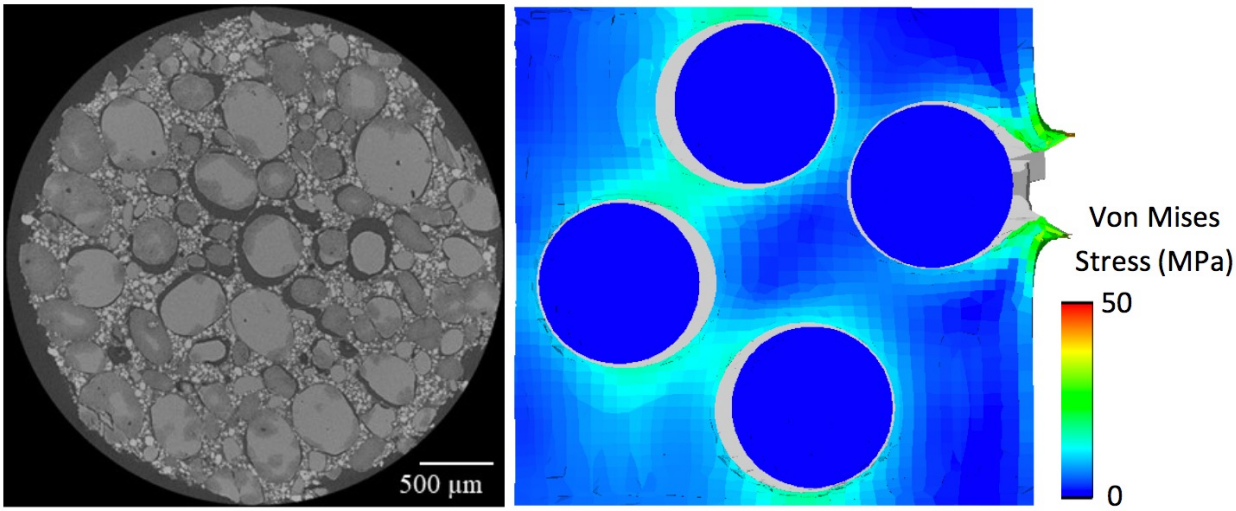


Figure 3-4. (Left) X-Ray Computed Tomography (CT) scan of a sample of ammonium perchlorate (AP) propellant which had been held at 215ÂřC for 2 hours. Crescent-shaped voids (dark gray) surround some of the large round AP particles. (Right) 2-D slice from interior of the 3-D simulation of AP particles surrounded by a polymer binder. A temperature-depended chemical decomposition reaction led to production of gases from the particles which induces local pressures and leads to void formation. Crescent-shaped voids occur in this simulation due to preferential gas expansion into regions with lower stress.

Mesh generation for this example is also illustrated in Appendix K where sphere definitions are represented analytically.

3.4. POLYMER MATERIALS

Figure 3-5 is an RVE of an idealized carbon-black polymer from an automotive tire. Sculpt was used to generate an adaptive mesh of 1300 numerically sized and located spheres. Similar to the previous examples, polymer materials may be represented using a similar mesh generation procedure. To generate geometry for this example, Sandia's LAMMPS [29] tool was used. LAMMPS is a molecular dynamics code that models ensembles of particles in a liquid, solid or gaseous state, the result of which, in this example, is a list of sphere centers and radii. Appendix K illustrates the procedure used for generating the mesh from analytic sphere data as well as the refinement operation illustrated in figure 3-5.

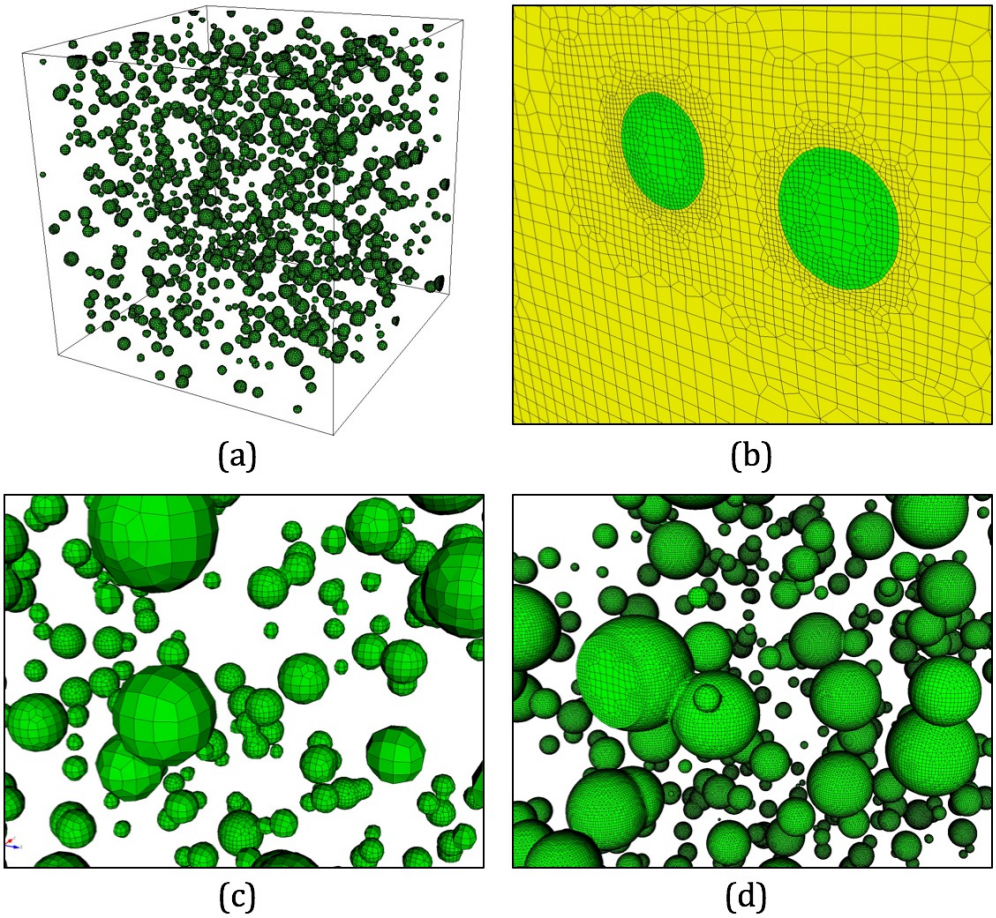


Figure 3-5. RVE model of an idealized carbon-black polymer from an automotive tire. It uses 1300 numerically sized and located spheres from analytic data. (a) View of FE mesh of spheres. (b) closeup demonstrating 4 levels of adaptive mesh at RVE boundary (c) View of spheres meshed with 2 adaptive levels (d) View of spheres meshed with 4 adaptive levels. Model courtesy of Goodyear Tire Company.

3.5. SPRAY-FORMED MATERIAL

Advances in thermal-spraying technologies present new opportunities for developing tailored, high-performance materials for a variety of applications through thermal-spray additive manufacturing. Their stochastic microstructure presents challenges for conformal mesh generation. Grain anisotropies, non-uniform pore distributions and interfacial roughness in multiphase materials create complex geometries that can be difficult to resolve. In this example, simulation is targeted at examining the effects of material microstructure on shock generation, propagation and material failure in a subset of spray-formed materials. Geometry for these examples comes from radiographic or metallographic imaging of material specimens. Two example meshes are illustrated in figure 3-6.

A mesh study including 40 different data sets generated with stochastic methods was performed and

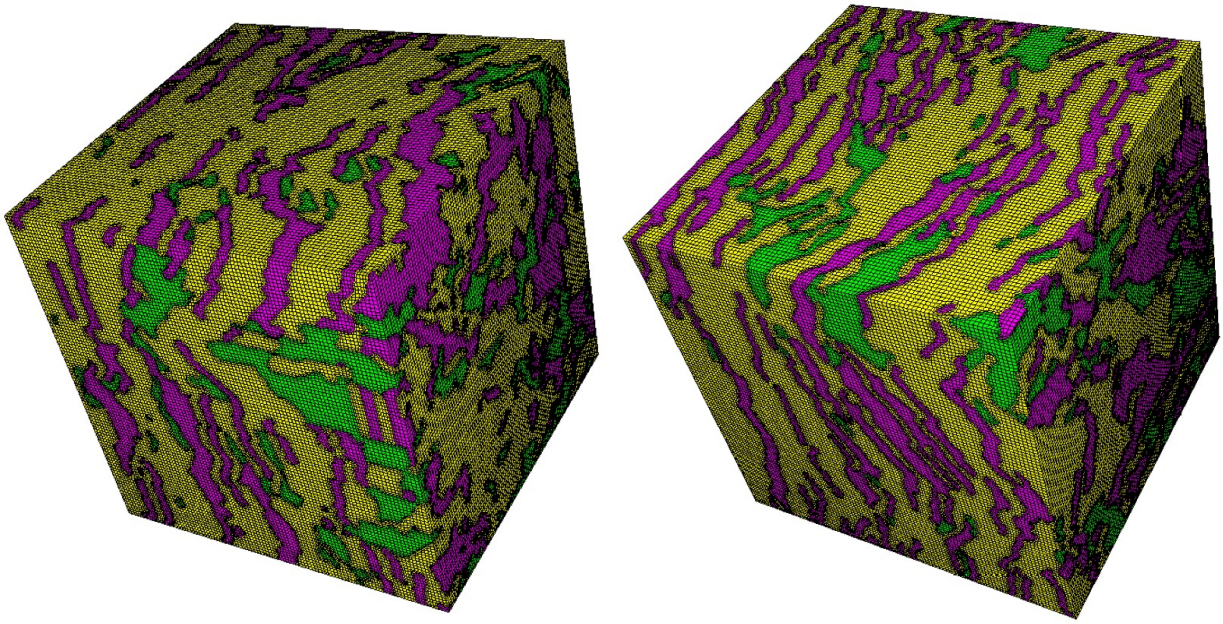


Figure 3-6. Examples of stochastic RVE meshes

results included in section 4.1 of this document. Appendix M also includes an example Sculpt input file for generating the meshes illustrated here.

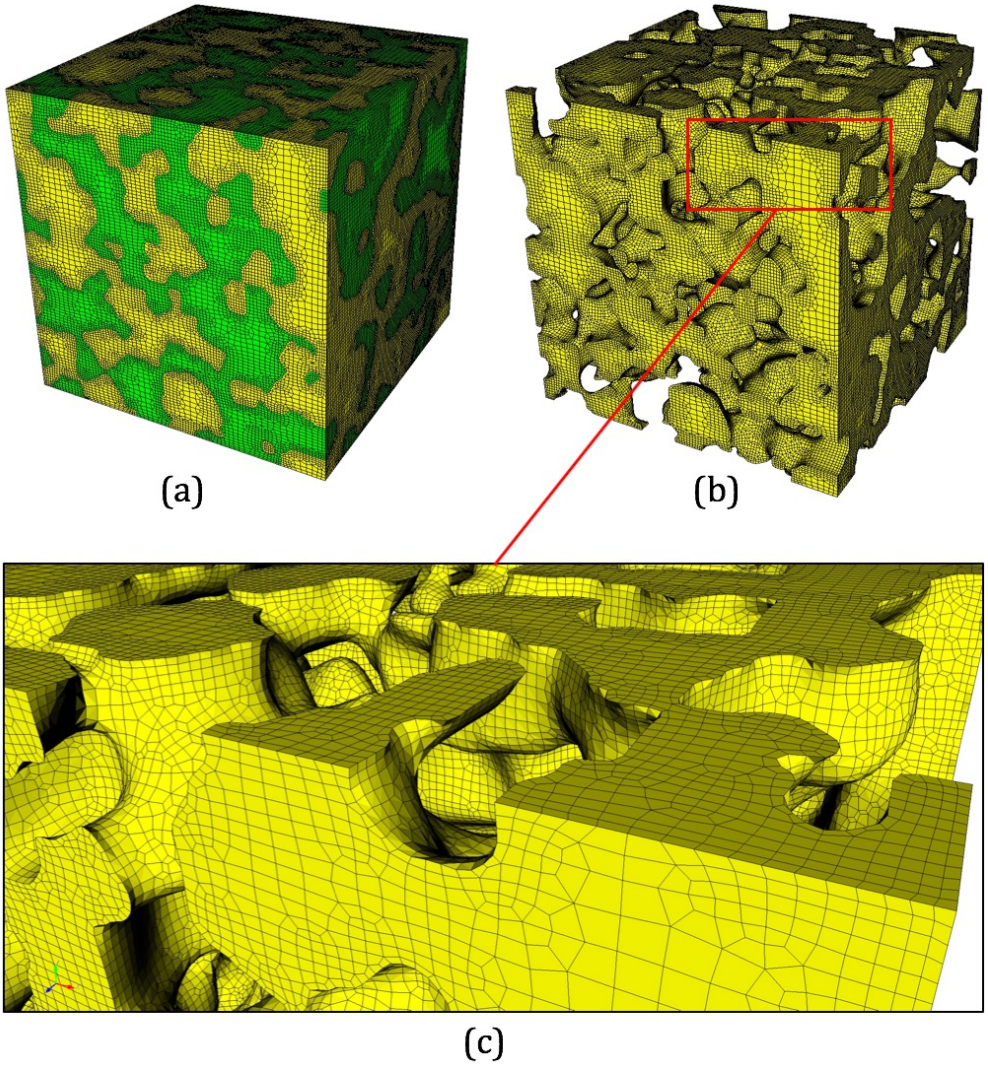


Figure 3-7. Adaptive RVE model of two-phase experimental polycrystalline microstructure generated with DREAM.3D [2] software. (a) View of both phases. (b) View of single phase. (c) Close-up of mesh of single phase illustrating smooth interior surfaces and adaptivity.

4. PERFORMANCE

We illustrate performance details from the proposed methods on several relevant RVE models in table 4-1. This table provide details from seven separate runs from models illustrated in figures 3-1 *(grains)*, 3-5 *(spheres)* and 3-7 *(2-phase)*. The columns are defined as follows:

1. Name of test case
2. *Fig:* Figure reference from this document
3. *num mats:* Total number of different materials represented in the input data
4. *num hexes*: Resulting number of hex elements in mesh
5. *min S.J.*: Minimum scaled Jacobian metric for any single hex in the mesh
6. *num*< 0.0*:* Number of hexes with scaled Jacobian less than 0.0
7. *num*< 0.2*:* Number of hexes with scaled Jacobian less than 0.2
8. *adapt levels:* Maximum number of levels of uniform 2-refinement applied (see sec. 2.5)
9. *num procs:* number of processors case was run on
10. *time:* CPU time taken for total mesh generation process on indicated number of processors.

Note that meshes with negative Jacobian elements are not intended to be used for analysis. They are documented here only to contrast the effect of using the *expand* and *defeature* options.

The first example, cases 1 and 2 (*grains*), is a crystal plasticity model illustrated in figure 3-1. It shows results both with (case 1) and without (case 2) the expansion layer capability described in section 2.7.5.

		Fig.	num mats	num hexes	min S.J.	num < 0.0	num < 0.2	adapt levels	num procs	time (sec.)
1	grains w/ expand	3-1	143	3,655,878	0.0749	0	73	0	12	201.2
2	grains w/o expand	3-1	143	3,359,218	-0.114*	1	142	0	12	223.9
3	spheres L1	3-5	2	3,614,934	0.1689	0	13	1	128	33.48
4	spheres L2	3-5	2	14,105,616	0.1334	0	21	2	128	192.4
5	spheres L3	3-5	2	50,278,564	0.1365	0	20	3	128	1015.4
6	2-phase w/ defeature	3-7	2	1,412,340	0.1211	0	11	1	8	181.2
7	2-phase w/o defeature	3-7	2	1,454,870	-0.949*	48	781	1	8	141.1

Table 4-1. Example cases from previous figures used to illustrate Sculpt performance

We note that without the expansion layer, exactly one negative Jacobian element is present making the mesh unusable. The addition of the expansion layer improves the mesh quality so the mesh is now computable with minimum Scaled Jacobian 0.075.

The second example, cases 3-5 (*spheres*), illustrated in figure 3-5, is an RVE model of an idealized carbon-black polymer used to represent the microstructure of an automotive tire. This example uses 1300 numerically placed and sized spheres that have been adaptively and conformally meshed within a separate matrix material. Here we demonstrate three different meshes using one, two and three adaptive levels respectively on 128 processors and note the algorithms' ability to scale to over 50 million hexes in about 17 minutes on 128 processors.

Finally, cases 6-7 demonstrate the effect of the defeature option described in section 2.7.2 with the *2-phase* model illustrated in figure 3-7. We note that without defeaturing (case 7) we are unable to achieve usable quality elements on a small number of elements. However including the defeature option (case 6), while increasing the compute time a modest amount, improves the mesh quality sufficiently to allow for FEA analysis. In addition, we note that the coarsening feature, described in section 2.5 is also illustrated in this model.

4.1. SPRAY-FORMED MATERIAL MESH STUDY

A series of 40 test cases were used to characterize Sculpt's capabilities to generate an all-hex mesh on highly stochastic input data. For this study, input was provided as pure cells (*.spn* file) on a Cartesian grid of resolution $100 \times 100 \times 100$ representing grain structures from radiographic or metallographic imaging of material specimens. Each of these specimens contained three different materials, including one material (*block 1*) representing pore space. Eight different primary configurations of input data were used with 5 variations of each, for a total of 40 models. Figures 4-2 and 4-3 show mesh images of the first variation from each category. Table 4.1 shows the details from each of the 40 meshes

Due to the stochastic nature of the data, we utilized the defeaturing and thickening capabilities described in section 2.7.2 to improve mesh quality. We also used the smoothing methods described in section 2.12.6.1 to improve material grain interface smoothness as well as the curve optimization threshold, C_{OT} described in section 2.12.6.2 to improve quality at curve interfaces.

Figures 4-2 and 4-3 show three different images from each input data set. Colors represent the three different materials present in the model where *block 1 = green (pore space), block 2 = yellow* and *block 3 = magenta*. The image on the left shows one slice of the raw $100 \times 100 \times 100$ Cartesian cells colored according to their assigned material. The middle images show the final mesh of the same slice after thickening, defeaturing, non-manifold resolution, pillowing and smoothing procedures have been performed. The image on the right is an isometric view of the RVE with a cutting plane to illustrate the interior structure of the model.

Table 4.1 shows the details from the 40 meshes generated to simulate spray-formed materials. The following describes the data from each column of table 4.1.

1. *case name*: Eight categories of configurations with five variations of each for a total of 40 models

2. *num hexes*: Final number of hexes in RVE. Each model begins with one million cells from raw input data (*.spn* file). Additional hexes are created at interfaces due to pillowing (see sec. 2.11.2)

3. *min S.J.*: Minimum scaled Jacobian in final mesh for any single hex element. (see sec. 2.12.1). Note that average *S.J.* is greater than 0.8 for all models. Minimum *S.J.* typically controls whether the model is useful for analysis, where *S.J.*< 0.0 is generally considered unusable.

4. *num* < 0.0: Number of hex elements with scaled Jacobian < 0.0. These meshes would be considered unusable in their current state.

5. *num* < 0.2: Number of hex elements with scaled Jacobian < 0.2. Some elements with *S.J.*< 0.2 are expected. Minimizing this number is advantageous.

6. *percent error*: The percent error by volume for each material based on the difference between the raw input data to the final hexahedral mesh. (see sec. 2.7.6) This value is generally affected by defeauturing and thickening. A percent error near zero is desirable for most cases.

7. *time (sec.)*: CPU time taken to generate the mesh on 12 processors.

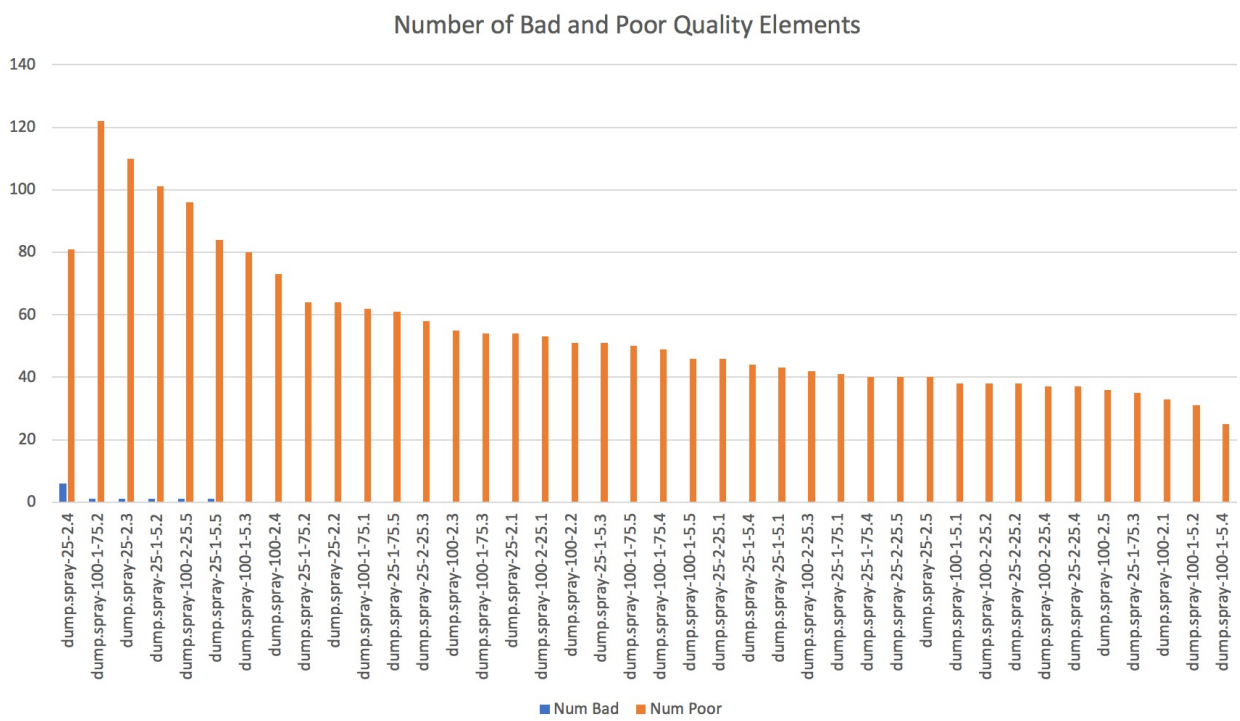


Figure 4-1. Spray-formed material meshes ordered by bad and poor quality elements.

Figure 4-1 illustrates the number of bad and poor quality elements for each of the 40 meshes ordered based on their total bad and poor quality elements. Note that *bad* elements are those with *S.J.* < 0.0 and *poor* are those with *S.J.* < 0.2. From this data we point out that six of the 40 meshes resulted in at least one bad element, representing an 85% success rate. In practice, these six meshes would be discarded and not used in a simulation. For this application, data can readily be generated using stochastic

methods. As a result, this meshing success rate is sufficient to provide a representative sample of the material characteristics.

We also point out that the meshing outcome will be significantly affected by modifying defeaturing and thickening parameters. For this application we found that small clusters of isolated cells, as well as narrow protrusions for localized material regions, resulted in unacceptable meshing results without using the defeaturing option. However, application of defeaturing in this case resulted in significant volumetric error compared with the initial input data. Figure 4-4(a) shows the initial raw cell assignment for block 1 on a small slice of representative input data. Compare with Figure 4-4(b) that illustrates the effect of defeaturing on the material assignment. As intended, the small clusters and protrusions have been removed, resulting in the mesh shown in 4-4(c). This mesh however has a volumetric percent error of 22% when compared with the volume of block 1 material in the input. To correct for this, the thickening option is used, which adds material to the boundaries of block 1. A trial-and-error approach is used to reduce the volumetric error. Figures 4-4(d) - (f) show mesh results using different thickening parameters. In this case, we settled on thickening=0.35 (figure 4-4(f)) which resulted in a volumetric percent error for block 1 of -1.7%.

Note that the volumetric percent error for each material block is reported in table 4.1. For consistency, A constant set of thickening values was used for all 40 mesh cases. (*block 1 = 0.33, block 2 = 0, block 3 = 0.2*). Because of the widely varying characteristics of the input data, significant differences are noted in the volumetric error between cases. In practice, we would recommend an approach to minimize volumetric error by adjusting thickening parameters for individual mesh cases.

Figure 4-4(g) shows an overlay of the initial raw input cell material assignment for block 1 displayed with the final mesh where thickening has been set to 0.35. We point out that judgement should be employed in determining whether the final mesh, that has been significantly adjusted and simplified to control element quality, is indeed representative of the intended physical characteristics to be modeled.

Appendix M provides an example of an input file and data for the mesh study case, `dump.spray-100-1-5.1`, in this study.

Table 4-2. Spray-formed materials mesh study results

case name	num hexes	min S.J.	num < 0.0	num < 0.2	percent error block 1	block 2	block 3	time (sec.)
dump.spray-100-1-5.1	1,629,018	0.144	0	38	-1.36	1.15	-2.47	99.82
dump.spray-100-1-5.2	1,656,084	0.119	0	31	-6.00	1.74	-1.07	91.96
dump.spray-100-1-5.3	1,761,378	0.095	0	80	-0.75	1.29	-2.32	113.98
dump.spray-100-1-5.4	1,693,294	0.148	0	25	4.36	-0.83	-1.29	96.85
dump.spray-100-1-5.5	1,682,310	0.110	0	46	4.66	-0.57	-2.68	106.03
dump.spray-100-1-75.1	1,714,808	0.099	0	62	1.36	-0.39	-0.01	94.39
dump.spray-100-1-75.2	1,796,816	-0.022	1	122	-15.89	1.23	2.58	100.88
dump.spray-100-1-75.3	1,828,360	0.112	0	54	-7.90	1.92	-0.40	112.98
dump.spray-100-1-75.4	1,749,288	0.112	0	49	2.08	-1.68	1.40	108.74
dump.spray-100-1-75.5	1,741,482	0.126	0	50	-4.80	1.34	-0.15	96.88
dump.spray-100-2-25.1	1,758,634	0.128	0	53	3.32	-0.58	-1.62	101.96
dump.spray-100-2-25.2	1,781,898	0.091	0	38	-9.60	1.11	0.97	99.46
dump.spray-100-2-25.3	1,793,948	0.111	0	42	-13.83	2.40	0.05	107.03
dump.spray-100-2-25.4	1,760,298	0.134	0	37	-23.54	1.91	1.34	97.63
dump.spray-100-2-25.5	1,747,262	-0.002	1	96	-16.22	1.85	1.32	94.54
dump.spray-100-2.1	1,743,950	0.118	0	33	6.75	-1.29	-2.34	120.86
dump.spray-100-2.2	1,726,572	0.129	0	51	4.50	-1.17	-2.30	109.14
dump.spray-100-2.3	1,743,946	0.133	0	55	3.78	0.81	-4.31	113.06
dump.spray-100-2.4	1,766,348	0.117	0	73	-1.45	-1.48	2.94	97.02
dump.spray-100-2.5	1,694,232	0.111	0	36	-3.08	1.18	-1.18	101.07
dump.spray-25-1-5.1	1,723,080	0.082	0	43	-12.51	2.22	-0.49	108.13
dump.spray-25-1-5.2	1,666,462	-0.228	1	101	-19.13	2.74	-0.61	102.81
dump.spray-25-1-5.3	1,763,740	0.111	0	51	2.63	-0.14	-1.29	105.55
dump.spray-25-1-5.4	1,716,754	-0.007	0	44	-9.79	1.50	0.31	104.13
dump.spray-25-1-5.5	1,760,418	0.131	1	84	-3.57	-0.24	1.85	106.56
dump.spray-25-1-75.1	1,817,064	0.133	0	41	-20.37	3.02	0.20	120.49
dump.spray-25-1-75.2	1,768,866	0.125	0	64	-30.17	2.22	2.34	132.07
dump.spray-25-1-75.3	1,763,262	0.115	0	35	-35.08	2.33	2.88	106.01
dump.spray-25-1-75.4	1,748,762	0.119	0	40	-29.17	2.34	1.58	103.87
dump.spray-25-1-75.5	1,822,654	0.114	0	61	-15.39	2.35	0.24	108.20
dump.spray-25-2-25.1	1,789,410	0.120	0	46	-44.28	3.10	1.88	93.59
dump.spray-25-2-25.2	1,777,166	0.140	0	38	-41.37	2.93	1.45	108.44
dump.spray-25-2-25.3	1,730,820	0.119	0	58	-31.57	3.53	0.03	95.69
dump.spray-25-2-25.4	1,760,298	0.134	0	37	-23.54	1.91	1.34	99.53
dump.spray-25-2-25.5	1,803,822	0.130	0	40	-28.94	1.62	2.94	103.06
dump.spray-25-2.1	1,746,548	0.115	0	54	-7.00	2.29	-2.09	104.00
dump.spray-25-2.2	1,738,320	0.113	0	64	-10.38	0.67	1.80	103.53
dump.spray-25-2.3	1,743,474	-0.149	1	110	3.89	-0.12	-2.73	111.31
dump.spray-25-2.4	1,783,298	-0.243	6	81	-26.73	2.10	2.56	101.86
dump.spray-25-2.5	1,733,680	0.114	0	40	-6.72	1.83	-0.79	103.63

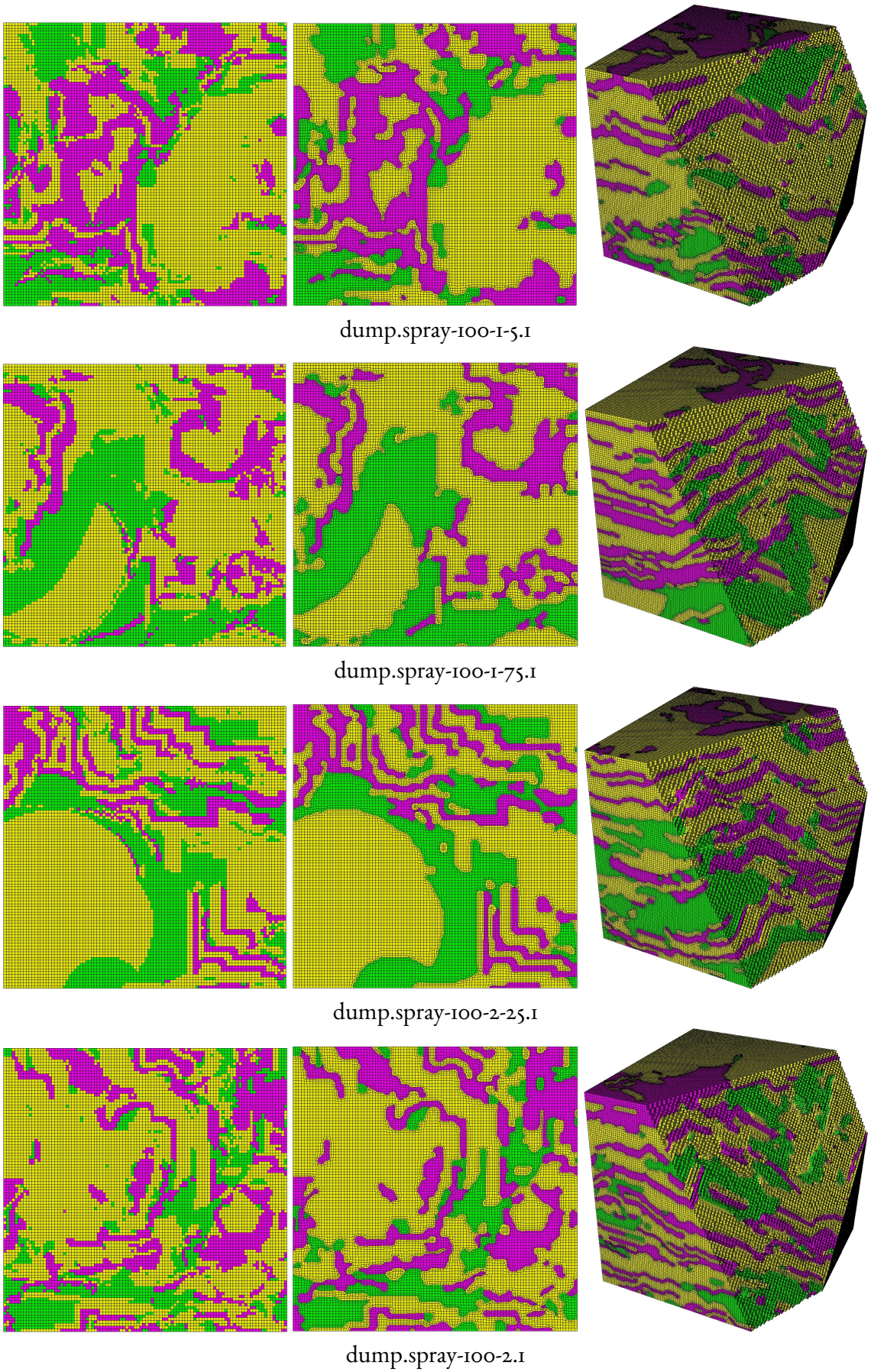


Figure 4-2. Examples of spray-formed material RVE meshes

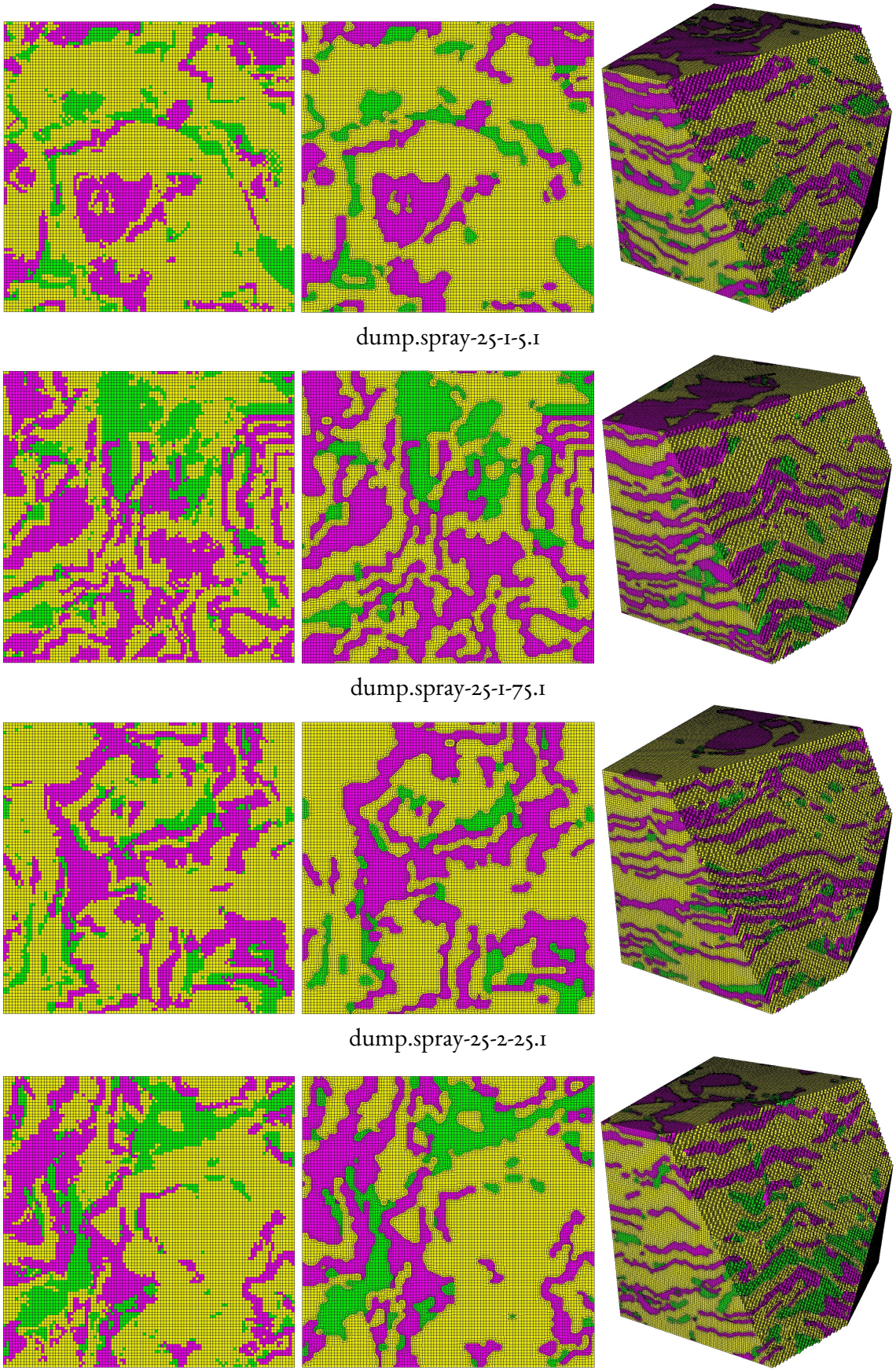
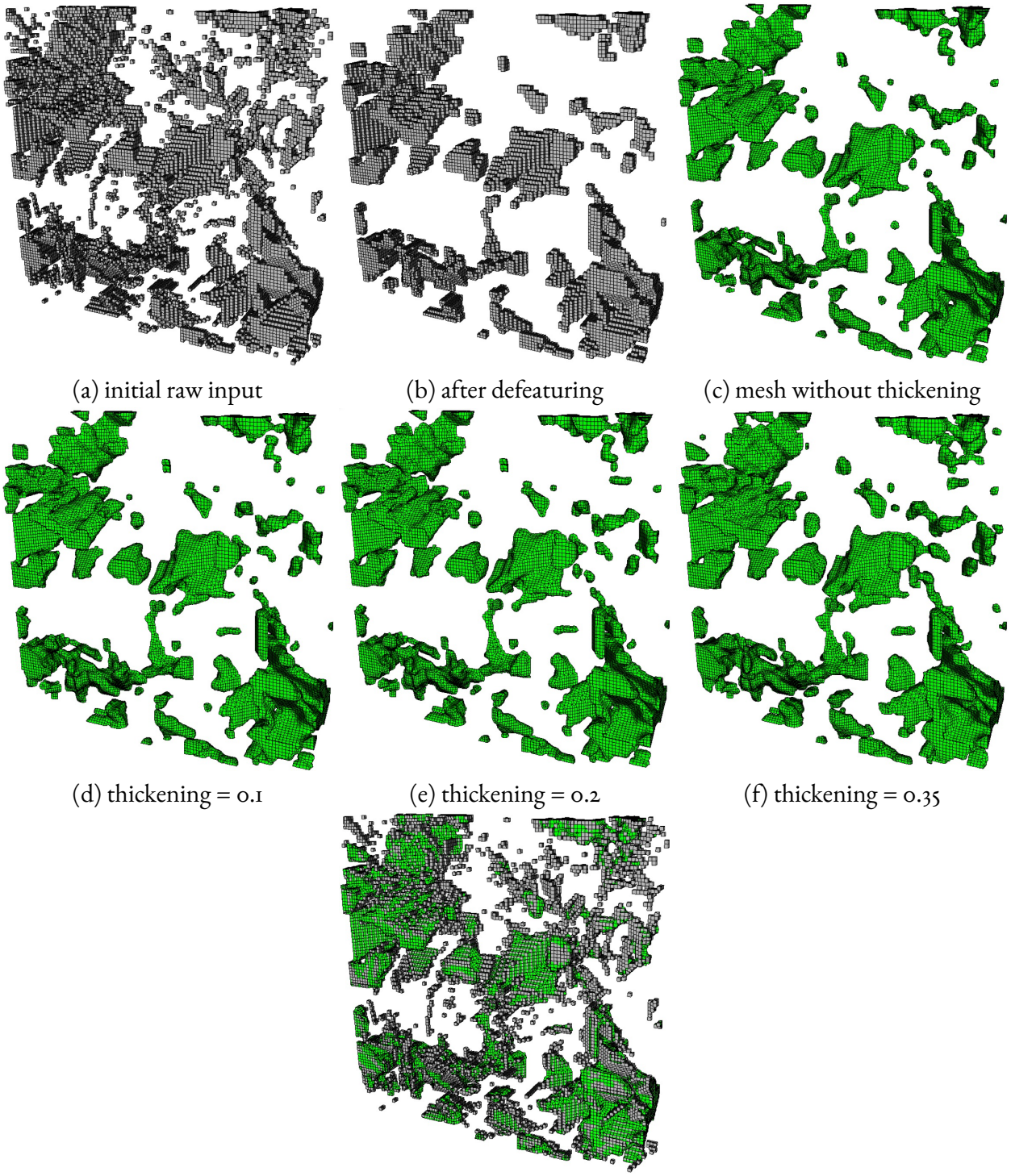


Figure 4-3. Examples of spray-formed material RVE meshes.



(a) initial raw input (b) after defeaturing (c) mesh without thickening

(d) thickening = 0.1 (e) thickening = 0.2 (f) thickening = 0.35

(g) overlay raw input with mesh (thickening=0.35)

Figure 4-4. A representative slice of one of the RVE models showing block 1 cells (pore space) only. Illustrates the effect of defeaturing and thickening on one material.

5. CONCLUSION

This work presents new methods for generating hexahedral and tetrahedral meshes for computational materials modeling. The proposed algorithms build upon overlay grid techniques previously introduced. This work contributes new innovations that have proven necessary for meshing data for modeling of representative volume elements. New methods have been introduced for data filtering, including generalized non-manifold resolution, defeaturing, thickening and layer expansion. These filtering methods were shown to directly influence mesh quality producing meshes that would otherwise contain inverted or poor quality elements. Improvements to the primal contouring method for constructing interfaces on a generalized unstructured grid with multiple materials were also introduced. A generalized procedure for improving hexahedral topology through pillowing at material interfaces was introduced and shown to be effective with the complex topologies needed to represent detailed grain structures. In addition, new methods for generating exact periodic meshes on RVE models were introduced. We have also shown that the methods introduced are directly applicable and an enabling capability for materials science research in a variety of applications.

REFERENCES

[1] Ansys inc. intro. to ansys icem cfd hexa. http://www.ansys.com/Services/training-center/platform/introduction-to-ansys-icem-cfd-Hexa. Accessed: 2019-05-17.

[2] Dream.3d. http://dream3d.bluequartz.net. Accessed: 2019-05-17.

[3] Pointwise structured grid generation. http://www.pointwise.com. Accessed: 2019-05-17.

[4] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin. A comparison of parallel graph coloring algorithms, 1995.

[5] Erik Boman, Karen Devine, Lee Ann Fisk, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, and William Mitchell. Zoltan home page. http://www.cs.sandia.gov/Zoltan, 1999.

[6] Scott A. Canann, Joseph R. Tristano, and Matthew L. Staten. An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. *Proceedings of the 7th International Meshing Roundtable*, pages 479–494, 1998.

[7] D. A. Crawford, A. L. Brundage, E. N. Harstad, K. Ruggirellonand G. Schmitt, S. C. Schumacher, and J. S. Simmons. Cth user manual and input instructions, version 10.3. Technical report, Sandia National Laboratories, Albuquerque, NM, 2 2013.

[8] Distene, S.A.S. An introduction to meshgems-cadsurf v1.0, a fast, robust, high quality cad surface mesher. marketing document, Bruyères le ChaÌĆtel France, 2019. http://www.meshgems.com.

[9] Distene, S.A.S. Volume meshing: Meshgems-tetra. website, Bruyères le ChaÌĆtel France, 2019. http://www.meshgems.com.

[10] Lori A. Freitag, M. Jones, and Paul E. Plassmann. An efficient parallel algorithm for mesh smoothing. *Proceedings 4th International Meshing Roundtable*, pages 47–58, 1995.

[11] David M. Hensinger, Richard R. Drake, James G. Foucar, and Thomas A. Gardiner. Pamgen, a library for parallel generation of simple finite element meshes. Technical Report SAND2008-1933, Sandia National Laboratories, Albuquerque, NM, 4 2008.

[12] E. A. Holm and C. C. Battaile. The computer simulation of microstructural evolution. *The Journal of The Minerals, Metals & Materials Society*, 53:20–23, 2001.

[13] Norman L. Jones. Solid modeling of earth masses for applications in geotechnical engineering. *PhD. Thesis, University of Texas, Austin*, 1990.

[14] K. K. Matous, M.G.D. Geers, V.G. Kouznetsova, and A. Gillman. A review of predictive nonlinear theories for multiscale modeling of heterogeneous materials. *Journal of Computational Physics*, 330:192–220, 2017.

[15] Patrick M. Knupp. 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, 2000.

[16] Patrick M. Knupp. Hexahedral and tetrahedral mesh untangling. *Engineering With Computers*, 17:261–268, 2001.

[17] Patrick M. Knupp. A method for hexahedral mesh shape optimization. *International Journal for Numerical Methods in Engineering*, 58:319–332, 2003.

[18] Hojun Lim, Fadi F. Abdeljawad, Steven. J. Owen, Byron. W. Hanks, James. W. Foulk III, and Corbett. C. Battaile. Incorporating physically-based microstructures in materials modeling: Bridging phase field and crystal plasticity frameworks. *Modeling and Simulation Materials Science Engineering*, 24, 2016.

[19] Steven J. Owen. Parallel smoothing for grid-based methods. *21st International Meshing Roundtable, Research Note*, 2012.

[20] Steven J. Owen, Judith A. Brown, Corey D. Ernst, Hojun Lim, and Kevin N. Long. Hexahedral mesh generation for computational materials modeling. *Procedia Engineering, Proceedings 26th International Meshing Roundtable*, 203:167–179, 2017.

[21] Steven J. Owen, Corey D. Ernst, and Clinton J. Stimpson. Sculpt: Automatic parallel hexahedral mesh generation. Technical Report SAND2019-6412, Sandia National Laboratories, Albuquerque, NM, 5 2019.

[22] Steven J. Owen and Tim R. Shelton. Validation of grid-based hex meshes with computational solid mechanics. *Proceedings 22nd International Meshing Roundtable*, pages 39–56, 2013.

[23] Steven J. Owen and Tim R. Shelton. Evaluation of grid-based hex meshes for solid mechanics. *Engineering with Computers*, 31(3):529–543, 2015.

[24] Steven J. Owen and Ryan M. Shih. A template-based approach for parallel hexahedral two-refinement. *Procedia Engineering, Proceedings 24th International Meshing Roundtable*, 124:31–43, 2015.

[25] Steven J. Owen and Ryan M. Shih. A template-based approach for parallel hexahedral two-refinement. *Computer-Aided Design*, 85(C):34–52, 2017.

[26] Steven J. Owen, Matthew L. Staten, and Marguerite C. Sorensen. Parallel hex meshing from volume fractions. *Proceedings of the 20th International Meshing Roundtable*, pages 161–178, 2011.

[27] Steven J. Owen, Matthew L. Staten, and Marguerite C. Sorensen. Parallel hex meshing from volume fractions. *Engineering with Computers*, 30(3):301–313, 2014.

[28] Steven J. Owen and David R. White. Mesh-based geometry: A systematic approach to constructing geometry from a finite element mesh. *Proceedings, 10th International Meshing Roundtable*, pages 83–96, 2001.

[29] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics, 1995. `http://lammps.sandia.gov`.

[30] T. M. Pollock and J.E. Allison. Integrated computational materials engineering: A transformational discipline for improved competitiveness and national security. *National Materials Advisory Board, NAE, National Academies Press, Report Number: ISBN-10: 0-309-11999-5.*, 2008.

[31] Jin Qian and Yongjie Zhang. Automatic unstructured all-hexahedral mesh generation from b-reps for non-manifold cad assemblies. *Engineering with Computers*, 28(4):345âĂŞ–359, 2012.

[32] Jin Qian, Yongjie Zhang, and Wenyan Wang. Quality improvement of non-manifold hexahedral meshes for critical feature determination of microstructure materials. *Proceedings 18th International Meshing Roundtable*, pages 211–230, 2009.

[33] W. Roshan Quadros, Steven J. Owen, Matthew L. Staten, Byron W. Hanks, Corey D. Ernst, Clinton J. Stimpson, Ray J. Meyers, and Randy Morris. Cubit geometry and meshing toolkit, version 15.4. Technical Report SAND2019-3478, Sandia National Laboratories, Albuquerque, NM, 4 2019. `https://cubit.sandia.gov/public/15.4/help_manual/WebHelp/cubithelp.htm`.

[34] T. M. Rodgers, J. D. Madison, V. Tikare, and M. C. Maguire. Predicting mesoscale microstructural evolution in electron beam welding. *The Journal of The Minerals, Metals & Materials Society*, 68.

[35] Gregory D. Sjaardema. Overview of the sandia national laboratories engineering analysis code access system (seacas). Technical Report SAND92-2292, Sandia National Laboratories, Albuquerque, NM, 1 1993. `https://github.com/gsjaardema/seacas`.

[36] J. Vollmer, Robert Mencl, and Heinrich Müller. Improved laplacian smoothing of noisy surface meshes. *Comput. Graph. Forum*, 18:131–138, 1999.

[37] Jun Wang and Zeyun Yu. Quality mesh smoothing via local surface fitting and optimum projection. *Graphical models*, 73(4):127–139, 2011.

[38] Y. Zhang and C. L. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering*, 195:942–960, 2006.

[39] Yongjie J. Zhang. *Geometric Modeling and Mesh Generation from Scanned Images*. CRC Press, Taylor & Francis Group, Boca Raton, FL, 2016.

APPENDICES

F. SCULPT EXAMPLES

The following examples are intended to illustrate a few basic use cases for generating hex and tet meshes for microstructures. These examples assume that you will have access to both `sculpt` and `cubit` and that you can run `sculpt` command line from a unix prompt. For Sandia staff, both `sculpt` and `cubit` are available on the `CEE-LAN` at `/projects/cubit` or can be downloaded from `http://cubit.sandia.gov/downloads.html` with a Kerberos password. For external access, check the licensing page at `http://cubit.sandia.gov/licensing.html` to obtain a copy.

For more information on setup and running `sculpt`, see the Sculpt User Manual available online at `https://cubit.sandia.gov/public/15.4/help_manual/WebHelp/mesh_generation/meshing_schemes/parallel/sculpt.htm` or the SAND Report: *Sculpt: Automatic Parallel Hexahedral Mesh Generation*. Help on `sculpt` command line options are also available by using the `-h` argument followed by the sculpt option. For example:

```
sculpt -h pillow_surfaces
```

will display details of the `pillow_surfaces` option. To display a summary of all `sculpt` options, use: "`sculpt -h`" with no additional arguments.

All data files used in these examples can be downloaded from `cubit.sandia.gov/tutorials.html` from the `examples` link.

G. EXAMPLE: VOLUME FRACTION DATA

Representative volume elements (RVE) can often be defined in terms of volume fractions on the overlay grid. For this type of data, volume fractions for every material present in the RVE are included for each cell or element of the overlay grid. Two different formats are available for representing volume fractions.

1. `input_micro = <filename.tec>`: This option will import an ascii format file on a Cartesian grid. A simple example of the ascii `.tec` format is illustrated in figure G-1. Each cell of the grid occupies one row of the file where the first three columns identify the centroid of a cell, and the remaining columns define the volume fraction for each material present in the model. The sum of these columns should be 1.0. The required header information includes a title, variable names associated with each column and the zone which defines the number of cells in x, y and z Cartesian directions.

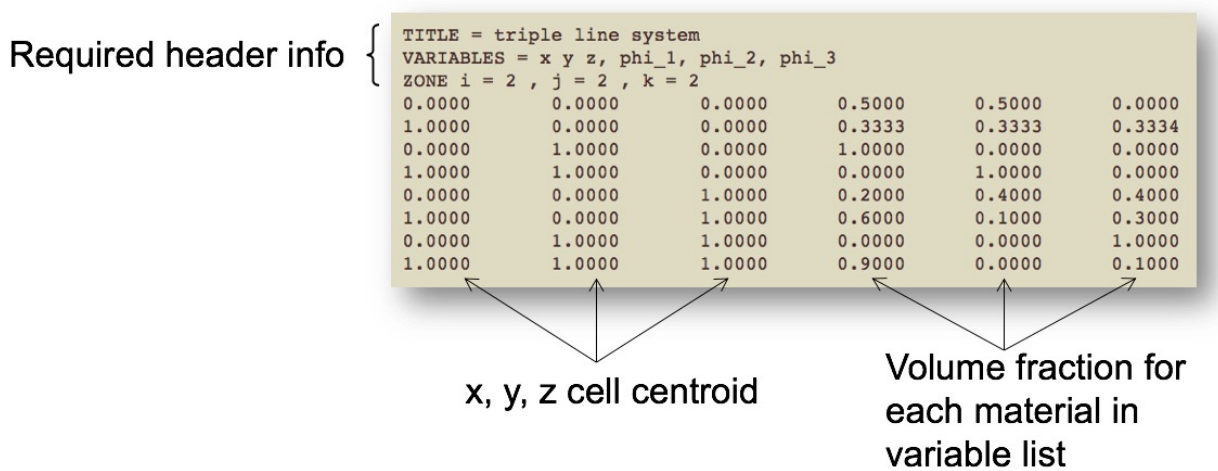


Figure G-1. Example `.tec` file describing volume fractions in each cell of a Cartesian grid

For the `input_micro` option, the size of each cell of the grid is assumed to be a unit cube with dimensions $1.0 \times 1.0 \times 1.0$. The locations identified in the first three columns (x, y, z) define the centroid location for each cell. For this format, the bounding box and interval specifiers (`xmin, ymin, zmin, xmax, ymax, zmax, nelx, nely, nely`) are not required in the input, but rather implied by the header information in the `.tec` file. To illustrate, the example

shown in figure G-1 would result in a grid with dimensions $2.0 \times 2.0 \times 2.0$ with coordinate dimensions ranging from `min = -0.5` to `max = 1.5`.

2. `input_mesh = <filename.exo>`: This option will import a binary exodus format file for any structured or unstructured mesh. For this option, element variables are required at each element of the exodus mesh. Each element should contain volume fractions for every material present in the mesh where the sum of volume fractions for each cell is 1.0. Although the exodus file format is binary, open source tools are available from `http://github.com` for reading and writing exodus files.

This example uses the file `micro.tec`, which defines a grid of $96 \times 96 \times 96$ cells with 20 different materials. Example files for this and all examples in this report are available from `http://cubit.sandia.gov/tutorials.html`.

Figure G-2 shows the resulting hex mesh that should be generated from `micro.tec`

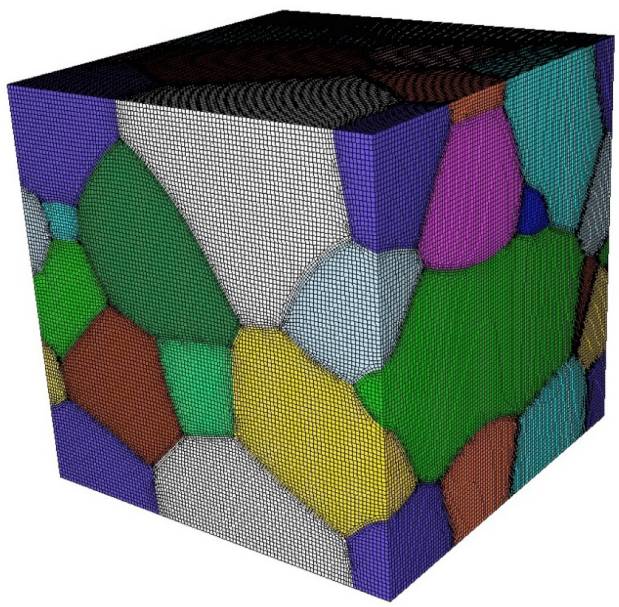


Figure G-2. Hex mesh generated from example `micro.tec`

The following Sculpt input file can be used for this example. Create a file with suggested name `micro.i`, and copy the following text into it:

```
Begin Sculpt
    input_micro = micro.tec
    exodus_file = micro
    micro_expand = 2
    pillow_surfaces = true
End Sculpt
```

To run sculpt with the designated input, make sure the file `micro.tec` is in the same directory and issue the following command at the unix prompt:

```
sculpt -i micro.i -j 8
```

This should execute Sculpt on 8 processors and export 8 separate exodus files to your working directory of the form `micro.e.8.x`, where `x` is `(0, 1, 2, ...7)`.

These files can be combined into a single file using the SEACAS `epu` tool. `epu` should be installed with a standard Cubit installation, but is also available with the SEACAS tools from github.com. To combine the files, use the following command from the unix prompt:

```
epu -p 8 micro
```

A file with the name `micro.e` should be created. To view the mesh, any tool that can visualize exodus files can be used. For this example, we import the mesh using Cubit. First make sure your current working directory is correctly set. Use the `pwd` and `cd` cubit commands to set the directory. From the Cubit command prompt, to import the mesh, use the following command.

```
import mesh "micro.e" no_geom
```

This will import the mesh as a *free mesh*. This means that the elements are not associated with any geometry (volumes, surfaces, curves).

To display the elements with each material or block represented as a different color, use the following command:

```
draw block all
```

To view the interior of the mesh, you may want to use the clipping tool. The arrow in figure G-3 illustrates the clipping tool in the graphical user interface.

In addition, it may be useful to display the mesh quality of the elements. Cubit provides many options for displaying mesh quality. The following option will display fringes of the Scaled Jacobian metric:

```
quality hex all scaled jacobian draw mesh
```

Expansion Layers: For this example, it may be useful to experiment with the expansion layers (`micro_expand`) option. The input file in this example used a value of `micro_expand = 2`. Figure 2-21 in the body of this report illustrates the effect of expansion layers. This option will add additional cells to the boundary of the Cartesian grid and copy the volume fraction from the closest cells at the boundary to the new cells. This can improve quality at the boundaries of the RVE where the material interfaces meet boundary at an acute angle. For a value of `micro_expand = 2`, 2 layers would be added to each side of the Cartesian grid resulting in a data size of $100 \times 100 \times 100$ from the original $96 \times 96 \times 96$ cells defined in the `micro.tec` file.

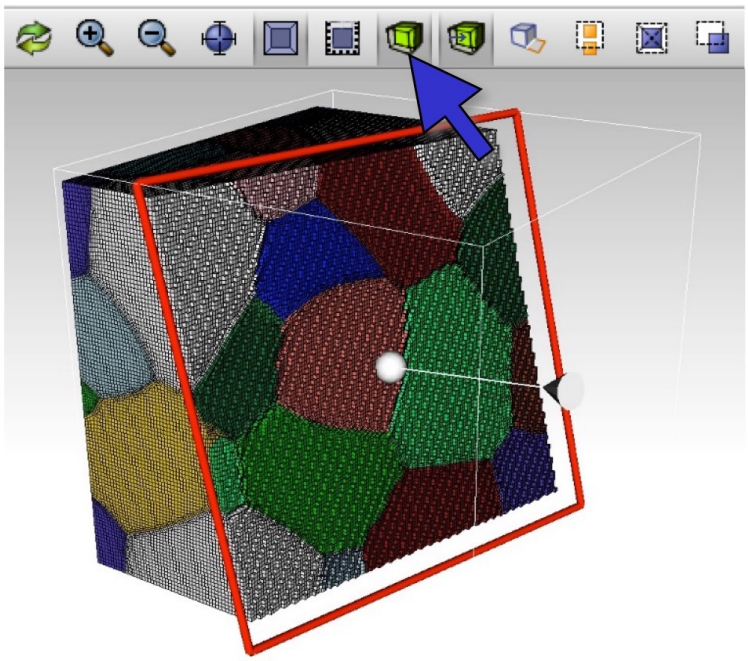


Figure G-3. Displaying the interior of hex mesh in Cubit using the clipping plane tool.

Pillowing: We note that for this example we use the `pillow_surfaces = true` option. Since this example contains many materials, we assume that *triple-junctions* will occur. A *triple-junction* occurs when three or more materials share a common curve interface. The curves defined at these interfaces tend to result in poor quality elements when nodes are projected. The option `pillow_surfaces = true` will introduce the additional layers illustrated in figure 2-28 to ensure mesh quality at *triple-junctions*.

H. EXAMPLE: CARTESIAN EXODUS

For this example, we illustrate the use of the `input_cart_exo` option to use an Exodus file as the source of the microstructure data. The exodus `block` defines a grouping of elements with a common material. For example, If 20 grains are to be represented, 20 different blocks would be defined in the exodus file with each hex element grouped according to its predominant material. For this case, the cells or elements of the input mesh define a *pure* material. In contrast to a volume fraction description that can vary continuously between 0.0 and 1.0, a *pure* cell assumes a volume fraction = 1.0 for the specified material.

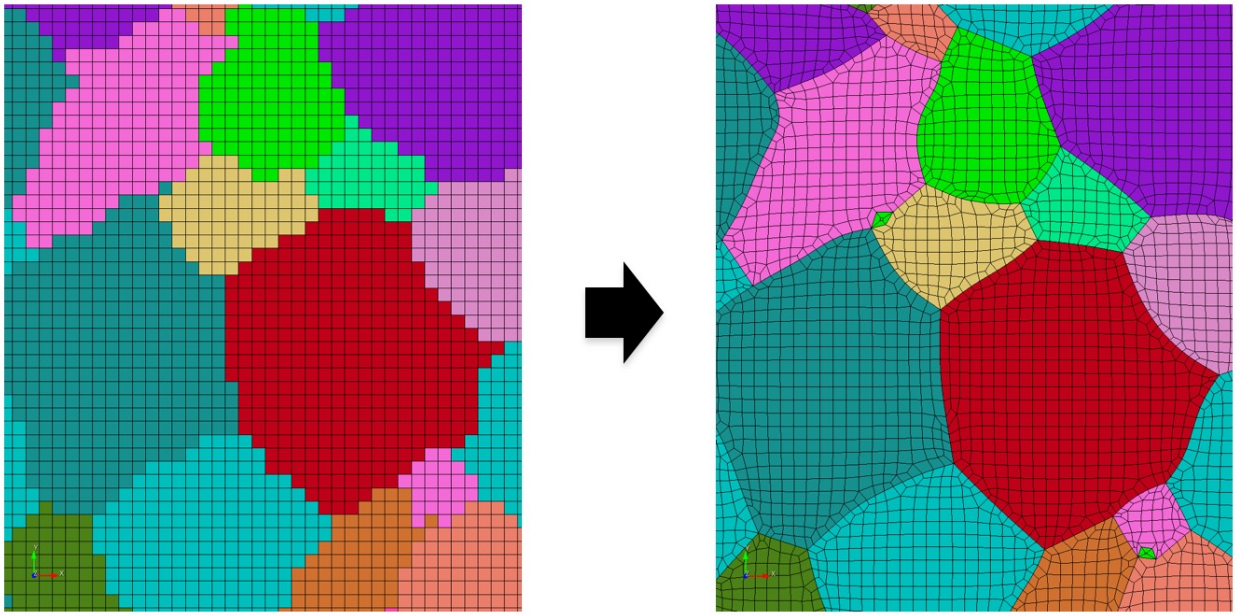


Figure H-1. *Left*: Initial Cartesian Exodus mesh displaying elements colored according to block ID. *Right*: Final mesh with smooth/conforming interface after running sculpt.

This format requires that the Exodus mesh be a regular Cartesian grid. While the dimensions may be any size, each cell must be a regular cube. This format differs from the `input_mesh` format described in Appendix G, as the `input_cart_exo` option is limited to a Cartesian grid and the element variables are not used to define material regions, but instead are distinguished by their exodus `block` assignment.

For this example we use the file `micro2D.e`, illustrated in figure H-1. This is an exodus file with hex

elements arranged as a Cartesian grid. For this example, the size of the mesh is exactly one layer thick in the Z-dimension. This is done to represent a 2-dimensional microstructure.

```
$ micro2D.i Sculpt input file for 2D microstructure
BEGIN SCULPT
    input_cart_exo = micro2D.e
    exodus_file = micro_3D
END SCULPT
```

To run `sculpt` use the following command from the unix prompt:

```
sculpt -i micro2D.i -j 8
```

This will generate a mesh with eight processors resulting in 8 separate exodus meshes on disk. Use the same methods described in Appendix G to combine the files with `epu` and import the mesh into `cubit` for visualization and to check quality.

Figure H-1 *Right* shows a portion of the final mesh generated with Sculpt. Rotating the mesh slightly in Cubit reveals the one-layer thickness as illustrated in figure H-2

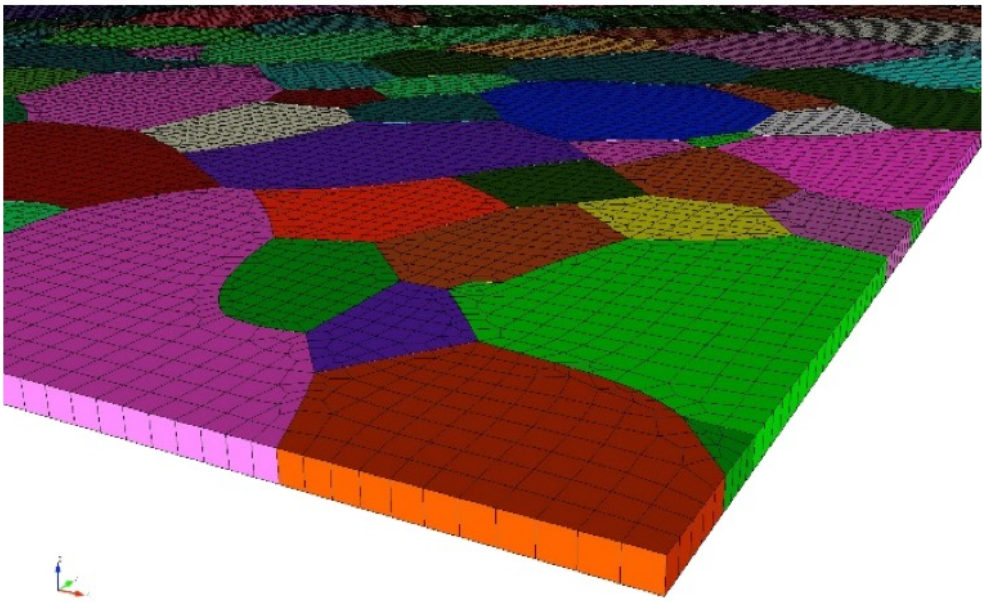


Figure H-2. Final mesh with smooth/conforming interface after running sculpt. The one-layer thick Cartesian grid illustrates the use of 2-dimensional data to build a mesh.

I. EXAMPLE: SPN FILE

The SPN file format also defines a set of *pure* cells. It is an ascii file with integers representing the material IDs present in the model. One integer is defined per cell representing the dominant material in each cell.

```
1 1 2 1 2 1 1 1 1
1 2 2 1 2 2 1 1 2
2 1 1 1 2 1 1 2 2
```

Figure I-1. Simple example of a SPN file showing two different materials

Any positive integer may be used to represent the material IDs. The resulting exodus mesh that Sculpt generates will have one block for each unique material ID defined in the `.spn` file.

Note: A material ID of zero may be used in the `.spn` file, however, since a block ID of zero is not permitted in Exodus, the resulting block ID in the exodus mesh for any element identified with zero will be $1 + ID_{max}$, where ID_{max} is the maximum material ID present in the file.

Since the `.spn` file format defines only one integer per cell, it is much more compact than the volume fraction format `.tec` file described in Appendix G. However, the resulting geometry of the material interfaces may not be as accurate or smooth for the `.spn` format compared with the `.tec` format.

Ordering of integers in the `.spn` file is important as it will determine the placement of each cell within the RVE. The default ordering of the integers is represented by the following C code:

```
for (i=0; i<nelx; i++)
  for (j=0; j<nely; j++)
    for (k=0; k<nelz; k++)
      // read next value from file
```

where `(nelx, nely, nelz)` are the number of cells in each dimension. For the default case illustrated, sculpt will read cells in the `Z`-axis first, followed by `Y`, then `X`. For data that is not ordered in this manner, the `spn_xyz_order` option can be used to specify the ordering of cells.

An example sculpt input file for a `.spn` file is as follows:

```
$ TwoPhase.i Sculpt input file to generate 2-phase microstructure
BEGIN SCULPT
    input_spn = TwoPhase.spn
    exodus_file = TwoPhase

    $ number of cells (required)
    nelx = 64
    nely = 64
    nelz = 64

    $ smoothing options
    smooth = fixed_bbox
    csmooth = vfrac
    laplacian_iters = 10
    max_opt_iters = 20
END SCULPT
```

Note that unlike the `.tec` format, the values for `(nelx, nely, nelz)` must be specified as part of the input.

For this example we use the file `TwoPhase.spn` which contains two different materials representing a viscous fluid. Running sculpt with the above input file and `TwoPase.spn` results in the meshes in figures I-2 - I-3.

Smoothing: This input file also includes additional options for smoothing. Since the raw data from *pure* cells is *stair-step* in nature, additional experimentation with smoothing parameters may be worthwhile to achieve desired smooth material interfaces. The smoothing methods used for surfaces and curves are set using the `smooth` and `csmooth` options respectively. The setting `smooth = fixed_bbox` ensures that the six faces of the RVE box will remain planar. Increasing the number of iterations for smoothing using the `laplacian_iters` and `max_opt_iters` options can effect the smoothness of the surfaces, but can also effect mesh quality. See the sculpt help for more information on curve and surface smoothing.

As an additional example, it is possible to increase the smoothness of the material interfaces by using the surface and curve smoothing methods as described in section 2.12.6.1 of this report.

```
smooth = no_surface_projections
csmooth = hermite
```

The smoothing procedures will improve element quality by moving interior nodes to optimize quality. Nodes that are associated with material interfaces are projected to the approximated surfaces and curves as described in section 2.9.1 and 2.9.3. When using the option `smooth = no_surface_projections`, nodes associated with surfaces are not projected to interfaces, but instead are free to *float* to improve quality. This has the effect of creating very smooth surfaces, however can result in volumes collapsing or losing geometric volume, especially for concave shapes. The curve

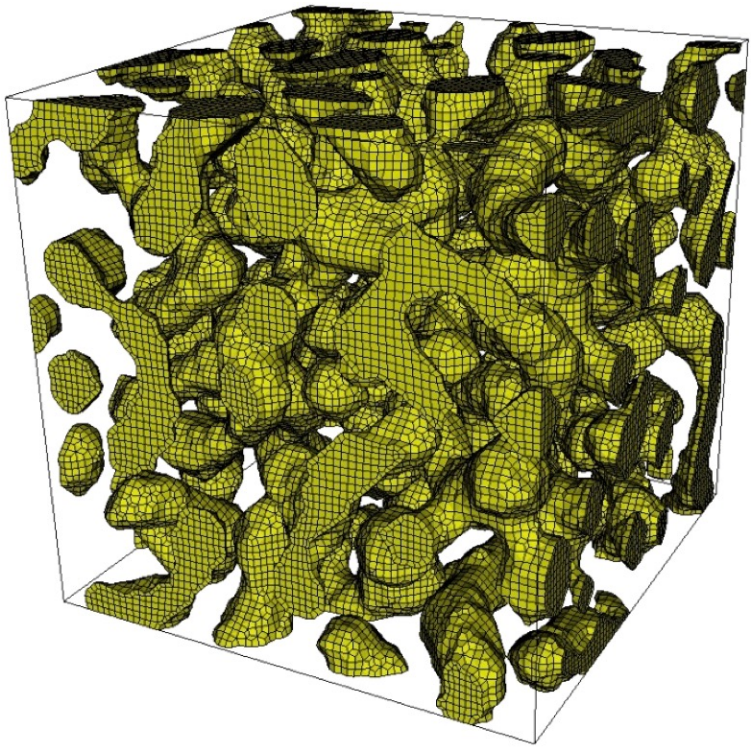


Figure I-2. Mesh generated from `TwoPhase.spn` file showing only one of the resulting material blocks.

smoothing method `csmooth = hermite` will also smooth curve interfaces using a hermite approximation of the curve, but can also result in a *collapasing* effect for some cases.

Figure I-4 illustrates the effect of using both options. Some control over the amount of collapsing or loss of volume can be controlled by modifying the number of laplacian smoothing iteration using the option `laplacian_iters` option. In most cases just 2 or 3 laplacian iterations may be sufficient to achieve results when using `smooth = no_surface_projections`.

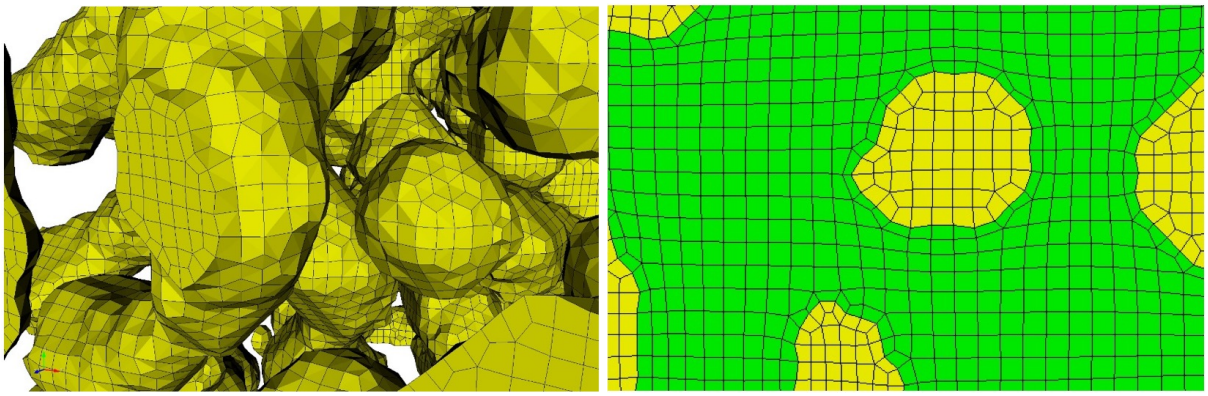


Figure I-3. Closeup of meshes from figure I-2. *Right* shows mesh at surface of RVE.

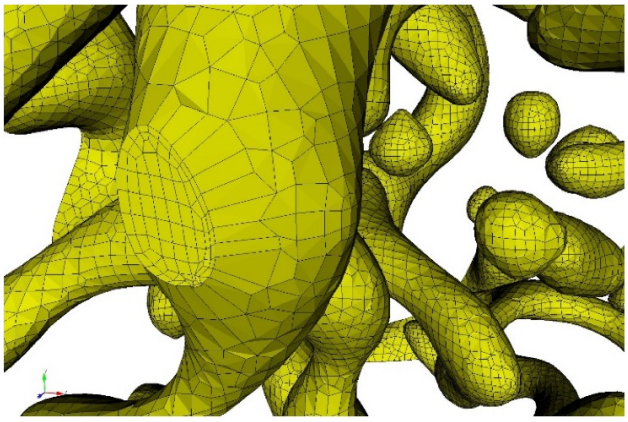


Figure I-4. Closeup of mesh from figure I-2. Illustrates the use of options `smooth = no_surface_projections` and `csmooth = hermite`. Compare with figure I-3 with default smoothing options.

J. EXAMPLE: TETRAHEDRAL MESH

The Sculpt tool currently provides only options for hexahedral mesh generation. To generate a tetrahedral mesh with Sculpt, we first generate the geometry from one of the standard microstructure format files (`input_micro, input_cart_exo, input_mesh, input_spn`) using Sculpt. We can then use Cubit as our tetrahedral meshing tool to generate the final mesh on the resulting geometry that Sculpt produces.

To generate a tetrahedral mesh, we first build a geometry in Sculpt that Cubit can use for tet meshing. Normally, when generating a hex mesh in Sculpt we must first build geometric curves and surfaces before generating the hexes. We can use this same procedure in sculpt for building geometry for tets. Once the geometry is built, it can be exported as two separate files for use in Cubit:

1. `Exodus File`: Contains triangle elements describing only the material interfaces and boundaries.
2. `Sculpt to Geometry File (S2G)`: Contains description of curves and surfaces and their association to surface triangles contained in the exodus file.

These files can then be used to construct a *mesh-based geometry* in Cubit that can be used as the basis for a tetrahedral mesh.

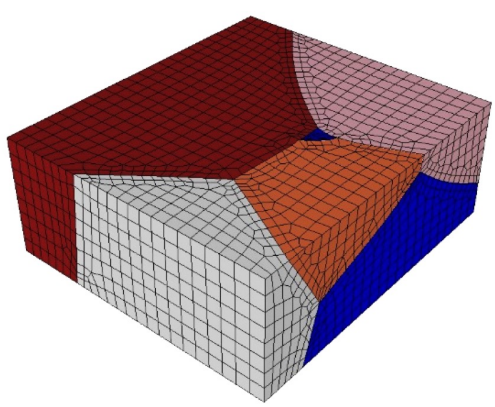


Figure J-1.

For example, figure J-1 shows a hex mesh generated in Sculpt from the `input_micro` option. To generate a tet mesh, instead of exporting the result as a hex mesh, we instead export an exodus and S2G file to be used in Cubit. After import to Cubit, a mesh-based geometry is constructed and meshing sizes

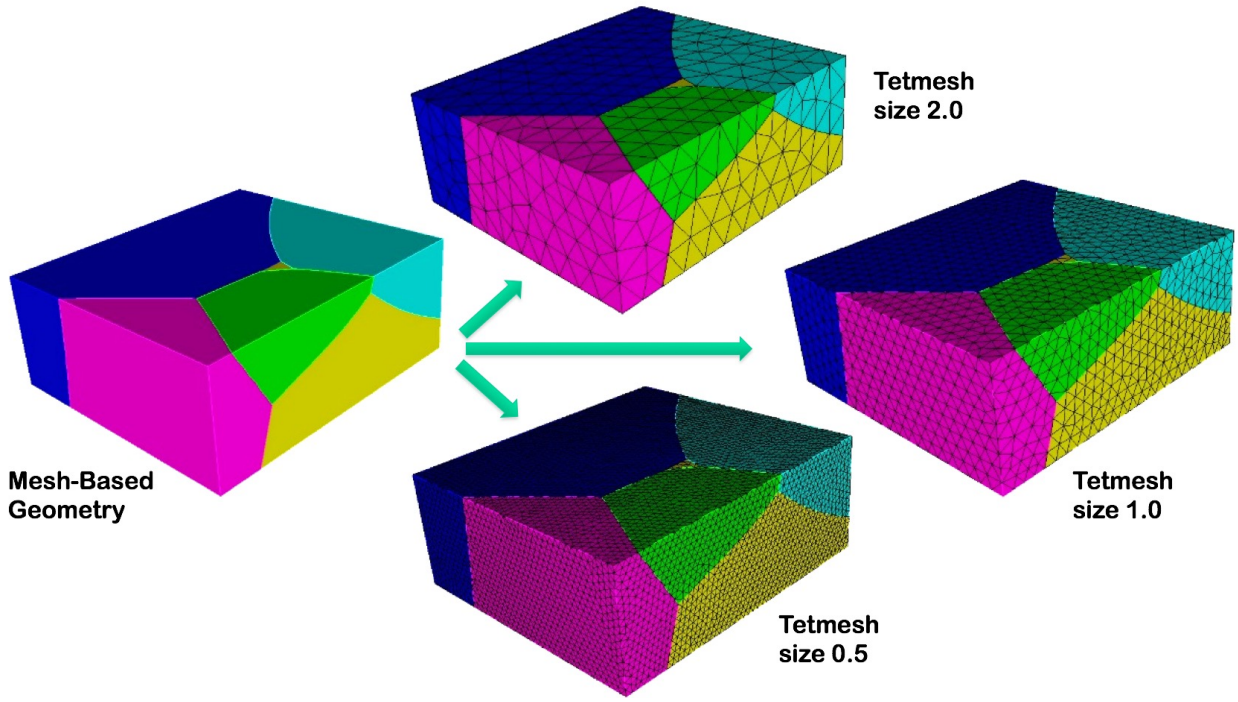


Figure J-2.

and schemes applied. Figure J-2 shows the resulting mesh-based geometry for this example. Figure J-2 also shows tet meshes at three different resolutions, illustrating the ability to use various tet meshing options to control the resolution and grading of the tet mesh.

For this example, we again use the file `micro.tec` which contains 20 materials on a $96 \times 96 \times 96$ Cartesian grid. To reduce run-rime for this example, we will use a clipping boundary to only utilize a portion of the full grid. Copy the following text to an input file and run it with the command `sculpt -i <filename>.i`. We also note that the `trimesh` and `write_geom` options are currently limited to serial execution. As a result, sculpt must be executed on a single processor for this case (`-j 1`).

```
$ micro_tet.i Sculpt input file to generate Cubit input for tet mesh
BEGIN SCULPT
    input_micro = micro.tec
    exodus_file = micro_clipped
    trimesh = true
    write_geom = true
    defeature = 3
    defeature_bbox = true
    smooth = no_surface_projections

    $ specify a clipping boundary so we don't generate the full mesh
    xmin = 52.145432
    ymin = 87.530418
```

```
    zmin = 39.749386
    xmax = 73.055649
    ymax = 150.00000
    zmax = 63.183693
END SCULPT
```

We note some additional options in this example:

Triangle Mesh: The `trimesh = true` option will generate triangles on the interfaces and boundaries that are defined in Sculpt and write them to an exodus file. These triangles are the result of splitting the quadrilaterals that would otherwise be defined on the surfaces when generating a hex mesh. The resulting exodus mesh using the `trimesh` option will contain only triangles of type TRI3.

Write Geometry: The `write_geom = true` option will export an ascii file (`.s2g`) containing the logical grouping of triangles defined in the exodus file into topological surfaces. This file is used by Cubit to construct the mesh-based geometry used for tet meshing.

Defeaturing: The option `defeature = 3` will do two main operations: filtering and collapsing:

1. *Filtering*: Remove small volumes, protrusions and isthmus from the data as described in section 2.7 of this report.
2. *Collapsing*: Collapse small curves and surfaces as described in section 2.7 of this report.

Once sculpt has completed, two files will have been generated:

```
micro_clipped.s2g
micro_clipped.e.1.0
```

From the Cubit command line, these files may be imported with a single command. Note that the S2G file format is currently a beta feature in Sculpt. As a result it is currently necessary to turn on a developer mode. To do so, issue the following command in Cubit:

```
set dev on
```

To import the files, Use the following command:

```
import s2g "micro_clipped"
```

This option should import both files and build a mesh based geometry from them. For this simple example, you should see 5 separate volumes that have their surfaces meshed with triangle elements that were generated by Sculpt. These triangles are normally not of sufficient quality to use as the basis for tet meshing, so we can delete these:

```
delete mesh
```

Set a mesh size, set the meshing scheme to `tetmesh` and mesh the volumes:

```
vol all size 1.0
vol all scheme tetmesh
mesh vol all
```

A. CONTROLLING TET GRADING

One common requirement for microstructures is to have mesh sizes smaller at material interfaces, but larger towards the interior of grains. The following illustrates one approach to generating a graded mesh in Cubit with this characteristic. After importing the geometry, the following sequence of Cubit commands can be used to generate the mesh shown in figure J-4

Note that after setting up controls for the tet mesher, we create a group comprising all of the interior surfaces as shown in figure J-3. These surfaces are assigned a smaller mesh size than their surrounding volumes.

```
#
#  Cubit Script for defining a graded mesh with refinement at material interfaces
#
# set up tetmesher with maximum element optimization and gradient control
volume all scheme tetmesh
tetmesher optimize level 6 overconstrained off sliver off
trimesher surface gradation 1.2
trimesher volume gradation 1.2
trimesher geometry sizing off
volume all size 2.0
surface all size 2.0
curve all size 2.0

#create a group of interior surfaces (exclude those on the RVE boundary)
#{xmin = 52.145432}
#{ymin = 87.530418}
#{zmin = 39.749386}
#{xmax = 73.055649}
#{ymax = 96.5}
#{zmax = 63.183693}
#{eps = 0.001}
group "interior_surfs" add surface with x_coord > {xmin + eps} and \
    x_coord < {xmax - eps} and with y_coord > {ymin + eps} and \
    with y_coord < {ymax - eps} with z_coord > {zmin + eps} and \
    with z_coord < {zmax - eps}
```

```
# set a smaller size on the material interfaces
# mesh and smooth the interior surfaces first
surf in interior_surfs size 0.5
mesh surf all
surf in interior_surfs smooth scheme mean ratio
smooth surf in interior_surfs

#mesh with tets
mesh vol all
```

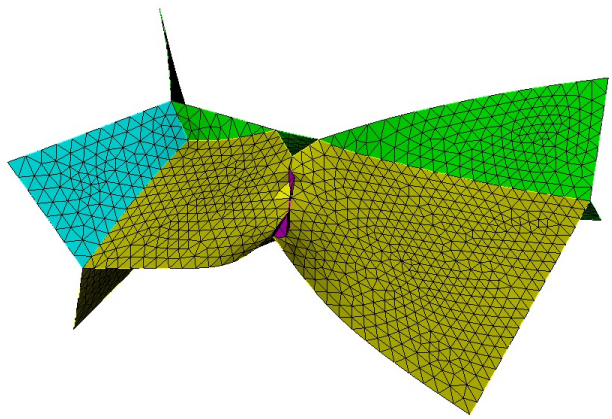


Figure J-3. Interior or material interface surfaces defined in group "`interior_surfs`" in the above script. These surfaces are assigned a size of 0.5

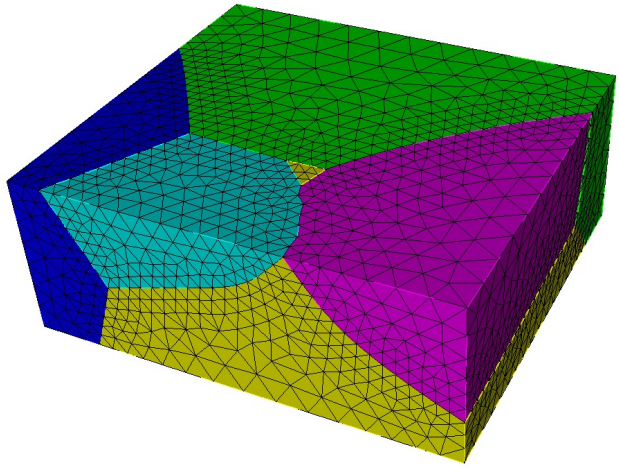


Figure J-4. Mesh generated in Cubit using the above script. Mesh is size 0.5 at the material interfaces transitioning to a size of 2.0 on the interior of the grains.

K. EXAMPLE: ANALYTIC GEOMETRY

One option for generating data for microstructures is from analytic geometry. For our example, we use spheres that are defined by a center and radius. We use the Diatom format described in the CTH documentation [7] which has various options for building primitive geometry types.

The following is an example Diatom format that defines six separate spheres that overlap with common centers but with progressively larger radii. When Diatom geometry overlaps, the first sphere in the list takes precedence. This results in the concentric pattern of spheres shown in figure K-1.

```
$ Diatom File: two_spheres.diatom
diatom
  package "spheres1"
    material 1
    insert sphere
      center = -5 0 0
      radius = 4
    endinsert
    insert sphere
      center = 5 0 0
      radius = 4
    endinsert
  endpackage
  package "spheres2"
    material 2
    insert sphere
      center = -5 0 0
      radius = 5
    endinsert
    insert sphere
      center = 5 0 0
      radius = 5
    endinsert
  endpackage
  package "spheres3"
    material 3
    insert sphere
```

```
      center = -5 0 0
      radius = 6
    endinsert
    insert sphere
      center = 5 0 0
      radius = 6
    endinsert
  endpackage
enddiatom
```

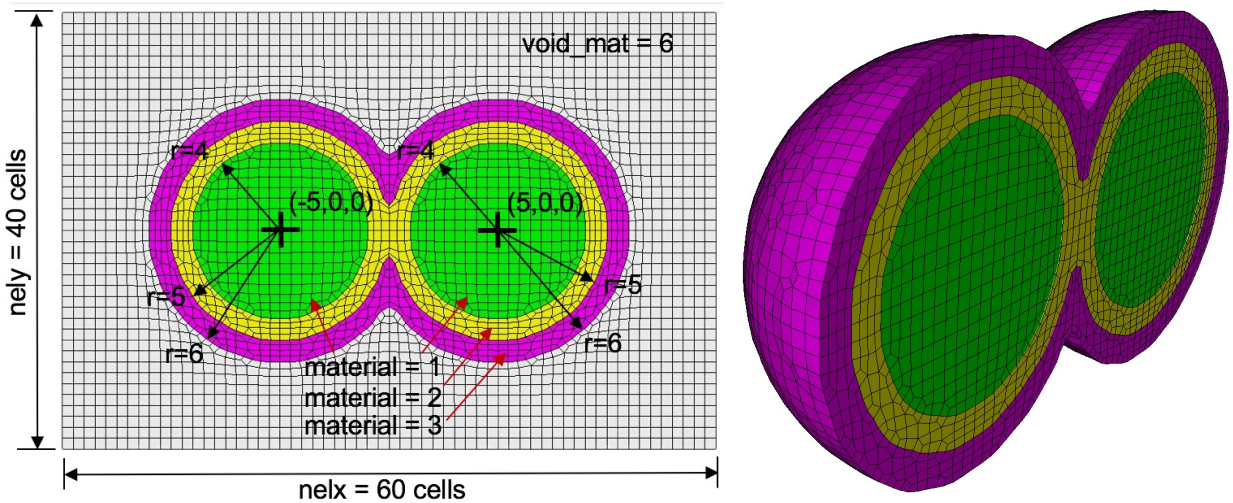


Figure K-1. Example mesh generated from analytic spheres defined from Diatoms

The following is a sculpt input file for this use case.

```
$ Sculpt input file: two_spheres.i
BEGIN SCULPT
    diatom_file = two_spheres.diatom
    exodus_file = two_spheres
    xmin = -15
    ymin = -10
    zmin = -10
    xmax = 15
    ymax = 10
    zmax = 0
    nelx = 60
    nely = 40
    nelz = 20
    smooth = fixed_bbox
    mesh_void = true
    void_mat = 6
END SCULPT
```

For this type of data, while the sphere geometry can be defined in the diatom file, the overlay grid description must be included in the input file (`xmin, ymin, zmin, xmax, ymax, zmax, nelx, nely, nelz`). The smooth option `smooth = fixed_bbox` should also be used to ensure nodes at boundaries will be projected to the 6 planar faces of the RVE.

Mesh Void: The `mesh_void = true` option ensures that the mesh will not only be generated in the spheres defined in the diatom description, but also in the void regions surrounding the spheres. While the material or block ID for the individual spheres may be defined in the diatom file, the block ID for the void region should be specified in the input file. In this example, we have identified the elements in the void region as `block 6`.

To run this simple example, create two files from the above text named `two_spheres.diatom` and `two_sphere.i` respectively. To run sculpt, use the command as follows:

```
sculpt -i two_spheres.i
```

This should result in a file named `two_spheres.e.1.0` which is displayed in figure K-1

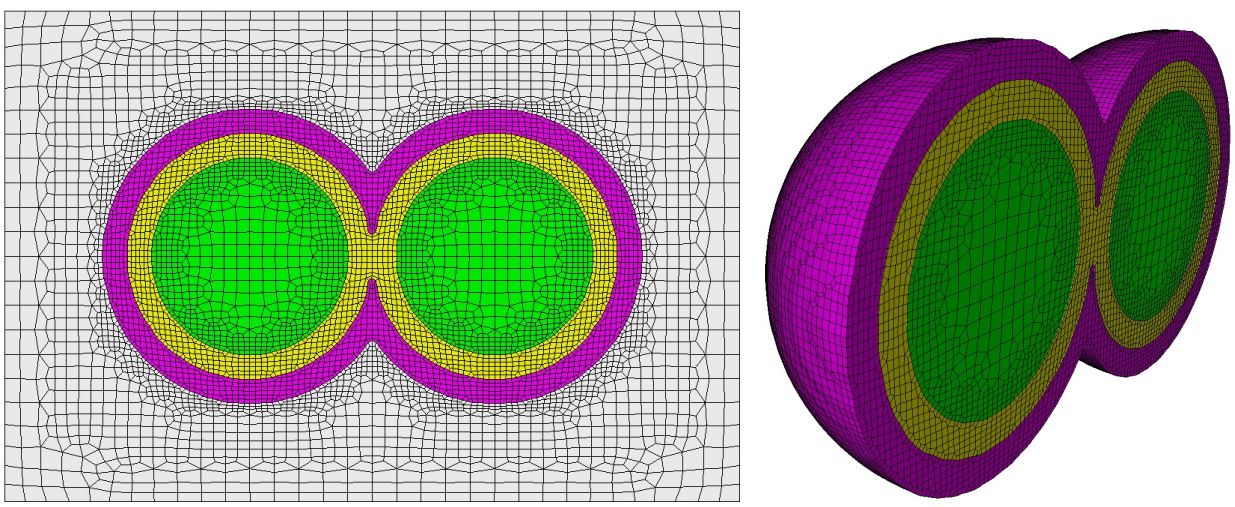


Figure K-2. Example use of adaptivity on analytic geometry. Compare to figure K-1 without adaptivity.

A. ADAPTIVE MESHING

Sculpt also provides the capability to adapt the mesh (refine or coarsen) based upon various geometric indicators. This example extends the analytic geometry example shown above to use the adapt capability. Add or replace the following lines to the input file `two_spheres.i` shown above and re-run sculpt.

```
nelx = 30
nely = 20
```

```
nelz = 10
adapt_type = vfrac_average
adapt_threshold = 0.0001
adapt_levels = 2
pillow_boundaries = true
```

The resulting mesh is displayed in figure K-2

Grid Resolution: Note that the initial overlay grid intervals can usually be decreased when using adaptivity. The options `nelx, nely, nelz` are used to define the initial resolution before adaptively splitting cells.

Adapt Type: The `adapt_type` option sets the criteria for adapting or splitting the cells into smaller cells. Check the sculpt documentation for definitions of the various criteria for adaptivity. The `adapt_levels = 2` sets the maximum number of times a hex may be split.

Pillowing: In this case the `pillow_boundaries` option is used since refinement extends to the boundary of the RVE. Figure 2-29 illustrates the effect of using this option. Omitting `pillow_boundaries` in this case, will result in elements with Scaled Jacobian of zero at boundaries where elements are refined.

L. EXAMPLE: PERIODIC MESH

The following example illustrates the use of the `periodic` option. In some cases it is useful to model the RVE as a continuous model where nodes and faces at the boundaries are matched precisely on opposite sides. This example also demonstrates the use of adaptivity to reduce overall element counts.

This example uses the diatom file format described in Appendix K that prescribes a set of analytic spheres. To ensure a periodic mesh, the geometry must also be periodic. This may mean replicating geometric structures over multiple periods in the diatom file. Figure L-1 shows the geometry used for this example with the cube shape representing the geometric period that is to be meshed.

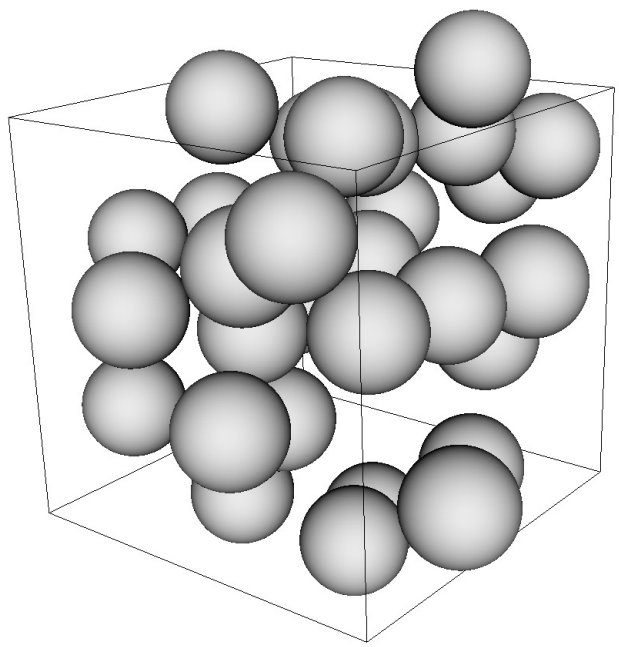


Figure L-1. Spheres defined from a diatom file. Brick represents one period in each Cartesian direction.

```
$ spheres_periodic.i Sculpt input file demonstrating periodic meshing
BEGIN SCULPT
    diatom_file = spheres_periodic.diatom
    xmin = -18.705510
    ymin = -18.705510
    zmin = -18.705510
    xmax = 18.705510
```

```
        ymax = 18.705510
        zmax = 18.705510
        nelx = 38
        nely = 38
        nelz = 38
        adapt_type = vfrac_average
        adapt_levels = 1
        adapt_threshold = 0.0001
        periodic = true
        exodus_file = spheres_periodic
        mesh_void = true
    END SCULPT
```

To run this example, create a file named `spheres_periodic.i` from the above text. Ensure that you also have the file `spheres_periodic.diatom` containing the analytic sphere data in your working directory. To run sculpt, use the command as follows:

```
sculpt -i spheres_periodic.i -j 16
```

Note that this option will generate 16 separate exodus files in your working directory when you use the `-j 16` option. To combine these files into a single file use the `epu` tool as follows:

```
epu -p 16 spheres_periodic
```

This should result in a single file, `spheres_periodic.e` being generated in your working directory. To visualize the stair-step mesh in Cubit, use the following command in Cubit to import the mesh and display the blocks as separate colors:

```
import mesh "spheres_periodic.e" no_geom
draw block all
```

Figure L-2 illustrates the resulting mesh for all blocks and just the spheres. To visualize mesh quality using the scaled Jacobian metric, use the following from the Cubit command line:

```
quality hex all scaled jacobian draw mesh
```

We note several new options in the above input file and discuss their implications here:

Adapt Type: This example illustrates the `adapt_type = vfrac_average (4)` option. While the base Cartesian grid is defined with intervals $38 \times 38 \times 38$, introducing `adapt_type = vfrac_average (4)` will refine cells where the measured volume fraction of the sphere geometry

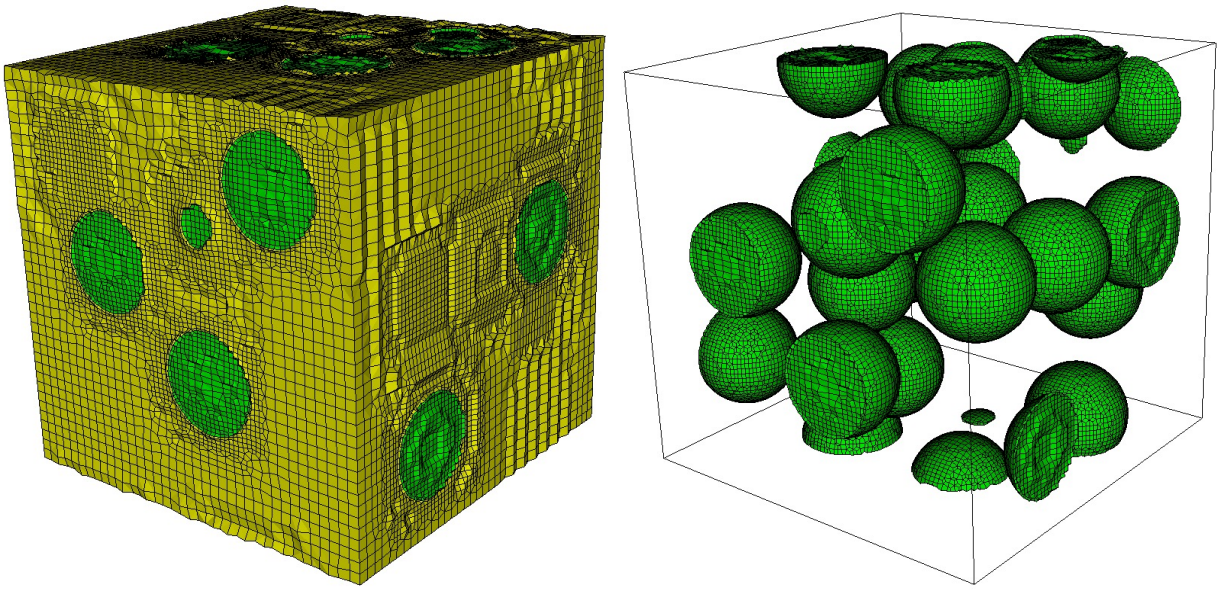


Figure L-2. Periodic mesh of full mesh (left) and the spheres only (right).

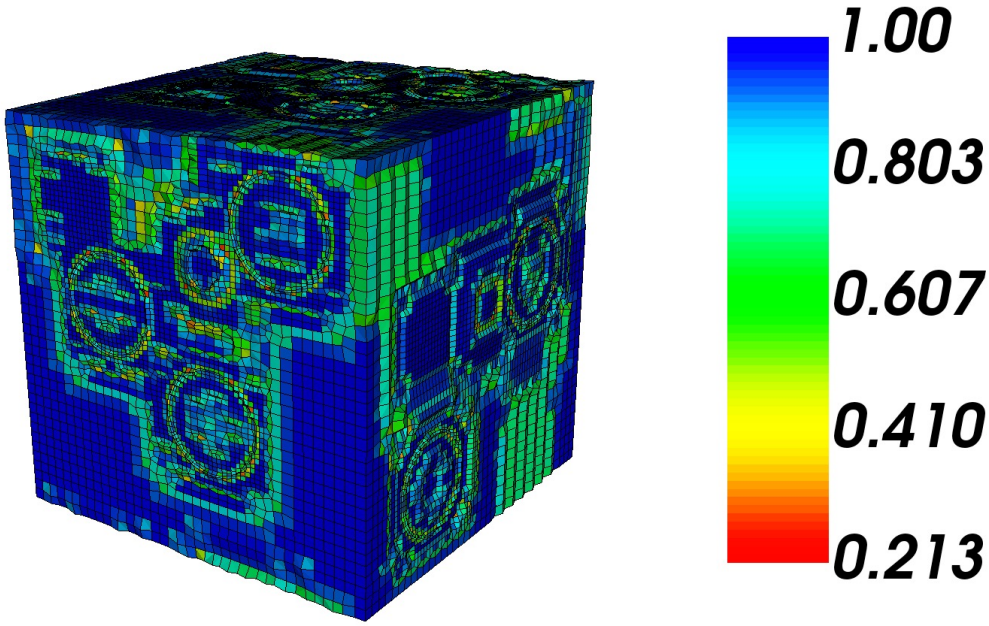


Figure L-3. Scaled Jacobian metric displayed in periodic mesh

changes by a threshold defined by the `adapt_threshold` option. We use a small value (0.0001) for `adapt_threshold` to ensure refinement occurs at boundaries of all spheres.

We also use an `adapt_levels` value of 1. This limits the number of refinement levels to a single iteration so that cells will be subdivided a maximum of one time where needed. Increasing this value to 2 or 3 will resolve the spheres better, but increase the element count significantly. Adjusting the base Cartesian grid resolution (`nelx, nely, nelz`) to be more coarse while increasing the `adapt_levels`

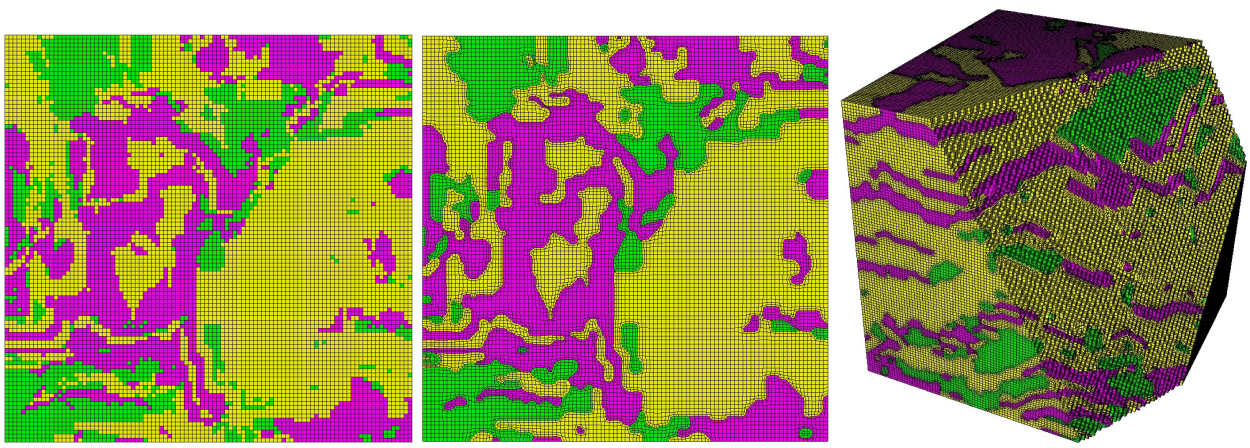
is another way to control the overall element count while maintaining geometry resolution.

Periodic: Including this option should result in a periodic mesh where copying and transforming the mesh a distance of one period should exactly match nodes and faces. As a consequence, we note that the boundaries of a periodic mesh will appear *ragged* as illustrated in figure L-2. Without the `periodic` option the six faces of the Cartesian grid are imposed as geometric constraints on the boundary elements. This can sometimes introduce artificial restrictions that can reduce element quality. The periodic option does not restrict boundary nodes to lie on one of these planes, instead allowing them to adjust to optimize element quality while maintaining periodic equivalence.

Mesh Void: This option will ensure that the region outside of the spheres defined in the diatom file will also me meshed. Without this option, only the spheres would be meshed.

M. EXAMPLE: STOCHASTIC MATERIALS

The following example uses the file `dump.spray-100-1-5.1.spn` as input. It is a $100 \times 100 \times 100$ data set comprised of 3 different materials. It represents thermal spray material grain structures from radiographic or metallographic imaging of material specimens. Figure 4-3(a) shows the front view of raw input data colored by block ID. The expected mesh is shown in Figure M-1(b). Figure M-1(c) shows an isometric view of the final mesh with a cut-away plane to reveal the interior elements.



(a) initial raw cell data (b) final mesh (c) isometric view of mesh

Figure M-1. Thermal spray material RVE mesh

A. STAIR-STEP MESH

It is possible to utilize the raw data in the *.spn* file as a finite element mesh without smoothing the interfaces between elements. The `stair` option will simply convert each cell represented by an integer in the *.spn* file into a hex element. Each hex will be assigned to its appropriate material block in the exodus format. The following input file will generate the mesh shown in figure M-1(a).

```
$ dump.spray-100-1-5.1-stair.i file for generating stair-step mesh
BEGIN SCULPT
    stair = fast
    input_spn = dump.spray-100-1-5.1.spn
```

```
    nelx = 100
    nely = 100
    nelz = 100
    exodus_file = dump.spray-100-1-5.1-stair
END SCULPT
```

To run this example, create a file named `dump.spray-100-1-5.1-stair.i` from the above text. Ensure that you also have the file `dump.spray-100-1-5.1.spn` containing the input data in your working directory. To run sculpt, use the command as follows:

```
sculpt -i dump.spray-100-1-5.1-stair.i -j 16
```

Note that this option will generate 16 separate exodus files in your working directory when you use the `-j 16` option. To combine these files into a single file use the `epu` tool as follows:

```
epu -p 16 dump.spray-100-1-5.1-stair
```

This should result in a single file, `dump.spray-100-1-5.1-stair.e` being generated in your working directory. To visualize the stair-step mesh in Cubit, use the following command in Cubit to import the mesh and display the blocks as separate colors:

```
import mesh "dump.spray-100-1-5.1-stair.e" no_geom
draw block all
```

B. BASIC MESH GENERATION

We note that very few options are needed in order to generate a mesh from a *.spn* file, as most options can be defaulted. The following is an example of the minimum information required in the input file.

```
$ dump.spray-100-1-5.1-basic.i Basic sculpt input file
BEGIN SCULPT
    input_spn = dump.spray-100-1-5.1.spn
    nelx = 100
    nely = 100
    nelz = 100
END SCULPT
```

Note that only the *.spn* file name and the number of elements in each direction in the Cartesian grid are required. (See the `spn_xyz_order` option to customize the expected ordering of the data in a *.spn* file). Other options are automatically defaulted. To check the options that will be used in addition to those specified in the input file, the `-print_input` option can be used. For example, for the above file, issue the following sculpt command:

```
sculpt -i dump.spray-100-1-5.1-basic.i --print_input
```

This will display all options available in Sculpt along with their *user* set value or *default* value. We should note however, for this example, the default options will not be sufficient to successfully generate a mesh.

C. CONTROLLING FILTERING AND ELEMENT QUALITY

For cases that include stochastic materials, represented by this example, the following illustrates options that can be useful in controlling filtering and element quality.

```
$ dump.spray-100-1-5.1.i Sculpt input file for generating thermal spray mesh
BEGIN SCULPT
    input_spn = dump.spray-100-1-5.1.spn
    nelx = 100
    nely = 100
    nelz = 100
    exodus_file = dump.spray-100-1-5.1
    defeature = 1
    defeature_iters = 20
    smooth = 3
    csmooth = 4
    laplacian_iters = 2
    curve_opt_thresh = 0.2
    thicken_material = 3 0.2
    thicken_material = 1 0.33
    compare_volume = true
END SCULPT
```

To run this example, create a file named `dump.spray-100-1-5.1.i` from the above text. Ensure that you also have the file `dump.spray-100-1-5.1.spn` containing the input data in your working directory. To run sculpt, use the command as follows:

```
sculpt -i dump.spray-100-1-5.1.i -j 16
```

Note that this option will generate 16 separate exodus files in your working directory when you use the `-j 16` option. To combine these files into a single file use the `epu` tool as follows:

```
epu -p 16 dump.spray-100-1-5.1
```

This should result in a single file, `dump.spray-100-1-5.1.e` being generated in your working directory. To visualize the mesh in Cubit, use the following command in Cubit to import the mesh and display the material blocks as separate colors:

```
import mesh "dump.spray-100-1-5.1-stair.e" no_geom
draw block all
```

To visualize mesh quality using the scaled Jacobian metric, use the following from the Cubit command line:

```
quality hex all scaled jacobian draw mesh
```

We note several new options in the above input file and discuss their implications here:

Defeaturing: The `defeature = 1` option will turn on the defeaturing option as described in section 2.7.2. This option is executed prior to generating the hex mesh and smoothing, and will attempt to reassign materials to cells to avoid small clusters of the same material. The `min_vol_cells` option can also be used to control the size of the clusters that will be reassigned. The default for this value is 5, which indicates, any cluster with 5 or fewer cells will be reassigned to the predominant neighboring surrounding material. Defeaturing will also reassign cells that are identified as protrusions or isthmus.

The `defeature_iters = 20` option in the above input file is used to increase the maximum number of defeaturing iterations used. In most cases, where stochastic properties are not manifest, the defeaturing operation will resolve in 2 or 3 iterations. For this case, however, we increase the maximum from the default 10 iterations to 20 to allow the defeaturing to complete successfully. Some trial and error may be required to set this value. Observing the sculpt output during the defeaturing procedure will indicate whether sufficient defeaturing iterations were specified.

Smoothing: For this case we specified a surface smoothing option (`smooth = 3`) that will maintain the bounding box of the RVE, but will not project nodes at material interfaces to the implicitly defined surfaces. This option tends to improve the smoothness of the material interfaces and is described in more detail in section 2.12.6.1. In addition, for curve smoothing (`csmooth = 4`) we use the `neighbor_surface_normal (4)` option which has proven most reliable for this type of data.

We also note the `laplacian_iters = 2` option is used to limit the number of Laplacian smoothing iterations. When using the `smooth = 3` option to not project nodes to interfaces, using too many Laplacian iterations can result in collapsing of some of the smaller volumes and reducing volume in localized concave regions. As a result, a small number of Laplacian iterations is usually sufficient when using this option for data such as this.

One additional option that has proven useful is the `curve_opt_threshold` or C_{OT}, normally set to a value of 0.1. Increasing this option to a value of 0.2 will remove additional geometric constraints at curves, allowing for improved element quality in some cases. This option is described in more detail in section 2.12.6.2 of this document.

Thickening: The option to *thicken* individual materials is utilized in this input file. This has the effect of adding additional volume to the boundaries of the specified material. This is often done to counteract

the effects of defeaturing, which can tend to reduce the volume of some materials. In this case, the values of:

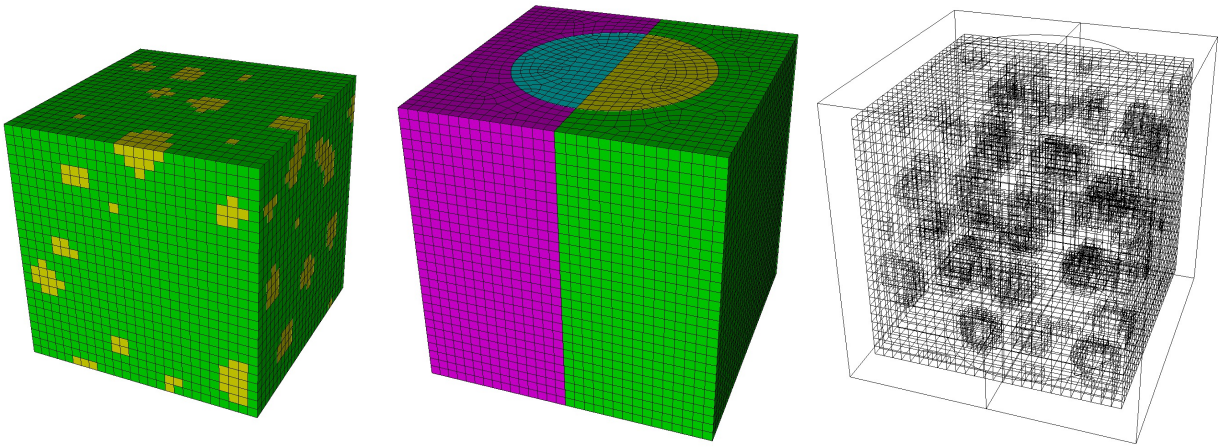
```
thicken_material = 3 0.2
thicken_material = 1 0.33
```

indicate that material 3 will first be *thickened* using a volume fraction of 0.2. This will be followed by a thickening operation to material 1 with a volume fraction of 0.33. The thickening operation will visit each cell at the boundary of the indicated material, adding the specified volume fraction to each neighboring cell. In practice, the materials and values used for thickening are selected using a trial-and-error approach in conjunction with the `compare_volume` option described below. See section 2.7.4 for more details on the thickening operation.

Compare Volume: The `compare_volume` option will write a table to output when the Sculpt procedure is completed. This provides details comparing the volume of each material from the raw input data to the volume of the final hex mesh. The information in this table can be used to help adjust thickening parameters as well as gauging the effect of smoothing and defeaturing options.

N. EXAMPLE: UNSTRUCTURED OVERLAY GRID

In many cases, the domain that contains microstructures, may not be defined by a simple RVE or rectangular domain. When this occurs, it is possible to use an unstructured `input_mesh` as the overlay grid. In this example we use an unstructured hex mesh that was initially generated in Cubit as well as a *.spn* file to describe the microstructures geometry.



(a) `balls.spn` (b) `multi_blocks.g` (c) overlay of `.spn` and `.g` files

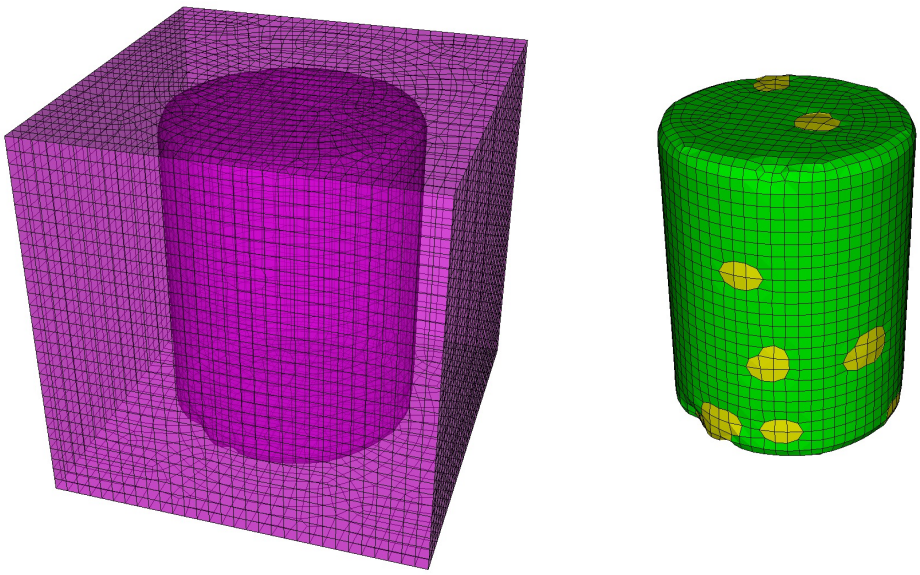
Figure N-1. Thermal spray material RVE mesh

Figure N-1 illustrates the input used for this example. Microstructure data, defined by a *.spn* file, is contained in `balls.spn` and illustrated in figure N-1(a). For this example we also use the exodus mesh, `multi_blocks.g` shown in figure N-1(b), which includes multiple material blocks. Our intent is to generate the microstructure mesh in only a designated set of blocks in the overlay grid. Figure N-1(c) also illustrates the spatial relationship between the `.spn` and `.g` files. We note that in this case, the *.spn* data is smaller than the overlay unstructured mesh. While in most cases, the *.spn* data will match or extend beyond the overlay unstructured mesh, this case demonstrates the effect when it is smaller. We also note that while the *.spn* data contains only material information defined by integers, sculpt will interpret the size of each cell of a *.spn* file as $1 \times 1 \times 1$ units as well as its spatial location where the cell at the smallest I-J-K location is defined at the origin $(0, 0, 0)$.

The following Sculpt input file can be used to generate a mesh from an unstructured overlay grid. In this case we restrict the mapping of the microstructures data to only blocks 2 and 4 on the input exodus mesh. These are represented by the yellow and blue cylindrical volumes in figure N-1(b). The final mesh

will conform with the other blocks (1 and 3), however microstructures data will not be mapped to these blocks.

```
$ multi_blocks.i Sculpt input file using an unstructured overlay grid
BEGIN SCULPT
    input_spn = balls.spn
    nelx = 26
    nely = 26
    nelz = 26
    input_mesh = multi_blocks.g
    input_mesh_blocks = 2 4
    exodus_file = multi_blocks
    mesh_void = true
    laplacian_iters = 2
    smooth = 3
    csmooth = 4
    defeature = 1
END SCULPT
```



(a)Block 3 (b)Block 1 and 2

Figure N-2. Sculpt mesh resulting from an unstructured overlay grid

Run this example using a single processor, similar to previous examples. Ensure that you have both files `balls.spn` and `multi_blocks.g` in your current working directory. Figure N-2 illustrates the result of the above input file.

```
sculpt -i multi_blocks.i -j 1
```

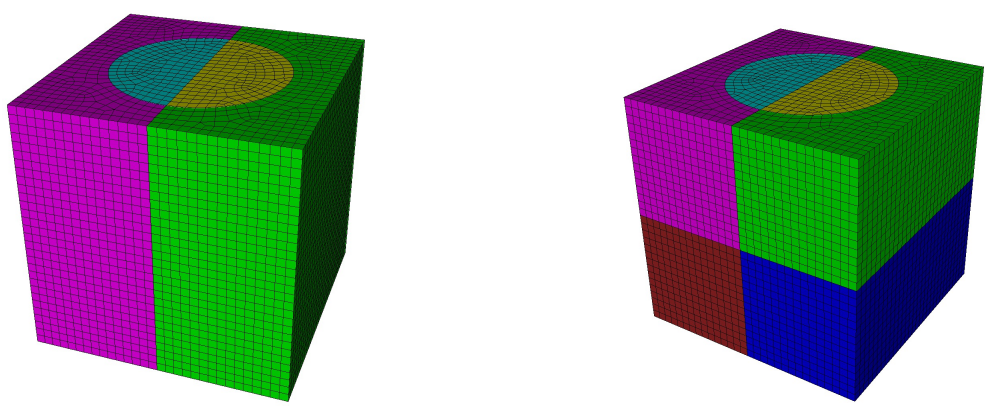
In this case, a single mesh file will be generated, so the `epu` tool is not necessary. Import the file into Cubit using the following command-line options.

```
import mesh "multi_blocks.e.1.0" no_geom
draw block all
draw block 1 2
```

To restrict the mapping of the microstructures data to a limited set of blocks in the input mesh, we used the option `input_mesh_blocks = 2 4`, which maps the materials data in the *.spn* file to only elements contained in blocks 2 and 4 of the overlay unstructured mesh. Note that block 3 (magenta) in Figure N-2(a) completely surrounds block 1 and 2 shown in Figure N-2(b) and does not contain any microstructures. If the `input_mesh_blocks` is not used, no restriction will be placed on which blocks of the input mesh will have microstructures data mapped on to it.

A. PARALLEL UNSTRUCTURED OVERLAY GRIDS

For Cartesian overlay grids, spatial decomposition and load balancing is accomplished internally in Sculpt as part of the mesh generation process. For Cartesian grids, the domain is subdivided into a set of roughly equally sized rectangular domains that are meshed on separate processors. For the unstructured case, the `input_mesh` option does not internally decompose the exodus mesh for parallel processing, instead it assumes the mesh has already been decomposed according to the specified number of processors requested. To accomplish this, we first use the `decomp` tool which is part of the `seacas` tool suite to decompose the input mesh.



(a) initial mesh without decomposition (b) mesh decomposed for 4 processors

Figure N-3. (a) initial mesh colored based on block ID (b) Decomposition applied for 4 processors. Additional colors represent decomposition for parallel

Figure N-3 illustrates the decomposition of the example input mesh `mesh_blocks.g`. To apply decomposition to `mesh_blocks.g`, ensure you have access to the `decomp` tool. Use the following command to decompose the file into four separate parallel files:

```
decomp -p 4 mesh_blocks.g
```

This should result in four separate files that are written to your working directory:

```
multi_blocks.g.4.0
multi_blocks.g.4.1
multi_blocks.g.4.2
multi_blocks.g.4.3
```

With the initial decomposed mesh defined, we can now execute the same input file, except with four processors instead of one. For example:

```
sculpt -i multi_blocks.i -j 4
```

DISTRIBUTION

Name	Org.	Sandia Email Address
Fadi F. Abdeljawad	01864	fabdelj@sandia.gov
Coleman Alleman	08363	callema@sandia.gov
Corbett C. Battaile	01864	ccbatta@sandia.gov
Joseph E. Bishop	01556	jebisho@sandia.gov
James B. Brian	01443	jbcarle@sandia.gov
Alexander Brown	01532	albrown@sandia.gov
Judith A. Brown	01516	judbrow@sandia.gov
James W. Bryson	01344	jwbryso@sandia.gov
James B. Carleton	01443	jbcarle@sandia.gov
Joe P. Castro	01341	jpcastr@sandia.gov
Brett W. Clark	01543	bwclark@sandia.gov
Peter Coffin	01553	pcoffin@sandia.gov
Mike E. Cuneo	1650	mecuneo@sandia.gov
Mohamed S. Ebeida	01464	msebeid@sandia.gov
William W. Erikson	01516	wweriks@sandia.gov
Corey D. Ernst	01543	cdernst@sandia.gov
James W. Foulk III	08363	jwfoulk@sandia.gov
Brian C. Franke	01341	bcfrank@sandia.gov
Micheal W. Glass	01545	mwglass@sandia.gov
Neal P. Grieb	02471	npgrieb@sandia.gov
Aaron C. Hall	05814	achall@sandia.gov
Byron W. Hanks	01543	bwhanks@sandia.gov
Glen A. Hansen	01443	gahanse@sandia.gov
Martin W. Heinstein	01545	mwheins@sandia.gov
Trevor Hensley	09326	thensle@sandia.gov
Chad Hovey	05421	chovey@sandia.gov
Technical Library	01177	libref@sandia.gov

DISTRIBUTION

Name	Org.	Sandia Email Address
Hojun Lim	01864	hnlim@sandia.gov
Kevin N. Long	01554	knlong@sandia.gov
Mikhail Mesh	01553	mmesh@sandia.gov
Scott A. Mitchell	01442	samitch@sandia.gov
Nathan W. Moore	01344	nwmoore@sandia.gov
Bryan V. Oliver	01340	bvolive@sandia.gov
Aaron Olsen	01341	aolson@sandia.gov
Jakob T. Ostein	08363	jtostie@sandia.gov
Steven J. Owen	01543	sjowen@sandia.gov
Byoung Yoon Park	08862	bypark@sandia.gov
Shawn D. Pautz	01341	sdpautz@sandia.gov
Micheal Powell	01443	micpowe@sandia.gov
W. Roshan Quadros	01543	wrquadr@sandia.gov
Benjamin Reedlunn	01554	breedlu@sandia.gov
Scott A. Roberts	01513	sarober@sandia.gov
Allen C. Robinson	01443	acrobin@sandia.gov
Theron Rodgers	01864	trodger@sandia.gov
Jason Sanchez	01443	jassanc@sandia.gov
Chris T. Seagle	01646	ctseagl@sandia.gov
Michael J. Skroch	01543	mjskroc@sandia.gov
Paul E. Specht	01646	pespech@sandia.gov
Matthew L. Staten	01543	mlstate@sandia.gov
Clinton S. Stimpson	01543	cjstimp@sandia.gov
John P. Sullivan	08425	jpsulli@sandia.gov
Russell D. Teeter	01555	rdteete@sandia.gov
Technical Library	01177	libref@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.