Sandia National Laboratories

Sculpt: Automatic Parallel Hexahedral Mesh Generation

Steven J. Owen, Corey E. Ernst, Clinton J. Stimpson

ABSTRACT

Sculpt is a companion application to Cubit [14] designed to run in parallel for generating all-hex meshes of complex geometry. It uses a unique overlay-grid procedure that extracts surfaces from a volume-fraction representation of the geometry. This allows for fast, automatic, fault-tolerant meshing in a high-performance computing (HPC) environment. Although Sculpt can be driven from Cubit as a GUI front-end, Sculpt was developed as a separate application so that it can be run independently from Cubit on HPC computing platforms. It was also designed as a separable software library so it can be easily integrated as an in-situ meshing solution within other codes. This work provides a brief technical discussion of the algorithms used in Sculpt as well as a complete user's manual. It includes details of the Cubit interface to Sculpt and the complete manual for the stand-alone application, including examples.

ACKNOWLEDGMENT

CONTENTS

LIST OF FIGURES

1. SCULPT TECHNICAL DESCRIPTION

This chapter provides a brief technical overview of the Sculpt application, a separate companion application to Cubit designed to generate all-hex meshes of complex geometries.

1.1. SCULPT PROCEDURE
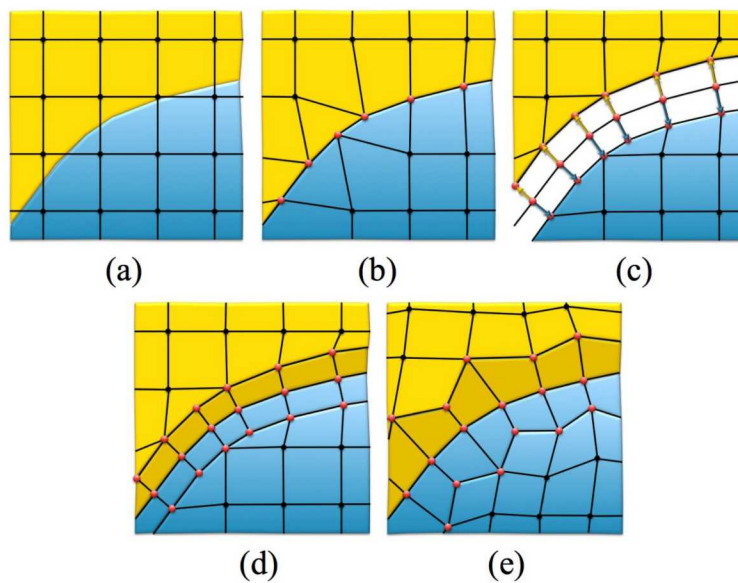
Figure 1-1. The procedure for generating a hex mesh using the Sculpt overlay grid method

The method for generating an all-hex mesh employed by Sculpt is often referred to in the literature as an *overlay-grid* or *mesh-first* method. This differs significantly from the algorithms employed by Sweeping and Mapping, which are classified as geometry-first methods. Geometry-frst meshing techniques are the most common methods used in Cubit [14], which can result in a very high quality mesh. Geometry-first methods usually require manual decomposition of the geometry into mappable or sweepable volumes to fit a recognized topology, a process that can often be very time consuming, tedious and sometimes impossible. In contrast, the Sculpt method begins with an overlay grid encompassing the geometry. The overlay grid is often a bounding-box fitted Cartesian grid but can also be any structured or unstructured mesh that overlaps the geometry. Geometric features are carved or sculpted from the overlay grid and

boundaries smoothed to create the final hex mesh. Unlike Mapping and Sweeping, Sculpt does not need a recognized topology on which to operate. This eliminates the need for prior decomposition which can be an enormous time savings for users. Input to Sculpt can be any geometry in the form of a standard STL format or geometric primitives. Input can also be voxel-based data in the form of volume fractions or bitmaps.

The basic Sculpt procedure is illustrated in figure 1-1. Beginning with a Cartesian grid as the base mesh, shown in figure 1-1(a), a geometric description is imposed. Nodes from the base grid that are near the boundaries are projected to the geometry, locally distorting the nearby hex cells (figure 1-1(b)). A pillow layer of hexes is then inserted at the surfaces by duplicating the interface nodes on either side of the boundaries and inserting hexes (figures 1-1(c) and (d)). While constraining node locations to remain on the interfaces, smoothing procedures can now be employed to improve mesh quality of nearby hexes (figure 1-1(e)).
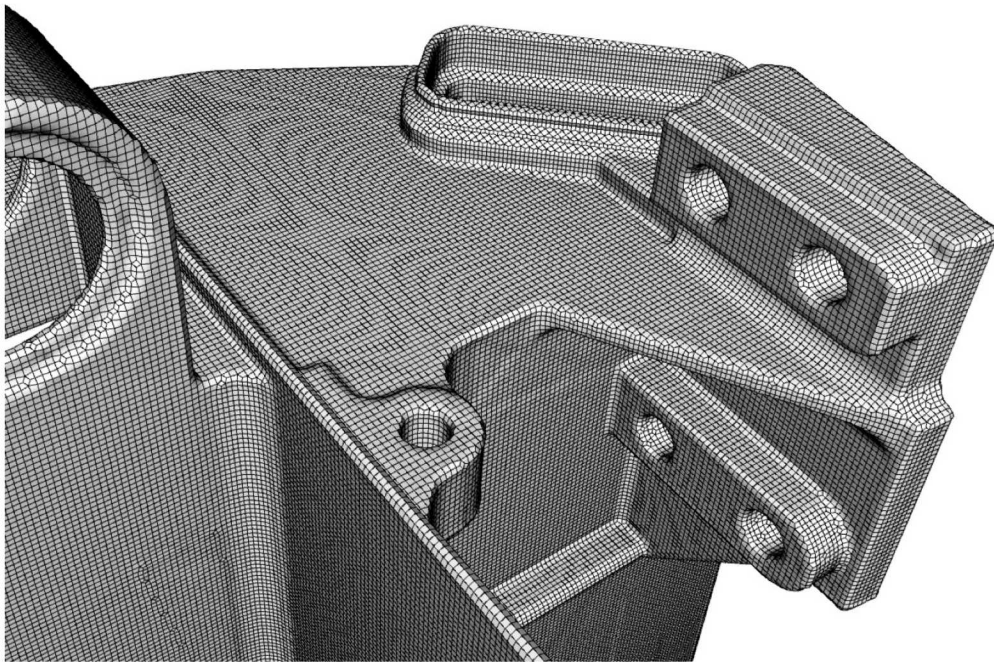
Figure 1-2. Hex mesh generated using the Sculpt overlay grid procedure

Sculpt is limited to capturing geometric features with the available resolution of the selected overlay grid. Because of this, care should be taken in selecting an appropriate cell size or mesh resolution for the input mesh. When a structured or Cartesian grid is used as the base mesh, adaptivity can be applied to refine the overlay grid to better capture features. In addition, the default mode will not attempt to capture sharp exterior features. Figure 1-2 shows an example of a sculpt mesh of a CAD model. Note that exterior corner features are rounded, however the effect of sharp feature capture becomes less pronounced as resolution increases as demonstrated in figures 1-3(a) and (b). Current research and development efforts include capabilities to incorporate sharp features, but is presently limited in its application.

Another aspect of model preparation for computational simulation involves geometry cleanup and simplification. In most cases, geometry-first methods, such as Sweeping, require an accurate

(a)   (b)
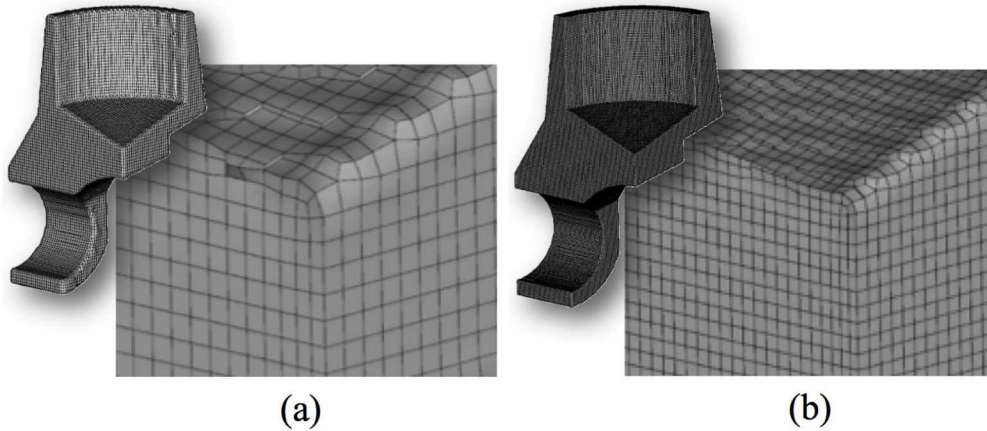
Figure 1-3. Examples of the same model meshed at two different resolutions showing a cutaway view of the mesh

non-manifold boundary representation before mesh generation can begin. Small, sometimes unseen gaps, overlaps and misalignments can result in sliver elements or mesh failure. Tedious manual geometry simplification and manipulation is often required before meshing can commence. Sculpt, however employs a solution that avoids much of the geometry inaccuracy issues inherent in CAD design models. Using a faceted representation of the solid model, a voxel-based volume fraction representation is generated. Figure 1-4 illustrates the procedure where a CAD model serving as input (figure 1-4(a)) is processed by a procedure that will generate volume fraction scalar data for each cell of an overlay Cartesian grid (figure 1-4(b)). One value per material per cell is computed that represents the volume fraction of material filling the cell. A secondary geometry representation is then extracted using an interface tracking technique from which the final hex mesh is generated (figure 1-4(c)). While similar to its initial facet-based representation, the new secondary geometry description developed from the volume fraction data results in a simplified model that tends to wash over small features and inaccuracies that are smaller than the resolution of the base cell size.

(a)   (b)   (c)
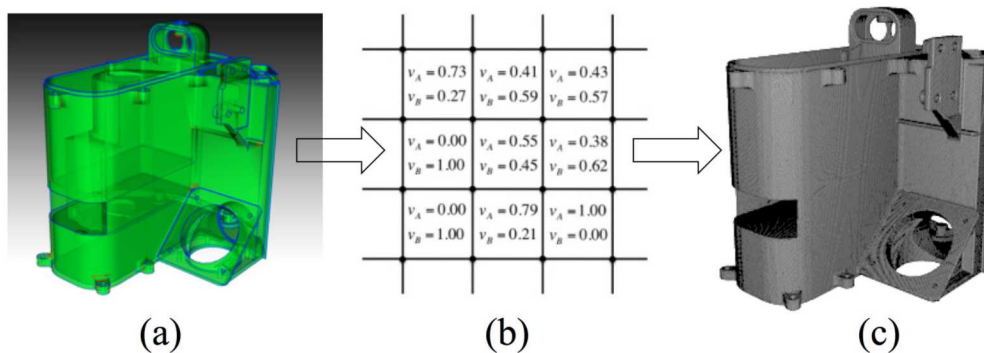
Figure 1-4. A representation of the procedure used to generate a hex mesh with Sculpt using Volume Fractions

While acknowledging some loss in model fidelity in this new volume-fraction based geometric model, the advantage and time-savings to the analyst of being able to ignore troublesome geometry issues is enormous. At the same time it may be important to understand what the additional discrete

approximations will make to solution accuracy and employ relevant engineering judgement in the use of this technology.

1.2.   LITERATURE REVIEW

Overlay-grid methods for generating hexahedral meshes have been available in the literature since the early 1990's. We provide a summary of some the main external contributions and publications. We also provide a list of publications by the authors that describe the methods used by Sculpt in more detail.

1.2.1.   Publications by External Authors

The following is a summary of some of external publication on overlay-grid methods with a brief description of their contribution. Full references are available at the end of this report:

- *Octree-based Generation of Hexahedral Element Meshes* by Schneiders et. al. [15]: First introduction of overlay grid methods as a solution for automatic hex meshing. Also introduces template-based refinement for octree adaptivity.

- *Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates* by Ito et. al. [2]: Extends the use of templates for octree refinement.

- *Adaptive hexahedral mesh generation based on local domain curvature and thickness using a modified grid-based method* by H. Zhang and Zhao [18]: Introduces mesh sizing for grid-based methods that includes geometric features such as curvature, proximity and local mesh size.

- *Adaptive and Quality Quadrilateral/Hexahedral Meshing from Volumetric Data* by Y. Zhang and Bajaj [19]: Introduces the dual contouring approach to build the hexahedral mesh from volumetric data.

- *Automatic 3D Mesh Generation for a Domain with Multiple Materials* by Y. Zhang et. al. [20]: Extends the dual contouring approach to capture internal features from volumetric data.

- *Constrained mesh optimization on boundary* by Yin and Teodosiu [17]: Also introduces dual contouring and feature capture methods for overlay grids.

- *Topologic and Geometric Constraint-based Hexahedral Mesh Generation* [16]: Introduces buffer layers at boundaries to improve element quality and concept of topological equivalence for feature capture.

- *Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features* by Marechal [3]: Overlay-grid method introduced for mechanical objects. Commercial tool *Hexa* by Distene Inc. is based on this approach.

- *Isotropic Conforming Refinement of Quadrilateral and Hexahedral Meshes Using Two-Refinement Templates* by Ebeida et. al. [1]: Introduces two-refinement methods to adapt overlay grids.

- *A Robust 2-Refinement Algorithm in Octree and Rhombic Dodecahedral Tree Based All-Hexahedral Mesh Generation* by Y. Zhang et. al. [21]: Extension of 2-refinement methods for overlay grids.

1.2.2.   Publications by the Authors

The Sculpt application presented in this report is the result of significant research by many people at Sandia over the past several years. The following technical papers, written by the authors of Sculpt, describe the overlay grid methods used in Sculpt. These papers were presented at the International Meshing Roundtable and other venues and most are available on the website `http://imr.sandia.gov`. Full references are included at the end of this report.

- *Parallel Hex Meshing from Volume Fractions* [12][13]: Describes the basic algorithms and mathematics used in the Sculpt procedure.

- *Embedding Features in a Cartesian Grid* [9]: Proposes new methods for capturing features from a Cartesian grid for hexahedral overlay-grid methods.

- *Parallel Smoothing for Grid-Based Methods* [4]: A brief description of the smoothing procedures used in Sculpt.

- *Validation of Grid-Based Hex Meshes with Computational Solid Mechanics* [7][8]: Describes a study where computational results from Sculpt meshes are compared with Sweep meshes using the Sierra Solid Mechanics Tool as a comparison.

- *Degenerate Hex Elements* [6]: Introduces use of a degeneracies or collapsed edges in hexahedral elements in a FEA mesh along with their implementation in Sculpt.

- *A Template-Based Approach for Parallel Hexahedral Two-Refinement* [10][11]: Describes the refinement procedures used for generating adapted Sculpt meshes.

- *Hexahedral Mesh Generation for Computational Materials Modeling* [5]: New developments to Sculpt for computational materials modeling

2. RUNNING SCULPT FROM CUBIT

Although Sculpt is a command-line application, separate from Cubit, it can use Cubit's graphical user interface as a front-end to operate sculpt. Cubit includes a command-line and GUI panel to drive sculpt. The cubit command-line, described in this chapter will build the appropriate input files to run an external sculpt process to mesh the current geometry. When the sculpt parallel command is executed in Cubit, sculpt will automatically generate the mesh externally to Cubit and bring the mesh back to Cubit.

A few practical examples of generating meshes with sculpt from Cubit are included in Appendix O

2.1. PREPARING TO USE SCULPT

2.1.1. Platforms

Sculpt is available for Windows, Mac and Linux operating systems.

2.1.2. Sculpt Installation

Sculpt is a stand-alone executable, separate from Cubit. In order for Cubit to start up Sculpt, it must be on your system and accessible to Cubit. The default installation of Cubit should install files in the correct locations for this to occur. Check with Cubit support if it did not come with your installation or you are not able to locate it or any of its supporting applications

To run Sculpt from Cubit, four executable files are needed:

- `sculpt`: Application that controls start-up of mpiexec and psculpt. Main entry point from Cubit, that checks for the existence and compatibility of either the system mpiexec application or will use a local cubit installation of mpiexec.

- `psculpt`: The main mpi-based Sculpt application. Requires mpiexec to run.

- `mpiexec`: Standard application available on most linux-based operating systems for starting up mpi-based applications on multiple processors. This should be available with your Cubit installation, but is also available from open-mpi.org

- epu : Used for combining multiple exodus files, generated with Sculpt, into a single exodus file. This executable is optional, but is useful for importing the resulting mesh into Cubit for viewing. It is part of the SEACAS tool suite developed by Sandia National Laboratories and is also included with your Cubit installation. It can also be obtained in open source form from sourceforge.net.

To view the current path to these executables that Cubit will use, issue the following command from the Cubit command window

```
Sculpt Parallel Path List
```

See the Sculpt Parallel Path Command for more info on setting and customizing these paths.

The following image illustrates the process flow when the sculpt parallel command is used in Cubit.
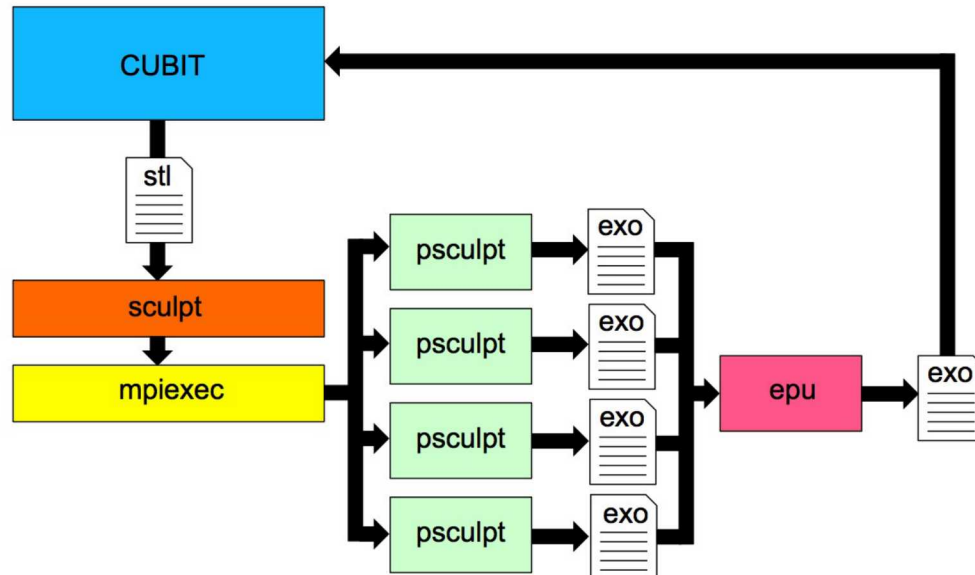
Figure 2-1. Sculpt process flow when invoked from Cubit

For the Sculpt meshing process, a set of files, including a facet-based stl file are written to disk. The sculpt application is then started up which in turn invokes mpiexec to start up multiple instances of psculpt in parallel. psculpt then performs the meshing and writes one exodus file per processor. These files can then be combined using epu and then imported back into Cubit for viewing.

2.1.3. Setting your Working Directory

When using the Sculpt Parallel command in Cubit, several temporary files will be written to the current working directory. Because of this, it is important to set your working directory before using Sculpt to a desired location where you want these files placed.

2.2.   SCULPT PARALLEL COMMAND

The command syntax for preparing a model for Sculpt is:

```
Sculpt Parallel [[volume <ids>] [block <ids>]]
  [processors <value>][fileroot '<root filename>']
  [exodus '<exodus filename>'][{OVERWRITE|no_overwrite}]
  [absolute_path][{EXECUTE|no_execute}]
  [size <value>|autosize <value>]
  [box {align | location <options>|expand <value>}]
  [smooth <value>][csmooth <value>]
  [num_laplace <value>][max_opt_iters <value>]
  [opt_threshold <value>][curve_opt_thresh <value>]
  [max_pcol_iters <value>][pcol_threshold <value>]
  [max_deg_iters <value>][deg_threshold <value>]
  [xintervals <value>][yintervals <value>][zintervals <value>]
  [gen_sideset <value>][{void|NO_VOID}][void_block <value>]
  [stair <value>][htet <value>][pillow <value>]
  [adapt_type <value>][adapt_threshold <value>][adapt_levels <value>]
  [scale <value>][xtranslate <value>][ytranslate <value>][ztranslate <value>]
  [{COMBINE|no_combine}][{IMPORT|no_import}][{SHOW|no_show}]
  [{capture|NO_CAPTURE}][{CLEAN|no_clean}]
  [{gen_input_file <filename>|no_gen_input_file}]
  [debug <value>]
```

The following tables 2-1 to 2-3 are a summary of options that can be invoked from the Cubit sculpt parallel command. It includes an abbreviated description of the option as well as the option's default. If the option is not explicitly used in the command, the default value listed will be used. The Sculpt option is the corresponding command that can be used in a sculpt input file when sculpt is invoked directly from an operating system terminal window.

2.3.   CONTROLLING THE EXECUTION OF SCULPT

The following command options can be used to control the execution of Sculpt from within Cubit and can be used with the `sculpt parallel` command.

```
volume <ids> | block <ids>
```

Table 2-1. Summary of Sculpt Cubit Command Options

Cubit Option	Description	Default	Sculpt Option
`volume <ids>` | `block <ids>`	List of volumes or blocks to include in the mesh.	Volume all	`stl_file, diatom_file`
`processors <value>`	Number of processors to use for meshing.	4	`num_procs`
`fileroot '<root filename>'`	Root of file names for output.	sculpt_parallel	
`exodus '<exodus filename>'`	Output Exodus mesh file name	<'root filename'>	`exodus_file`
`OVERWRITE` | `no_overwrite`	Force overwrite of files in directory	overwrite	
`absolute_path`	Use absolute path for filenames	OFF (relative path)	
`EXECUTE` | `no_execute`	Run sculpt or dump input files only	execute	
`size <value>` | `autosize <value>`	Set size of cells in Cartesian grid	autosize 10	`cell_size`
`xintervals <value> yintervals <value> zintervals <value>`	Number of cells in each coordinate direction in the overlay Cartesian grid	automatically computed from size	`nelx, nely, nelz`
`box align`	Automatically align geometry to grid	OFF	`align`
`box location <options>`	Define bounds of the Cartesian grid	Enclosing bounding box with 2.5 additional cells on each side	`xmin, ymin, zmin, xmax, ymax, zmax`
`box expand <value>`	Define Cartesian grid by expansion percentage from a tight bounding box.	OFF	`bbox_expand`
`smooth <value>`	Smoothing method for volumes and surfaces	1	`smooth`
`csmooth <value>`	Smoothing method for curves	5	`csmooth`
`num_laplace <value>`	Number of Laplacian smoothing iterations	2	`laplacian_iters`
`max_opt_iters <value>`	Maximum number of parallel Jacobi optimization iterations	5	`max_opt_iters`

Table 2-2. Summary of Sculpt Cubit Command Options (continued)

Cubit Option	Description	Default	Sculpt Option
opt_threshold <value>	Stopping criteria for Jacobi optimization smoothing	0.6	opt_threshold
curve_opt_thresh <value>	Metric at which curves are not honored	0.1	curve_opt_thresh
max_pcol_iters <value>	Maximum number of parallel coloring smoothing iterations	100	max_pcol_iters
pcol_threshold <value>	Stopping criteria for parallel color smoothing	0.2	pcol_threshold
max_deg_iters <value>	Maximum number of degenerate iterations	0	max_deg_iters
deg_threshold <value>	Convert hexes below threshold to degenerates	0.2	deg_threshold
gen_sidesets <value>	Sideset and nodeset generation method	0	gen_sidesets
void	Mesh void	OFF	mesh_void
void_block	Block ID of void mesh	0	void_mat
stair <value>	Generate Stair-step mesh	OFF	stair
htet <value>	Convert hexes below quality threshold to tets	-1	htet
pillow <value>	Set pillowing criteria	0	pillow
adapt_type <value>	Adaptive meshing type	0	adapt_type
adapt_threshold <value>	Threshold for adaptive meshing	$\frac{cell\_size}{4 \cdot adapt\_levels^2}$	adapt_threshold
adapt_levels <value>	Number of levels of adaptive refinement	2	adapt_levels
scale <value>	Scale mesh by factor	1.0	scale
xtranslate <value> ytranslate <value> ztranslate <value>	Translate mesh in coordinate directions	0.0	xtranslate, ytranslate, ztranslate
COMBINE | no_combine	Combine Exodus mesh files into a single mesh for import	combine	
IMPORT | no_import	Import the mesh after mesh generation in Sculpt	import	
SHOW | no_show	Echo the output of Sculpt to command line window	show	

Table 2-3. Summary of Sculpt Cubit Command Options (continued)

Cubit Option	Description	Default	Sculpt Option
`capture | NO_CAPTURE`	Project boundary nodes to STL geometry (beta feature)	`no_capture`	`capture`
`CLEAN | no_clean`	Delete temporary files generated during Sculpt run	`clean`	
`gen_input_file <file name> | no_gen_input_file`	Generate a Sculpt input file with current settings	`gen_input_file`	
`debug <value>`	Set a debug processor for debugging	`-1`	

List of volumes or blocks to include in the mesh. One file containing a faceted representation (STL) per volume will be generated and saved in the current working directory to be used as input to Sculpt. Each volume will be treated as a separate material within sculpt and a conforming mesh will be generated where volumes touch. If the Block command is used, one file per block will be used. Each block represents a separate material in Sculpt.

```
fileroot '<root filename>'
```

Root of file names for output. When the sculpt parallel command is executed, Cubit will generate multiple files in the working directory used for input to the Sculpt application. The '<root filename>' will be used as the basis for naming these files.

```
OVERWRITE | no_overwrite
```

By default, Cubit will overwrite an existing set of files with the same '<root filename>'. To over-ride, use the `no_overwrite` option.

```
absolute_path
```

By default, Cubit will write the relative path names of files used in the .run and .diatom files. To force absolute path names to be written, use the `absolute_path` option

```
EXECUTE | no_execute
```

By default, Cubit will attempt to run sculpt in parallel on the machine Cubit is currently running on. To generate just the required input to run Sculpt at a later time or on another machine, use this option. A file of the form <root filename>.run will be generated in the current working directory. (for example "model.run"). Executing the .run file from the linux command line should run sculpt in parallel. It can also be used to run sculpt on a cluster where a Cubit executable may not be available.

```
size <value> | autosize <value>
```

The `size` option is the absolute cell size for the Cartesian grid and is the same as the `cell_size` option in sculpt. The `autosize` option is a value between 0 and 10. It represents a model independent size where 1 is the small size and 10 is large. This is the same scaling factor used in Cubit's auto sizing but is divided by ten. A size value will be computed from the selected autosize and used as the absolute cell size for the base Cartesian grid.

```
box location <options>
```

`Location` options define the bounds of the Cartesian grid. The first Location <option> defines the minimum Cartesian coordinate of the grid and the second, the maximum. The <options> can be any valid method for defining a coordinate location in cubit. In most cases the `position` option can be used. The default is computed as an enclosing bounding box with 2.5 additional cells on each side.

```
COMBINE | no_combine
```

If the `no_combine` option is used, following execution of Sculpt, the resulting exodus meshes will not be combined using the `epu seacas` tool. Otherwise the default will automatically combine the meshes generated by each processor into a single mesh. Note that epu should be installed on your system and the path to epu defined using the `sculpt parallel path` command. Epu is a code developed by Sandia National Laboratories and is part of the SEACAS tool suite. It combines multiple Exodus databases produced by a parallel application into a single Exodus database. The epu program should be included with distributions of Cubit beginning with Version 15.0.

```
IMPORT | no_import
```

If the `no_import` option is used, following execution of Sculpt, the result will be not be imported into Cubit as a free mesh. The default `IMPORT` option will automatically import the mesh that was generated in Sculpt. If the `no_combine` option has been used, then multiple free meshes will be imported with duplicate nodes and faces at processor domain boundaries. Otherwise a single free mesh, the result of the `epu` code, will be imported. Note that the resulting mesh will not be associated with the original geometry, however Block (material) definitions will be maintained. In addition, a separate group will be generated for each imported mesh (One per processor). The default will automatically import the mesh following mesh generation in Sculpt.

```
SHOW | no_show
```

If the `no_show` option is used, while the external Sculpt process is running, no output from the Sculpt application will be displayed to the command window. Otherwise, the default `SHOW` is used and output from the Sculpt application will be echoed to the Cubit command window. This option is only effective if the `no_execute` is not used.

```
CLEAN | no_clean
```

If the `clean` option is used, temporary files generated during the sculpt parallel command will be deleted. This includes any exodus mesh files, .stl, .diatom, .log and .run files. The default for this option is CLEAN, therefore, use the `no_clean` option to keep any temporary files generated as part of the current Sculpt run.

```
gen_input_file <file name> | no_gen_input_file
```

An input file with the given file name will be generated when the command is executed. This is a text file containing all sculpt options used in the command. The input file is intended to be used for batch execution of sculpt. To run sculpt from the operating system command line you would use the -i option. For example: `sculpt -i myinputfile.i -j 4` where `myinputfile.i` is the name of the input file specified with the `gen_input_file` option and `-j 4` is the number of processors to use.

```
debug <value>
```

The debug option is used only as a developer debugging tool. It will set the debug processor and sleep upon execution to allow a debugger to be attached to the process.

2.4.   SCULPT PARALLEL PATH COMMAND

The command for letting Cubit know where the Sculpt and related applications are located is:

```
Sculpt Parallel Path [List|Psculpt|Epu|Mpiexec]
```

This command defines the path to `psculpt, epu` and `mpiexec` on your system. In most cases, however, these paths should be automatically set provided Sculpt was successfully installed with your Cubit installation. The `list` option will list the current paths that Cubit will use for these tools. If an alternate path to these executables is desired, it is recommended that this command be used in the .cubit initialization file so that it wont be necessary to define these parameters every time Cubit is run.

2.5. SCULPT MESH QUALITY CONTROL

In most cases, the Sculpt tool can be used without adjusting default values. Depending on the characteristics of the geometry to be meshed, the default values may not yield adequate mesh quality. Upon completion, Sculpt reports to the command line, a summary of the mesh that was generated. This includes a summary of the mesh quality. Care should be taken to review this summary to ensure the minimum mesh quality is in a range suitable for analysis.

The element metric used for computing mesh quality in Sculpt is the Scaled Jacobian. This is a value between -1 and 1 that is a relative measure of the angles at the element's nodes. A value of 1 indicates a perfect 90 degree angle between each of its edges. In most cases a value less than zero, or negative Jacobian element, indicates an unusable mesh. Sculpt's default settings try to achieve a minimum Scaled Jacobian of 0.2, which is normally usable in most analysis. The following discussion provides several options for adjusting the model or Sculpt parameters to help improve mesh quality.

1. *Locating poor mesh quality:* When the Sculpt mesh has been imported back into CUBIT it is a good idea to display the element quality. You can do this with variations of the following commands:

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

2. *Modifying the geometry:* Zooming in to poor quality elements may reveal that the mesh does not adequately represent the underlying geometry. In some cases you may find that small features, or small gaps between parts may be on the order of the size of the Sculpt cell size. If these features are not important to the analysis, you may consider using Cubit's geometry modification tools to remove features or close small gaps.

3. *Modifying the cell size:* In cases where small geometric features or gaps are important to the simulation, it may be necessary to use a smaller base cell size. Use the size or autosize input parameters or increase the numbers of intervals. Normally to adequately capture a feature you would want the cell size to be no greater than about 1/3 to 1/2 the size of the smallest feature you would want to represent in the simulation.

4. *Turning on Pillowing for multiple materials:* For models that have more than one material that share an interface, unless the geometry is precisely aligned with the global axis, it is usually a good idea to turn on pillowing. Pillowing automatically inserts an additional layer of hexes at interface boundaries to improve mesh quality. Without pillowing may notice inverted or poor quality elements at curve interfaces where 2 or more materials meet.

5. *Modifying smoothing parameters:* Sculpt 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. Default numbers of iterations and thresholds for each smoothing phase have been tuned for general use, however it may be worthwhile to adjust these parameters. The three smoothing phases include:

a) Laplacian Smoothing: Applied to all elements. Very inexpensive fast approach to improve quality, but can result in degraded element quality if applied to excess. A fixed default of 2 iterations is applied to all hexes. Increasing the `num_laplace` parameter can improve some cases, especially convex shapes.

b) Optimization Smoothing: Applied only to elements who's scaled Jacobian falls below the `opt_threshold` parameter (default 0.6) and their surrounding elements. This approach uses a more expensive optimization technique to improve regions of elements simultaneously. The `max_opt_iters` parameter can control the maximum number of iterations applied (default is 5). Iterations will terminate, however, if no further improvement is detected. Because this method optimizes node locations simultaneously, neighboring nodes with competing optimum can sometimes limit mesh quality.

c) Spot Optimization: Also known as parallel coloring, is applied only to elements who's element quality falls below the `pcol_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 using the `max_pcol_iters` to achieve desired results. Increasing the `pcol_threshold` and `max_pcol_iters` may yield improved results.

Observing the mesh quality output to the command line following each smoothing iteration can provide some insight on the effect of modifying smoothing parameters.

6. *Creating degenerate hexes:* Some geometries will not permit a usable mesh with a traditional all-hex mesh. Sculpt includes the option to automatically and selectively collapse element edges to improve low-quality elements. The `max_deg_iters` and the `deg_threshold` values are used to control the creation of degenerates. Degenerate elements are treated as standard hex elements, but use repeated nodes in the eight-node connectivity array.

7. *Creating hex-dominant mesh:* Another option for avoiding mesh quality issues is to generate a few tet elements in the mesh using the `htet` option. With this option you can specify a scaled Jacobian threshold value below which hexes will be converted to tet elements. The interface between hex and tet elements is managed by an automatically defined set of nodesets and sidesets that describe where multi-point constraints will be applied.

8. *Defeaturing:* The defeature option does an initial filter on the cells of the base grid and attempts to reassign the material ID for cells that meet certain criteria. These are cases where a small grouping of cells form a small volume, or where protrusions exist that would otherwise be difficult or impossible to mesh with good quality elements. By reassigning the cells in these locations, in many cases it will allow the mesh to be acceptable. This operation may result in small changes to the boundary or surface definitions, however usually small enough to still be a reasonable approximation.

3. SCULPT APPLICATION

This chapter describes the Sculpt application, a separate companion application to Cubit designed to run in parallel for generating all-hex meshes of complex geometry. Sculpt was developed as a separate application so that it can be run independently from Cubit on high performance computing platforms. It was also designed as a separable software library so it can be easily integrated as an in-situ meshing solution within other codes. As installed with Cubit, Sculpt can be set up and run directly from Cubit, in a batch process from the unix command line or from a user-defined input file. This chapter describes the input file and command line syntax for the Sculpt Application when running in batch mode.

Two examples of running sculpt from the operating system command line are included in Appendix P. Appendix Q also includes example geometry and diatom files that can be used for the examples.

3.1.   SCULPT SYSTEM REQUIREMENTS

Sculpt is currently built for windows, linux and mac operating systems. Current supported OS versions should be the same as those supported by Cubit. It is designed to take advantage of 64 bit multicore and distributed memory computers, using open-mpi as the basis for parallel communications.

3.2.   RUNNING SCULPT

Sculpt can be run using one of two executables:

- `psculpt`: requires the use of `mpiexec` to start the process. Number of processors to use is specified by the `-np` argument to `mpiexec`. `psculpt` and its input parameters are also used as input to `mpiexec`. For example:

    ```
    mpiexec -np 8 psculpt -stl myfile.stl -cs 0.5
    ```

    If appropriate system paths have not been set, you may need to use full paths when referring to `mpiexec` and `psculpt`.

- `sculpt`: This application assumes that `mpiexec` is included in the standard CUBIT installation directory. The number of processors to use is specified by the `-j` option. For example:

    ```
    sculpt -j 8 -stl myfile.stl -cs 0.5
    ```

If the `-j` option is not used, sculpt will default to a single processor for execution. The `-mpi` option can also be used with the sculpt application to indicate a specific mpi installation that is not included with CUBIT. For example:

```
sculpt -j 8 -mpi /path/to/mpiexec -stl myfile.stl -cs 0.5
```

If the path specified by the `-mpi` option does not exist or the mpi version is incompatible, sculpt will attempt to use the local CUBIT-installed `mpiexec` or else the system `mpiexec` in the PATH environment.

3.3. SCULPT HELP

Help on Sculpt input syntax and command descriptions are available by using the `-h` or `--help` options on the command line. For example:

```
sculpt -h
```

will display the command summary shown in section 3.4. This shows all commands available in Sculpt along with a brief description. For a full description of specific commands, use the command after the `-h`. For example:

```
sculpt -h diatom_file
```

will display details about the `diatom_file` option. Note that the documentation contained in this report will be the same as that displayed with the `-h` or `--help` options.

3.4. SCULPT COMMAND SUMMARY

Following is a listing of the available input commands to either sculpt or psculpt. When used from the unix command line, commands may be issued using the short form argument, designated with a single dash(-), or with the longer form, designated with two dashes (–). When used in an input file, only the long form may be used, omitting the two dashes (–). The following chapters describe these options in more detail

```
Usage: psculpt [options]
  --help,-h <args>Displays this information

Process Control  --process  -pc
  --num_procs           -j   <arg> Number of processors requested
  --input_file          -i   <arg> File containing user input data
  --debug_processor     -D   <arg> Sleep to attach to processor for debug
  --debug_flag          -dbf <arg> Dump debug info based on flag
  --quiet               -qt        Suppress output
  --print_input         -pi        Print input values and defaults then stop
  --version             -vs        Print version number and exit
  --threads_process     -tpp <arg> Number of threads per process
  --iproc               -ip  <arg> Number of processors in I direction
  --jproc               -jp  <arg> Number of processors in J direction
  --kproc               -kp  <arg> Number of processors in K direction
  --periodic            -per        Generate periodic mesh
  --check_periodic      -cp  <arg> Check for periodic geometry
  --periodic_axis       -pax <arg> Axis periodicity is about
  --periodic_nodesets   -pns <arg> Nodesets ids of master/slave nodesets
  --build_ghosts        -bg         Write ghost layers to exodus files for debug
  --vfrac_method        -vm  <arg> Set method for computing volume fractions

Input Data Files  --input  -inp
  --stl_file            -stl <arg> Input STL file
  --diatom_file         -d   <arg> Input Diatom description file
  --input_vfrac         -ivf <arg> Input from Volume Fraction file base name
  --input_micro         -ims <arg> Input from Microstructure file
  --input_cart_exo      -ice <arg> Input from Cartesian Exodus file
  --input_spn           -isp <arg> Input from Microstructure spn file
  --spn_xyz_order       -spo <arg> Ordering of cells in spn file
  --lattice             -l   <arg> STL Lattice Template File

Output  --output  -out
  --exodus_file         -e   <arg> Output Exodus file base name
  --volfrac_file        -vf  <arg> Output Volume Fraction file base name
  --quality             -Q         Dump quality metrics to file
  --export_comm_maps    -C         Export parallel comm maps to debug exo files
  --write_geom          -G         Write geometry associativity file
  --write_mbg           -M         Write mesh based geometry file <beta>
  --compare_volume      -cv        Report vfrac and mesh volume comparison

Overlay Grid Specification  --overlay  -ovr
  --nelx                -x   <arg> Num cells in X in overlay Cartesian grid
  --nely                -y   <arg> Num cells in Y in overlay Cartesian grid
  --nelz                -z   <arg> Num cells in Z in overlay Cartesian grid
```

```
--xmin                    -t   <arg> Min X coord of overlay Cartesian grid
--ymin                    -u   <arg> Min Y coord of overlay Cartesian grid
--zmin                    -v   <arg> Min Z coord of overlay Cartesian grid
--xmax                    -q   <arg> Max X coord of overlay Cartesian grid
--ymax                    -r   <arg> Max Y coord of overlay Cartesian grid
--zmax                    -s   <arg> Max Z coord of overlay Cartesian grid
--cell_size               -cs  <arg> Cell size (nelx, nely, nelz ignored)
--align                   -a         Automatically align geometry to grid
--bbox_expand             -be  <arg> Expand tight bbox by percent
--input_mesh              -im  <arg> Input Base Exodus mesh
--input_mesh_blocks       -imb <arg> Block ids of Input Base Exodus mesh
--input_mesh_material     -imm <arg> Material definition with input mesh
--input_mesh_pamgen       -imp <arg> Input Base mesh defined by Pamgen

Mesh Type  --type  -typ
  --stair                 -str <arg> Generate Stair-step mesh
  --mesh_void             -V   <arg> Mesh void
  --htet                  -ht  <arg> Convert hexes below quality threshold to tets
  --trimesh               -tri       Generate tri mesh of geometry surfaces
  --tetmesh               -tet <arg> Under Development
  --deg_threshold         -dg  <arg> Convert hexes below threshold to degenerates
  --max_deg_iters         -dgi <arg> Maximum number of degenerate iterations
  --htet_material         -htm <arg> Convert hexes in given materials to tets
  --htet_transition       -htt <arg> Transition method between hexes and tets
  --htet_pyramid          -htp <arg> Local transition pyramid
  --htet_tied_contact     -htc <arg> Local transition tied contact
  --htet_no_interface     -htn <arg> Local transition none

Boundary Conditions  --boundary_condition  -bc
  --void_mat              -VM  <arg> Void material ID (when mesh_void=true)
  --gen_sidesets          -SS  <arg> Generate sidesets
  --free_surface_sideset  -FS  <arg> Free Surface Sideset
  --match_sidesets        -mss <arg> Sidesets ids of matching pairs

Adaptive Meshing  --adapt  -adp
  --adapt_type            -A   <arg> Adaptive meshing type
  --adapt_threshold       -AT  <arg> Threshold for adaptive meshing
  --adapt_levels          -AL  <arg> Number of levels of adaptive refinement
  --adapt_export          -AE        Export exodus mesh of refined grid

Smoothing  --smoothing  -smo
  --smooth                -S   <arg> Smoothing method
  --csmooth               -CS  <arg> Curve smoothing method
  --laplacian_iters       -LI  <arg> Number of Laplacian smoothing iterations
  --max_opt_iters         -OI  <arg> Max. number of parallel Jacobi opt. iters.
```

```
--opt_threshold         -OT  <arg> Stopping criteria for Jacobi opt. smoothing
--curve_opt_thresh      -COT <arg> Min metric at which curves won't be honored
--max_pcol_iters        -CI  <arg> Max. number of parallel coloring smooth iters.
--pcol_threshold        -CT  <arg> Stopping criteria for parallel color smooth
--max_gq_iters          -GQI <arg> Max. number of guaranteed quality smooth iters.
--gq_threshold          -GQT <arg> Guaranteed quality minimum SJ threshold

Mesh Improvement  --improve  -imp
--pillow                -p   <arg> Set pillow criteria (1=surfaces)
--pillow_surfaces       -ps        Turn on pillowing for all surfaces
--pillow_curves         -pc        Turn on pillowing for bad quality at curves
--pillow_boundaries     -pb        Turn on pillowing at domain boundaries
--pillow_curve_layers   -pcl <arg> Number of elements to buffer at curves
--pillow_curve_thresh   -pct <arg> S.J. threshold to pillow hexes at curves
--pillow_smooth_off     -pso       Turn off smoothing following pillow operations
--capture               -c   <arg> Project to facet geometry <beta>
--capture_angle         -ca  <arg> Angle at which to split surfaces <beta>
--capture_side          -sc  <arg> Project to facet geometry with surface ID
--defeature             -df  <arg> Apply automatic defeaturing
--min_vol_cells         -mvs <arg> Minimum number of cells in a volume
--defeature_bbox        -dbb       Defeature Filtering at Bounding Box
--defeature_iters       -dfi <arg> Maximum Number of Defeaturing Iterations
--thicken_material      -thm <arg> Expand a given material into surrounding cells
--micro_expand          -me  <arg> Expand Microstructure grid by N layers
--micro_shave           -ms        Remove isolated cells at micro. boundaries
--remove_bad            -rb  <arg> Remove hexes with Scaled Jacobian < threshold

Mesh Transformation  --transform  -tfm
--xtranslate            -xtr <arg> Translate final mesh coordinates in X
--ytranslate            -ytr <arg> Translate final mesh coordinates in Y
--ztranslate            -ztr <arg> Translate final mesh coordinates in Z
--scale                 -scl <arg> Scale final mesh coordinates by constant

Boundary Layers  --boundary_layer  -bly
--begin                 -beg <arg> Begin specification blayer or blayer_block
--end                   -zzz <arg> End specification blayer or blayer_block
--material              -mat <arg> Boundary layer material specification
--num_elem_layers       -nel <arg> Number of element layers in blayer block
--thickness             -th  <arg> Thickness of first element layer in block
--bias                  -bi  <arg> Bias of element thicknesses in blayer block

Use --help <args> or -h <args> to display detailed help on any option.
Use "all" argument to display help for all options.
```

4. PROCESS CONTROL

Options for controlling the execution of Sculpt. Sculpt is a parallel application that uses MPI to distribute and build the hex mesh on multiple processors. The -j or num_procs option is normally used to specify the number of processors to use. Sculpt will write a separate exodus file for each processor, which can be joined into a single file using the epu utility. While any number of processors may be used, you would normally use a -j value less than or equal to the number of cores available on your hardware.

Sculpt options can be specified directly from the command line using the "short" commands, or from an input file where the longer forms of the commands are used. Since an input file can be commented and modified, it is generally the recommended method for running Sculpt.

```
Process Control  --process  -pc
  --num_procs           -j   <arg> Number of processors requested
  --input_file          -i   <arg> File containing user input data
  --debug_processor     -D   <arg> Sleep to attach to processor for debug
  --debug_flag          -dbf <arg> Dump debug info based on flag
  --quiet               -qt        Suppress output
  --print_input         -pi        Print input values and defaults then stop
  --version             -vs        Print version number and exit
  --threads_process     -tpp <arg> Number of threads per process
  --iproc               -ip  <arg> Number of processors in I direction
  --jproc               -jp  <arg> Number of processors in J direction
  --kproc               -kp  <arg> Number of processors in K direction
  --periodic            -per        Generate periodic mesh
  --check_periodic      -cp  <arg> Check for periodic geometry
  --periodic_axis       -pax <arg> Axis periodicity is about
  --periodic_nodesets   -pns <arg> Nodesets ids of master/slave nodesets
  --build_ghosts        -bg        Write ghost layers to exodus files for debug
  --vfrac_method        -vm  <arg> Set method for computing volume fractions
```

4.1.   NUMBER OF PROCESSORS

```
Command: num_procs       Number of processors requested

Input file command:   num_procs <arg>
Command line options: -j <arg>
Argument Type:        integer > 0
```

Command Description:

The number of processors that Sculpt will use to generate the mesh. The Cartesian domain will be divided into roughly equal sizes based on this value and the mesh for each portion of the domain generated independently. Continuity across processor boundaries is maintained with MPI (Message Passing Interface). Each processor will write a separate Exodus II file to disk containing its portion of the domain. The Sandia SEACAS tool, "EPU" can be used to join parallel files into a single file.

If not specified on the command line, the number of processors used will be 1.

For additional control on the arrangement of processor domains see arguments iproc, jproc, kproc.

4.2.   INPUT FILE

```
Command: input_file     File containing user input data

Input file command:   input_file <arg>
Command line options: -i <arg>
Argument Type:        file name with path
```

Command Description:

Rather than specifying a complicated series of arguments on the command line, an input file may also be used. An input file is a simple text file containing all arguments and parameters to be used in the current sculpt run. Input files are normally expected to have a ".i" extension. Arguments used in the input file are limited to the Long Names indicated for each command.

User comments can also be made anywhere in the file but must follow a "$" sign. The argument assignments that are intended to be read must be contained within a "begin sculpt" and "end sculpt" block. All arguments may use upper or lower case and can optionally use "=" between the command and its parameter. The following is an example input file:

```
BEGIN SCULPT
  stl_file = "mygeom.stl"
  cell_size = 0.5
  exodus_file = "mymesh"
  mesh_void = true
END SCULPT
```

The following is an example of using an input file with sculpt:

```
sculpt -j 4 -i myinput.i
```

Note that the number of processors (-j) should always be used on the command line and cannot be included in the input file. Relative or absolute paths for files may also be used.

4.3.   DEBUG PROCESSOR

```
Command: debug_processor      Sleep to attach to processor for debug

Input file command:    debug_processor <arg>
Command line options: -D <arg>
Argument Type:         integer >= 0
```

Command Description:

Used for debugging. All processes will sleep until the designated process is attached to a debugger. Note: value of 0 corresponds to first processor, 1 to second, etc.

4.4.   DEBUG FLAG

```
Command: debug_flag      Dump debug info based on flag

Input file command:    debug_flag <arg>
Command line options: -dbf <arg>
Argument Type:         integer >= 0
```

Command Description:

Used for debugging. Set flag to dump specific info based on the following:

0 Default, No debug output

1 Dump processor lost node info

2 Export Non-manifold resolution state as exodus file after each inner and outer iteration.

3 Export Defeature state as exodus file after each inner and outer iteration.

4 Export the Thickened state as exodus file after each material has been thickened.

Guaranteed Quality:

5 Turn off initial minimizer projection.

6 Use Non-manifold reversal case

7 Combine debug_flag 5 and 6

8 Use guaranteed quality laplacian color smoothing

9 Combine debug_flags 5,6 and 8

4.5.   QUIET

```
Command: quiet     Suppress output
```

```
Input file command:   quiet
Command line options: -qt
```

Command Description:

Suppress any output to the command line from Sculpt as it is running.

4.6.   PRINT INPUT

```
Command: print_input     Print input values and defaults then stop
```

```
Input file command:   print_input
Command line options: -pi
```

Command Description:

Display all input parameters and defaults used in the current Sculpt run to the output window and then stop. No mesh (or volume fractions) will be generated.

4.7. VERSION

```
Command: version      Print version number and exit

Input file command:   version
Command line options: -vs
```

Command Description:

Prints Sculpt version information and exits.

4.8. THREADS PER PROCESSOR

```
Command: threads_process      Number of threads per process

Input file command:   threads_process <arg>
Command line options: -tpp <arg>
Argument Type:        integer > 0
```

Command Description:

This option is currently experimental and under development. Sculpt may use shared memory parallelism to improve performance. When built with the Kokkos library, some algorithms in sculpt will use shared memory parallel threads in addition to MPI distributed memory parallelism (MPI+X). Currently this option is implemented only for surface and volume Laplacian smoothing algorithms. This option may not be available requiring a custom build of sculpt to be used. Check with developers if you would like to use this option.

4.9. NUMBER OF PROCESSORS IN I

```
Command: iproc      Number of processors in I direction

Input file command:   iproc <arg>
Command line options: -ip <arg>
Argument Type:        integer > 0
```

Command Description:

Arguments iproc, jproc and kproc provide user control over the processor decomposition in I, J, and K directions respectively. iproc * jproc * kproc must equal the number of processors specified on the command line using the -j option.

4.10. NUMBER OF PROCESSORS IN J

```
Command: jproc     Number of processors in J direction

Input file command:    jproc <arg>
Command line options: -jp <arg>
Argument Type:        integer > 0
```

Command Description:

Arguments iproc, jproc and kproc provide user control over the processor decomposition in I, J, and K directions respectively. iproc * jproc * kproc must equal the number of processors specified on the command line using the -j option.

4.11. NUMBER OF PROCESSORS IN K

```
Command: kproc     Number of processors in K direction

Input file command:    kproc <arg>
Command line options: -kp <arg>
Argument Type:        integer > 0
```

Command Description:

Arguments iproc, jproc and kproc provide user control over the processor decomposition in I, J, and K directions respectively. iproc * jproc * kproc must equal the number of processors specified on the command line using the -j option.

4.12. GENERATE PERIODIC MESH

Command: periodic   Generate periodic mesh

Input file command: periodic
Command line options: -per

Command Description:

Generates a periodic mesh for either Cartesian or unstructured mesh input. Ensures that resulting mesh nodes and faces are precisely matching on opposite sides of the mesh.

Unstructured mesh input: When used with the -input_mesh option opposite sides of the mesh must be identified using pairs of leading and trailing nodesets using the -periodic_nodesets (-pns) option. Nodes in the nodeset pairs must be separated by a constant translation or rotation. If a rotation is used between leading and trailing nodesets, the -periodic_axis (-pax) option must be used. If not used, then the transformation is assumed to be pure translation. Input geometry is assumed to be periodic with a period equal to that of the input mesh. Results from non-periodic geometry used with the periodic option may be unpredictable. The following is an example of an input file that uses the periodic option on an unstructured input mesh:


```
BEGIN SCULPT
    diatom_file = geometry_file.diatom
    input_mesh = input_exodus_file.g
    exodus_file = output_exodus_file
    smooth = to_geometry
    capture = 5
    capture_angle = 10
    free_surface_sideset = 1000
    gen_sidesets = input_mesh_and_free_surfaces
    periodic = true
    periodic_nodesets = 3224 3225
    periodic_axis = 0 0 0 0 1 0
END SCULPT
```


Cartesian grid input: This option is often used for computational materials modeling. Sculpt can generate a true periodic mesh in a representative volume element (RVE) where meshes on all opposite faces of the RVE will precisely match. When used with a Cartesian grid, the -periodic_nodesets and -periodic_axis options are ignored. The following is an example sculpt input file that utilizes the -periodic option on a Cartesian grid with geometry defined in a diatom file. It also utilizes the -adapt_type option to automatically refine and the gen_sidesets = RVE option to generate sidesets at the six RVE faces.

```
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
    periodic = true
    defeature = 1
    min_vol_cells = 10
    adapt_type = vfrac_average
    adapt_levels = 2
    adapt_threshold = 0.00001
    gen_sidesets = RVE
    exodus_file = spheres_periodic
    mesh_void = true
END SCULPT
```

Geometry Requirements: In order to generate a valid periodic mesh, the input geometry must also be periodic and the bounding box parameters should span exactly one period of the geometry. To check the periodicity of the geometry and prescribed bounding box, see the check_periodic option.

Note: The resulting mesh at the boundaries of the Cartesian grid (RVE) will not be projected to the planes of the bounding box. The result will be a "ragged" boundary in order to maintain periodicity between nodes on opposite sides of the mesh. Also note that results from the use of the periodic option may be undefined or unstable when used with non-periodic input geometry.
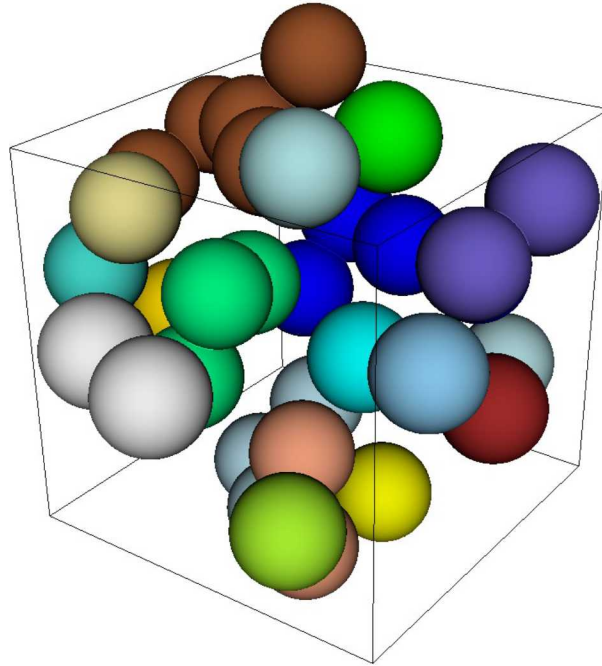
Figure 4-1. Periodic geometry used for example described in di-atom file. RVE boundary shown with respect to the geometry.
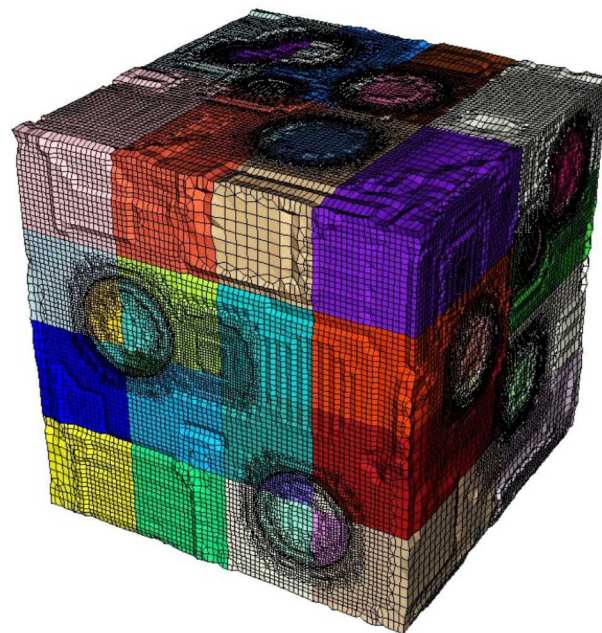
Figure 4-2. Resulting periodic mesh generated from example input.
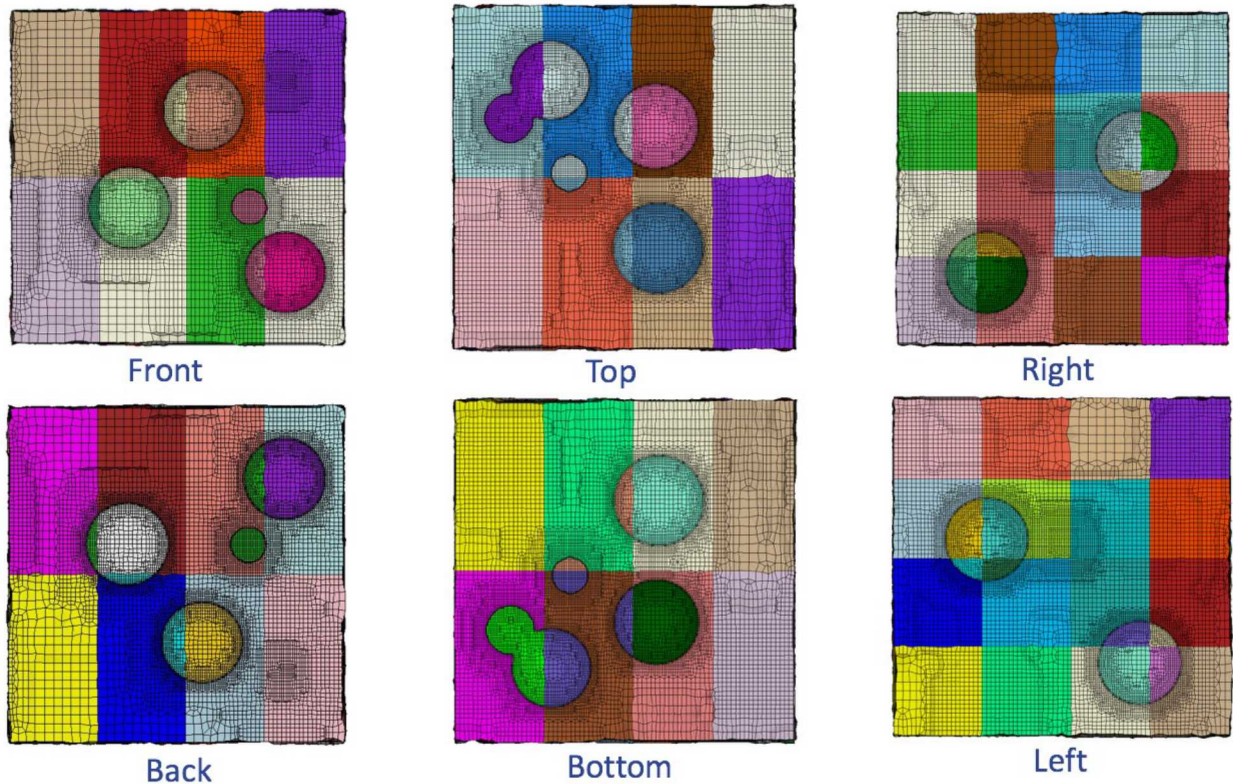
Figure 4-3. Six faces of the RVE from above example illustrating periodicity on a 32 processor decomposition. Note that top three images are a mirror image of the bottom three images.

4.13. CHECK FOR PERIODIC GEOMETRY

```
Command: check_periodic      Check for periodic geometry

Input file command:    check_periodic <arg>
Command line options: -cp <arg>
Argument Type:         on, off, only
Input arguments: off (0)
                 on (1)
                 only (2)
```

Command Description:

When using the `periodic` option with a Cartesian base grid, the input geometry must be periodic with respect to the grid bounding box in order to meet the minimum requirements of a valid periodic mesh. The bounding box must span exactly one period in each dimension. If this requirement is not met, a

valid mesh may still be generated, however, periodicity will not be guaranteed. The `check_periodic` option is used to check this requirement.

Options:

- `ON`: The `check_periodic` option is ON by default to ensure periodicity is enforced. Sculpt will fail if the geometry and bounding box do not meet the requirements for periodicity.

- `OFF`: Turning this option OFF will by-pass this check and attempt to generate the mesh even if periodic requirements are not met.

- `ONLY`: The ONLY option will perform a check for periodic requirements and report diagnostics. An exodus file (or files) will be produced with the name "check_periodic.o.o.x.x". A stair-step mesh of the domain will be produced with an additional block 999 showing where periodicity is not matched. Sculpt will immediately stop execution after producing the "check_periodic.o.o.x.x" mesh. Note that 2 additional layers on all sides of the Cartesian grid will be present in the mesh. These are used internally in Sculpt for parallel ghosting.

The `check_periodic` option is ignored if the `periodic` option is OFF or set to false.

4.14. PERIODIC MESH AXIS

```
Command: periodic_axis     Axis periodicity is about

Input file command:   periodic_axis <arg>
Command line options: -pax <arg>
Argument Type:        six floating point values
```

Command Description:

For an unstructured base grid, specifies an axis about which the nodes in the master (leading) nodesets will be rotated about to produce the slave (trailing) nodesets. Six floating point numbers are specified, the first three define the origin of the axis and the last three define the axis direction. This option must be used with `-periodic` (-per), `-periodic_nodesets` (-pns), and `-input_mesh` (-im) options. If the `-periodic` (-per) option is used without the –periodic_axis option, the transformation between leading and trailing nodesets is assumed to be pure translation.

4.15. PERIODIC NODESET IDS

Command: `periodic_nodesets`   Nodesets ids of master/slave nodesets

```
Input file command:    periodic_nodesets <arg>
Command line options: -pns <arg>
Argument Type:         integer(s) >= 0
```

Command Description:

For an unstructured base grid, specifies the master-slave (leading-trailing) nodeset pairs. Master nodesets should be able to be translated or rotated about a specified axis to produce the nodes in the slave nodesets. Nodesets must be specified in pairs, where each master (leading) nodeset corresponds to a single slave (trailing) nodeset. Each nodeset pair must maintain an identical translation or rotation. If a rotation is used, the axis and origin of rotation must be specified with the `–periodic_axis` (-pax) option. This option should be used with `–periodic` (-per), `–periodic_nodesets` (-pns), and `–input_mesh` (-im) options.)
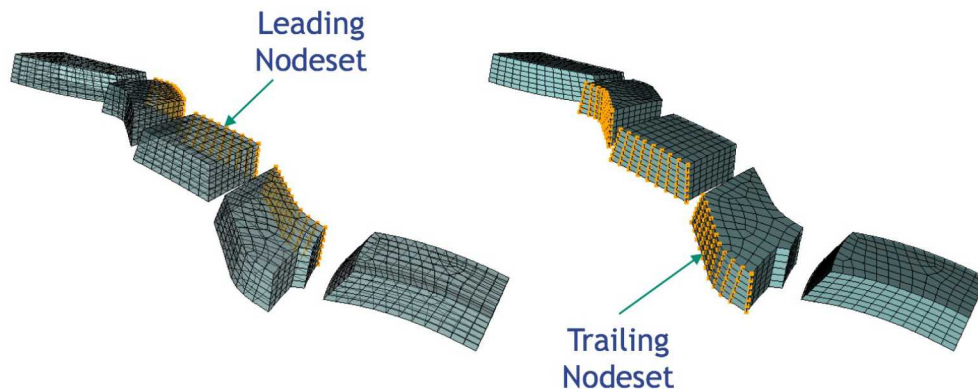
Figure 4-4. Unstructured input mesh used to generate periodic mesh. Matching leading and training nodesets are defined in the exodus file.

4.16. WRITE THE GHOST LAYERS FOR DEBUG

Command: `build_ghosts`   Write ghost layers to exodus files for debug

```
Input file command:    build_ghosts
Command line options: -bg
```

44

Command Description:

If set, this option will dump the ghost hexes at the boundaries of processor domains to the exodus files. This is used only for debugging.

4.17. VOLUME FRACTION CALCULATION METHOD

```
Command: vfrac_method      Set method for computing volume fractions

Input file command:    vfrac_method <arg>
Command line options: -vm <arg>
Argument Type:         integer (1, 2)
Input arguments: cth (0)
                 cth (1)
                 r3d (2)
```

Command Description:

Sets the method used for computing volume fractions from geometry input. Two options are currently available:

CTH (1): The default method. It uses the CTH third party library from Sandia Laboratories for approximating intersections using an adaptive ray firing method to determine inside-outside status of multiple locations within a grid cell. This method can be used with STL and all valid primitive types defined by the diatom format.

R3D (2): Uses the R3D third party library developed by Los Alamos Laboratories. Machine precision intersection calculations are performed to generate accurate volume fractions from the STL description. This method is valid for STL and diatom input packages specifying STL input files. Non STL format geometry defined in the diatom file will be ignored for this format.

5. INPUT DATA FILES

Options for specifying input files to Sculpt. Sculpt uses a method for representing geometry based upon volume fractions defined on a Cartesian or unstructured grid. Sculpt will accept facet-based (STL) or analytic (diatom) geometry, but will first convert the input geometry to the required volume fraction description before generating the hexahedral mesh. Various formats for volume fraction data can also be imported directly into Sculpt and used as the basis for hex meshing. The following formats for geometry are currently supported in Sculpt:

- STL (stl_file)

- Diatom (diatom_file)

- Volume Fractions

    - Exodus element variables (import_vfrac)

    - Exodus blocks (import_cart_exo)

    - Ascii Voxel Data (import_spn)

    - Ascii Volume Fraction Data (import_micro)

```
Input Data Files   --input  -inp
  --stl_file              -stl <arg> Input STL file
  --diatom_file           -d   <arg> Input Diatom description file
  --input_vfrac           -ivf <arg> Input from Volume Fraction file base name
  --input_micro           -ims <arg> Input from Microstructure file
  --input_cart_exo        -ice <arg> Input from Cartesian Exodus file
  --input_spn             -isp <arg> Input from Microstructure spn file
  --spn_xyz_order         -spo <arg> Ordering of cells in spn file
  --lattice               -l   <arg> STL Lattice Template File
```

5.1. STL FILE

```
Command: stl_file      Input STL file

Input file command:   stl_file <arg>
Command line options: -stl <arg>
Argument Type:        file name with path
```

Command Description:

File name of a single STL (facet geometry) file to be used as input. Either an stl_file or diatom_file designation should be included to run Sculpt. The stl_file option will support a single STL file. To use multiple STL files, where each file represents a different material, use the diatom_file file option where multiple file names may be specified.

It is recommended that STL files used as input to Sculpt be "water-tight". While in many cases non-watertight geometries will be successful, unexpected or incorrect results may result. It is recommended practice to use Cubit to first import the STL geometry and allow the sculpt parallel command to write a new STL geometry file for use in Sculpt. Cubit's sculpt parallel command will attempt to stitch and repair any triangle facets that are not completely closed. Other commercial tools are available for STL geometry that may be effective in repairing the geometry prior to use in Sculpt.

5.2. DIATOM FILE

```
Command: diatom_file      Input Diatom description file

Input file command:   diatom_file <arg>
Command line options: -d <arg>
Argument Type:        file name with path
```

Command Description:

File name of a diatom file to be used as input to Sculpt. Both stl_file and diatom_file cannot be used simultaneously. A diatom file is a constructive solid geometry description containing primitives for generating a full geometric definition of the model. Diatoms are commonly used as input to Sandia's CTH and Alegra codes. Multiple STL files can also be defined in a Diatom file. The following is a simple example of a diatom file that would read 3 different STL files:

```
diatoms
  package 'blue_material'
    material 1
    insert stl
      file = 'blue_part1.stl'
```

```
      endinsert
      insert stl
        file = 'blue_part2.stl'
      endinsert
    endpackage
    package 'red_material'
      material 2
      insert stl
        file = 'red_part1.stl'
      endinsert
    endpackage
  enddiatom
```

Note that the first two files, blue_part1.stl and blue_part2.stl belong to the same material. As a result, elements generated within the geometry of these files will belong to block 1. Likewise, the elements generated within the geometry of red_part1.stl will belong to block 2.

`Bitmap Files`

The Diatom format will also support bitmap files. These are binary files that set each cell either on or off for the specified material. The following is an example diatom specification for a bitmap file. Note that the bitmap specification includes nx, ny, nz dimensions for the size of the input file.

```
    diatoms
      package 'Skull'
        material 1
        insert bitmap
          file = 'skull_bitmap_file'
          nx = 680
          ny = 408
          nz = 236
        endinsert
      endpackage
    enddiatom
```
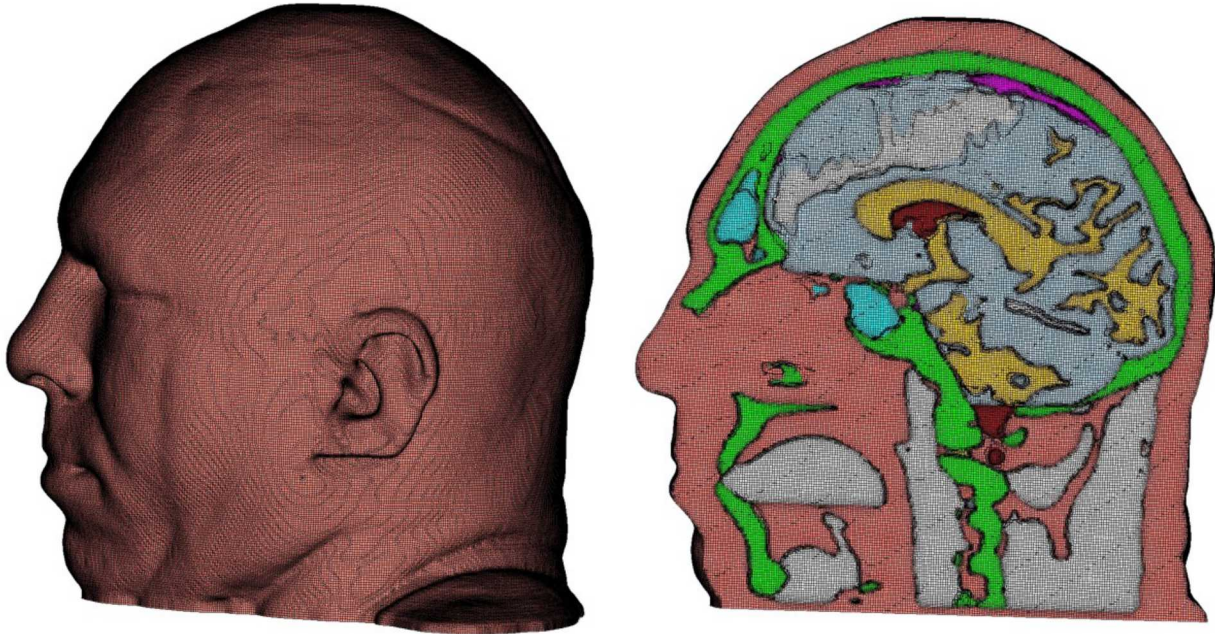
Figure 5-1. Example mesh generated with Diatom bitmap option

For a full description of the diatom format see the CTH or Alegra documentation.

D. A. Crawford, A. L. Brundage, E. N. Harstad, K. Ruggirello, R. G. Schmitt, S. C. Schumacher and J. S. Simmons, "CTH UserâĂŹs Manual and Input Instructions, Version 10.3", CTH Development Project, Sandia National Laboratories, Albuquerque, New Mexico 87185, February 14, 2013

5.3. INPUT VOLUME FRACTION FILE

```
Command: input_vfrac      Input from Volume Fraction file base name

Input file command:    input_vfrac <arg>
Command line options:  -ivf <arg>
Argument Type:         base file name with path
```

Command Description:

Sculpt can optionally take an exodus file containing volume fraction data stored as element variables. Normally the exodus file has initially been written using the –volfrac_file (-vf) option. Since the exodus file will be a Cartesian grid spread across multiple processors, the base filename for the parallel series of exodus files is used as the argument for this command. The input volume fraction file(s) would be used instead of an STL or diatom file. Since computing volume fractions from geometry can be time consuming, precomputing the volume fractions and reading them from a file can be advantageous if multiple meshes are to be generated from the same volume fraction data.

5.4.   INPUT MICROSTRUCTURE FILE

```
Command: input_micro      Input from Microstructure file
```

```
Input file command:    input_micro <arg>
Command line options: -ims <arg>
Argument Type:         file name with path
```

Command Description:
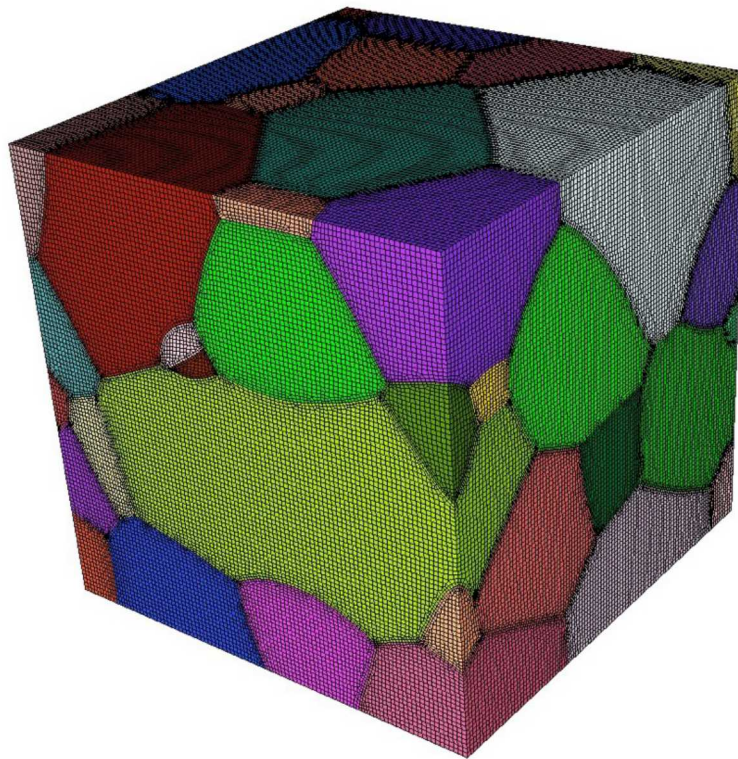
Figure 5-2. Example all-hex mesh of microstructure

A microstructure file is an ascii text file containing volume fraction data for each cell of a Cartesian grid. The format for this file includes header information followed by data for each cell. The following is an example:

```
TITLE = triple line system
VARIABLES = x y z, phi_1, phi_2, phi_3
ZONE i = 2 , j = 2 , k = 2
0.0000        0.0000        0.0000        0.5000        0.5000        0.0000
```

1.0000	0.0000	0.0000	0.3333	0.3333	0.3334
0.0000	1.0000	0.0000	1.0000	0.0000	0.0000
1.0000	1.0000	0.0000	0.0000	1.0000	0.0000
0.0000	0.0000	1.0000	0.2000	0.4000	0.4000
1.0000	0.0000	1.0000	0.6000	0.1000	0.3000
0.0000	1.0000	1.0000	0.0000	0.0000	1.0000
1.0000	1.0000	1.0000	0.9000	0.0000	0.1000

The header information should contain the following:

TITLE: any descriptive character string

VARIABLES: a list of variables separated by spaces or commas. It should include x, y, z as the first three variable names. The remaining names are arbitrary. The number of variable names listed must correspond to the number of data values for each cell of the Cartesian grid.

ZONE: Specify the number of cells in the i, j and k directions (corresponding to x, y, and z respectively)

The body of the file will contain one line per cell of the grid. The first three values correspond to the centroid location of a cell in the grid. The remaining values represent volume fractions for the cell for each variable listed. The sum of the volume fractions for each individual cell should be 1.0

Currently this format assumes that cell sizes are exactly 1.0 x 1.0 x 1.0 and the minimum cell centroid location is always 0.0, 0.0, 0.0. This results in a Cartesian grid with minimum coordinate = (-0.5, -0.5, -0.5) and maximum coordinate = (i-0.5, j-0.5, k-0.5). If a size other than 1x1x1 is required consider using the scale and/or translate options.

Example usage of this command is as follows:

```
sculpt -j 8 -ims my_micro_file.tec -p 1
```

Smoothing: Sculpt will set automatic defaults for smoothing if user options have not been defined. These include:

```
--smooth 9 (surface smoothing option - no surface projection)
--csmooth 2 (curve smoothing option - hermite interpolation)
```

These options will generally provide a smoother curve and surface representation but may not adhere strictly to the volume fraction geometric definition. To over-ride the defaults, consider using the following options:

```
--smooth 8 (surface smoothing option - projection to interpolated surface)
--csmooth 5 (curve smoothing option - projection to interpolated curve)
```

Pillowing: For most 3D models it is recommended using pillowing since triple junctions (curves with at least 3 adjacent materials) will typically be defined where malformed hex elements would otherwise be generated. Surface pillowing (option 1) is usually sufficient to remove poor quality elements at triple junctions.

Figure 5-3. Pillows (hex layers) inserted at surfaces to improve element quality around curves. Note mesh quality at curve between surfaces A and B.

5.5. INPUT CARTESIAN EXODUS FILE

```
Command: input_cart_exo      Input from Cartesian Exodus file

Input file command:    input_cart_exo <arg>
Command line options: -ice <arg>
Argument Type:         file name with path
```

Command Description: An exodus mesh containing a Cartesian grid of elements can also be used as the source of a sculpt mesh. For this option the following conditions must be met:
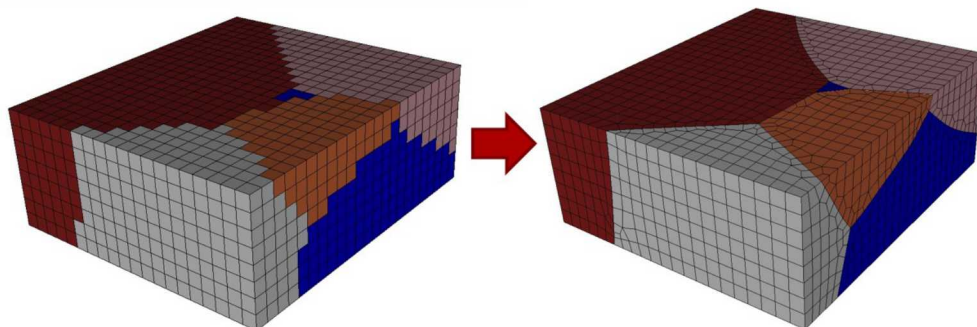
Figure 5-4. Example Cartesian Exodus file and the resulting hex mesh.

1. A single (non-parallel) exodus II format file.

2. Contains only hex elements configured as a Cartesian grid.

3. All hex elements must be exactly equilateral cubes.

4. Each hex element has been assigned to exactly one block. (Any number of blocks may be defined in the file)

Provided these conditions are met, sculpt will treat each block as a separate material and generate a smooth conforming mesh between the materials. This option is useful for converting a stair-step mesh into a smooth conforming mesh. The resulting sculpt mesh will have the same dimensions as the original exodus mesh, but will add layers of hexes at material interfaces.

Example usage of this command is as follows:

```
sculpt -j 8 -ice my_cartesian_file.e -p 1
```

Smoothing: Sculpt will set automatic defaults for smoothing if user options have not been defined. These include

```
--smooth 9 (surface smoothing option - no surface projection)
--csmooth 2 (curve smoothing option - hermite interpolation)
```

These options will generally provide a smoother curve and surface representation but may not adhere strictly to the volume fraction geometric definition. To over-ride the defaults, consider using the following options:

```
--smooth 8 (surface smoothing option - projection to interpolated surface)
--csmooth 5 (curve smoothing option - projection to interpolated curve)
```

Pillowing: For most 3D models it is recommended using pillowing since triple junctions (curves with at least 3 adjacent materials) will typically be defined where malformed hex elements would otherwise be generated. Surface pillowing (option 1) is usually sufficient to remove poor quality elements at triple junctions.

5.6. INPUT MICROSTRUCTURE SPN FILE

```
Command: input_spn      Input from Microstructure spn file


Input file command:    input_spn <arg>
Command line options: -isp <arg>
Argument Type:          file name with path
```

Command Description:

A .spn file is an optional method for importing volume fraction data into sculpt for meshing. This format is a simple ascii text file containing one integer per cell of a Cartesian grid. Each integer represents a unique material identifier. Any number of materials may be used, however for practical purposes, the number of unique materials should not exceed more than about 50 for reasonable performance.

An example file containing a 3 x 3 x 3 grid with 2 materials may be defined as follows:

```
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
```

Any unique integer may be used to identify a material. All cells with the same ID will be defined as a continuous block with the same exodus block ID in the final mesh. All integers should be separated by a space or newline. The number of integers in the file should exactly correspond to the size of the Cartesian grid. The dimensions of the Cartesian grid must be specified on the command line as part of the input. The following is an example:

```
sculpt -j 8 -x 10 -y 24 -z 15 -isp "my_spn_file.spn" -p 1
```

The default order of the cells in the input file will be read according to the following schema:

```
for (i=0; i<nx; i++)
  for (j=0; j<ny; j++)
    for (k=0; k<nz; k++)
        // read next value from file
```

Where nx, ny, nz are the number of cells in each Cartesian direction. This ordering can be changed to nz, ny, nx using the spn_xyz_order option. The initial size of the Cartesian grid will be exactly nx X ny X nz with the minimum coordinate at (0.0, 0.0, 0.0). If a size other than the default is required, consider using the scale and/or translate options.

Smoothing: Sculpt will set automatic defaults for smoothing if user options have not been defined. These include:

```
--smooth 9 (surface smoothing option - no surface projection)
--csmooth 2 (curve smoothing option - hermite interpolation)
```

These options will generally provide a smoother curve and surface representation but may not adhere strictly to the volume fraction geometric definition. To over-ride the defaults, consider using the following options:

```
--smooth 8 (surface smoothing option - projection to interpolated surface)
--csmooth 5 (curve smoothing option - projection to interpolated curve)
```

Pillowing: For most 3D models it is recommended using pillowing since triple junctions (curves with at least 3 adjacent materials) will typically be defined where malformed hex elements would otherwise be generated. Surface pillowing (option 1) is usually sufficient to remove poor quality elements at triple junctions.

5.7.   XYZ ORDERING OF CELLS IN SPN FILE

```
Command: spn_xyz_order     Ordering of cells in spn file

Input file command:    spn_xyz_order <arg>
Command line options: -spo <arg>
Argument Type:        integer (0 to 5)
Input arguments: xyz (0)
                 xzy (1)
                 yxz (2)
                 yzx (3)
                 zxy (4)
                 zyx (5)
```

Command Description:

This option is valid with the 'input_spn' option. The default order of the cells in the spn input file will be read according to the following schema:

```
for (i=0; i<nx; i++)
  for (j=0; j<ny; j++)
    for (k=0; k<nz; k++)
        // read next value from file
```

If the spn file has the cells in a different order, use this option to specify the order. 0 (xyz) is the default.

5.8.   STL LATTICE TEMPLATE FILE

```
Command: lattice       STL Lattice Template File

Input file command:   lattice <arg>
Command line options: -l <arg>
Argument Type:        file name with path
```

Command Description:
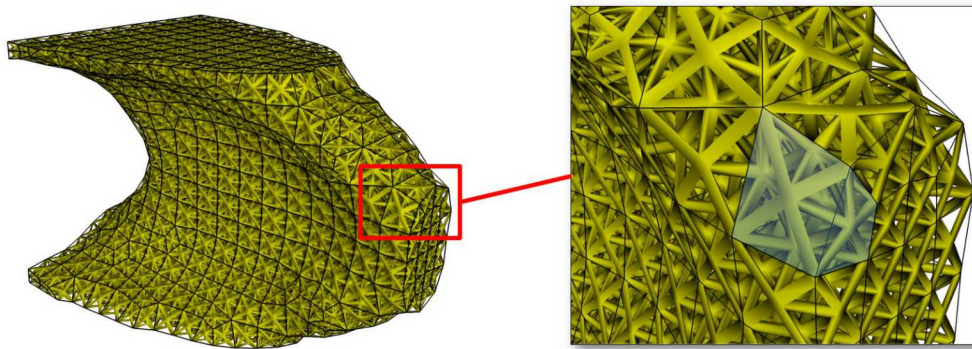
Figure 5-5. Lattice geometry generated from exodus mesh.

Generate a lattice structure from a hex mesh. This command takes the name of an STL format template file which defines the lattice over a unit cube. To generate a valid lattice structure, the facets should be symmetric to the three coordinate planes. The lattice structure will be transformed and copied into each hex of the mesh. The result will be an STL file containing lattice geometry for the mesh.

This option currently requires the name of an exodus mesh on which to define the lattice. Use the –exodus_file (-e) option to specify its path. The current implementation is limited to one block, however if a second block is contained in the Exodus file it will be treated as a solid and stl facets will be generated at the skin of the block.

The name of the output STL file may also be defined by using the –stl_file (-stl) option. If no stl file is specified, the output will use the name of the input exodus file with the extension "_lattice.stl" appended.

In addition to the full lattice geometry, an additional file containing only the lattice from the first layer of hexes will be written. This may be useful in reducing the size of the STL file for visualization purposes only. The name of this file will be the name of the full STL geometry file with the extension ".vis.stl" appended.

The following is an example input file using the lattice option:

```
BEGIN SCULPT
  lattice = lattice_template.stl  $contains unit cube with triangles
  exodus_file = file.e $ hex mesh containing one or two element blocks
  stl_file = file.stl $ name of output stl file
END SCULPT
```

Note that this option is currently limited to serial execution (-j 1)

6. OUTPUT

Sculpt options for specifying output. The primary format for the hex meshes produced from Sculpt is Exodus II. One exodus file will be produced for each processor based upon the -j or num_procs argument. If required, the exodus files can be joined using the epu utility.

Other options for export include the ability to dump the volume fraction representation of the input geometry as well as the ability to write geometry files for use in Cubit.

```
Output  --output  -out
  --exodus_file             -e   <arg> Output Exodus file base name
  --volfrac_file            -vf  <arg> Output Volume Fraction file base name
  --quality                 -Q         Dump quality metrics to file
  --export_comm_maps        -C         Export parallel comm maps to debug exo files
  --write_geom              -G         Write geometry associativity file
  --write_mbg               -M         Write mesh based geometry file <beta>
  --compare_volume          -cv        Report vfrac and mesh volume comparison
```

6.1.   EXODUS FILE

```
Command: exodus_file    Output Exodus file base name

Input file command:   exodus_file <arg>
Command line options: -e <arg>
Argument Type:        character string
```

Command Description:

The base file name of the resulting exodus mesh. Exodus files will be in the form <exodus_file>.e.<nproc>.<iproc>. For example, if the number of processors used is 3 and the exodus_file argument is "model" the following files would be written:

```
    model.e.3.0
    model.e.3.1
    model.e.3.2
```

If no exodus_file argument is used, output files will be in the form
<stl_file>_diatom_results.e.<nprocs>.<iproc>. For example, if the number of processors used is 3 and the stl_file (or diatom_file) is "model.stl", the following files would be written:

```
model_diatom_results.e.3.0
model_diatom_results.e.3.1
model_diatom_results.e.3.2
```

A full path may be used when specifying the base exodus file name, otherwise files will be placed in the current working directory. If the exodus_file option is not used, exodus files will be placed in the same directory as the input diatom or stl file.

6.2.　VOLUME FRACTION FILE

```
Command: volfrac_file     Output Volume Fraction file base name

Input file command:   volfrac_file <arg>
Command line options: -vf <arg>
Argument Type:        character string
```

Command Description:

Optionally generate exodus files containing a hex mesh of the Cartesian grid containing volume fraction data as element variables. This series of parallel exodus files can later be used as direct input to sculpt using the –input_vfrac (-ivf) command. If not specified, no volume fraction data files will be generated.

6.3.　QUALITY

```
Command: quality     Dump quality metrics to file

Input file command:   quality
Command line options: -Q
```

Command Description:

A file named 'quality.csv' will be created in the current working directory (or appended). Quality metrics and other details of the run will be written to this file. This option is currently off by default.

6.4.   EXPORT COMMUNICATION MAPS

```
Command: export_comm_maps      Export parallel comm maps to debug exo files
```

```
Input file command:   export_comm_maps
Command line options: -C
```

Command Description:

Used for debugging and verification. Exodus files of the mesh containing the communication nodes and faces at processor boundaries will be written as nodes and side sets. This provides a way to visually check the validity of the parallel communication maps.

6.5.   WRITE S2G GEOMETRY FILE

```
Command: write_geom      Write geometry associativity file
```

```
Input file command:   write_geom
Command line options: -G
```

Command Description:

An s2g (Sculpt to Geometry) file, with the pattern <fileroot>.s2g, will be produced when this argument is used where fileroot is the string specified by the –exodus_file or -e option. An s2g file includes geometry associativity for the exodus file that is written. If used with Cubit's "import s2g <fileroot>" a mesh-based geometry will be generated in Cubit with geometric entities prescribed by Sculpt through the s2g file.

When used with the –trimesh option, the s2g file can provide information to Cubit to build a set of mesh-based geometry volumes where only the surfaces are meshed. This is useful for using the tet meshing capabilities in Cubit to mesh the discrete geometry that was generated in Sculpt. For example, a tet mesh may be constructed from microstructures spn data (see import_spn) with the following workflow:

1. Run Sculpt to generate an exodus and s2g file. An example input file may look like the following:

```
begin sculpt
  import_spn = myfile.spn
  trimesh = true
  write_geom = true
  pillow = 1
end sculpt
```

2. Import the file into Cubit to generate a mesh based geometry:

```
import s2g myfile
```

3. Delete the triangle mesh, set sizes and mesh:

```
delete mesh
vol all scheme tetmesh
vol all size 2.0
mesh vol all
```

Note that the write_geom and trimesh options are still in development and will currently only work with a single processor (-j 1).

6.6.   WRITE MESH BASED GEOMETRY

```
Command: write_mbg      Write mesh based geometry file <beta>

Input file command:    write_mbg
Command line options: -M
```

Command Description:

An MBG (Mesh Based Geometry) file will be produced when this argument is used with the pattern <fileroot>.mbg, where fileroot is the string specified by the –exodus_file or -e option. An MBG file includes the surface and topology definition defined by sculpt as a result of the interface reconstruction process. It will correspond to the boundary of the 3D elements that are generated in the exodus file, or the surface elements generated with the –trimesh option.

An MBG file can be be imported into Cubit using the following Cubit command line options:

```
import mbg "<fileroot>.mbg"
```

6.7.   REPORT VFRAC TO MESH VOLUME COMPARISON

```
Command: compare_volume      Report vfrac and mesh volume comparison

Input file command:    compare_volume
Command line options: -cv
```

Command Description:

A report will be generated and printed to the terminal following the mesh summary that compares the input volume fraction of the geometry with that of the final finite element mesh. If a volume fraction format is not used as input, the volume fractions will be computed on the refined base grid and used as comparison. Note that exact geometric volumes of the STL or analytic geometry are not used for comparison, rather the volume fraction approximation of the geometry on the refined Cartesian grid.

```
=============================== VOLUME COMPARISON ===============================
Block ID      Num Elems      Sum VFrac      Elem Vol        Diff        Percent Err      VFrac
1             1460156        863.819        868.16          4.34073     0.502504         0.159767
2             9171           3.095          2.90369        -0.191313    6.18135          0.000534363
3             148952         53.268         50.0251        -3.24293     6.08796          0.00920606
4             194233         117.063        115.3          -1.76334     1.50632          0.0212185
5             553            0.089          0.0697202      -0.0192798   21.6627          1.28305e-05
6             95367          37.029         37.0632         0.0341906   0.0923346        0.0068207
7             545329         176.477        162.542        -13.9348     7.89612          0.0299125
8             847440         493.112        491.576        -1.53645     0.311582         0.0904642
9             1397057        656.51         668.967         12.4568     1.89743          0.123109
10            1305491        912.571        912.232        -0.339128    0.0371618        0.167877
11            2947876        2121.99        2125.09         3.1019      0.146179         0.391078
--------------------------------------------------------------------------------
Total         8951625        5435.02        5433.93         1.09365     0.0201223        1
```

Figure 6-1. Example output from the `compare_volume` command.

The following is a brief description of each column:

- `Block ID`: ID of material/block

- `Num Elems`: Number of hex elements assigned to block in final mesh

- `Sum VFrac`: Sum of input volume fraction for block. For STL or diatom geometry, approximates the volume fraction. For 3D image data (ie. bitmap, input_spn) sums the exact volume fraction input.

- `Elem Vol`: Sum of final mesh volume for block

- `Diff`: Absolute difference between input and output volume fractions for block. (Elem Vol - Sum VFrac)

- `Percent Err`: Percent error represented by Difference between input and output volume fractions for block

- `VFrac`: Total volume fraction represented by Elem Vol. for block. VFrac volume should sum to 1.0.

7. OVERLAY GRID SPECIFICATION

Sculpt options for setting up the overlay grid. Sculpt is an overlay-grid method that requires a base mesh that it will modify to generate the final mesh. The base mesh can be in the form of a Cartesian grid, but can also be any general unstructured hexahedral mesh defined in an exodus file (see the `input_mesh` option). Pamgen can also be used to generate an unstructured base mesh (see `input_mesh_pamgen`).

When an overlay Cartesian grid is used as the basis for the all-hex mesh that will be produced, the bounds and size of the cells defining the grid must be specified. The Cartesian grid can be defined in one of two ways:

1. Define the bounding box and number of intervals in each coordinate direction. (`xmin`, `ymin`, `zmin`, `xmax`, `ymax`, `zmax`, `nelx`, `nely`, `nelz`)

2. Define a `cell_size`. Sculpt will then automatically define the Cartesian grid coordinates and intervals by evaluating the bounding box of the input geometry and adding a small number of cells in each coordinate direction.

Other options for setting up the Cartesian base grid include `align` and `expand` which are normally used with the second method. The `align` option will automatically rotate the grid to best match the characteristic direction of the geometry rather than maintaining alignment with the global Cartesian directions. The `expand` option over-rides the default expansion of the Cartesian grid beyond the bounding box of the geometry and allow the user to specify a specific expansion percentage.

```
Overlay Grid Specification  --overlay  -ovr
  --nelx              -x   <arg> Num cells in X in overlay Cartesian grid
  --nely              -y   <arg> Num cells in Y in overlay Cartesian grid
  --nelz              -z   <arg> Num cells in Z in overlay Cartesian grid
  --xmin              -t   <arg> Min X coord of overlay Cartesian grid
  --ymin              -u   <arg> Min Y coord of overlay Cartesian grid
  --zmin              -v   <arg> Min Z coord of overlay Cartesian grid
  --xmax              -q   <arg> Max X coord of overlay Cartesian grid
  --ymax              -r   <arg> Max Y coord of overlay Cartesian grid
  --zmax              -s   <arg> Max Z coord of overlay Cartesian grid
  --cell_size         -cs  <arg> Cell size (nelx, nely, nelz ignored)
  --align             -a         Automatically align geometry to grid
  --bbox_expand       -be  <arg> Expand tight bbox by percent
  --input_mesh        -im  <arg> Input Base Exodus mesh
```

```
--input_mesh_blocks    -imb <arg> Block ids of Input Base Exodus mesh
--input_mesh_material  -imm <arg> Material definition with input mesh
--input_mesh_pamgen    -imp <arg> Input Base mesh defined by Pamgen
```

7.1. NUMBER OF INTERVALS X

```
Command: nelx    Num cells in X in overlay Cartesian grid

Input file command:   nelx <arg>
Command line options: -x <arg>
Argument Type:        integer > 0
```

Command Description:

Defines the number of intervals in the x direction of the base Cartesian grid used for defining the volume fraction definition and meshing For best results the intervals specified should result in approximately equilateral cells.

See also nely, nelz

7.2. NUMBER OF INTERVALS Y

```
Command: nely    Num cells in Y in overlay Cartesian grid

Input file command:   nely <arg>
Command line options: -y <arg>
Argument Type:        integer > 0
```

Command Description:

Defines the number of intervals in the y direction of the base Cartesian grid used for defining the volume fraction definition and meshing For best results the intervals specified should result in approximately equilateral cells.

See also nelx, nelz

7.3.   NUMBER OF INTERVALS Z

Command: nelz   Num cells in Z in overlay Cartesian grid

Input file command: nelz <arg>
Command line options: -z <arg>
Argument Type: integer > 0

Command Description:

Defines the number of intervals in the z direction of the base Cartesian grid used for defining the volume fraction definition and meshing For best results the intervals specified should result in approximately equilateral cells.

See also nelx, nely

7.4.   XMIN BOUNDING BOX RANGE

Command: xmin   Min X coord of overlay Cartesian grid

Input file command: xmin <arg>
Command line options: -t <arg>
Argument Type: floating point value

Command Description:

Defines the minimum x coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also ymin, zmin, xmax, ymax, zmax.

7.5.   YMIN BOUNDING BOX RANGE

Command: ymin   Min Y coord of overlay Cartesian grid

Input file command: ymin <arg>
Command line options: -u <arg>
Argument Type: floating point value

Command Description:

Defines the minimum y coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also xmin, zmin, xmax, ymax, zmax.

7.6. ZMIN BOUNDING BOX RANGE

```
Command: zmin      Min Z coord of overlay Cartesian grid

Input file command:    zmin <arg>
Command line options: -v <arg>
Argument Type:         floating point value
```

Command Description:

Defines the minimum z coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also xmin, ymin, xmax, ymax, zmax.

7.7. XMAX BOUNDING BOX RANGE

```
Command: xmax      Max X coord of overlay Cartesian grid

Input file command:    xmax <arg>
Command line options: -q <arg>
Argument Type:         floating point value
```

Command Description:

Defines the maximum x coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also xmin, ymin, zmin, ymax, zmax.

7.8.   YMAX BOUNDING BOX RANGE

```
Command: ymax     Max Y coord of overlay Cartesian grid

Input file command:   ymax <arg>
Command line options: -r <arg>
Argument Type:        floating point value
```

Command Description:

Defines the maximum y coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also xmin, ymin, zmin, xmax, zmax.

7.9.   ZMAX BOUNDING BOX RANGE

```
Command: zmax     Max Z coord of overlay Cartesian grid

Input file command:   zmax <arg>
Command line options: -s <arg>
Argument Type:        floating point value
```

Command Description:

Defines the maximum z coordinate of the bounding box or range of the Cartesian mesh to be used for meshing.

See also xmin, ymin, zmin, xmax, ymax.

7.10.   CELL SIZE

```
Command: cell_size     Cell size (nelx, nely, nelz ignored)

Input file command:   cell_size <arg>
Command line options: -cs <arg>
Argument Type:        floating point value
```

Command Description:

Defines a target edge size for the cells of the base Cartesian grid. Both interval and cell_size can not be specified simultaneously. If cell_size is used without a range specification, a bounding box of the geometry will be computed and used as the default range

---

7.11. ALIGN

```
Command: align     Automatically align geometry to grid

Input file command:    align
Command line options: -a
```

Command Description:

The align option will attempt to orient the Cartesian grid with the main dimensions of the geometry. This is done by defining a tight bounding box around the geometry using an optimization procedure where the objective is to minimize the difference in volume between an enclosing box and the geometry. Using the align command will override any bounding box parameters previously entered and will build an "aligned" bounding box around the full geometry. It is currently only implemented for STL geometry and will ignore any other diatom definitions. Note that this option will also write temporary stl and diatom files to the working directory.

7.12. BOUNDING BOX EXPANSION FACTOR

```
Command: bbox_expand     Expand tight bbox by percent

Input file command:    bbox_expand <arg>
Command line options: -be <arg>
Argument Type:         floating point value
```

Command Description:

Sculpt will measure a tight bounding box of the input model and expand the box by the specified percentage in x, y and z. Input value can be any positive or negative floating point value where 1.0 represents 100 percent expansion. If not specified, the default will add about 2.5 cell widths to the bounding box on each side. This option should be used with the cell_size option. It will be ignored if a specific bounding box has been defined (ie. xmin, ymin, etc...).

7.13. INPUT BASE EXODUS MESH

Command: input_mesh   Input Base Exodus mesh

Input file command: input_mesh <arg>
Command line options: -im <arg>
Argument Type: file name with path

Command Description:

Option to import an Exodus file to use as the base mesh for Sculpt. Sculpt's meshing procedure requires a base mesh from which geometry is recovered and captured. The default base mesh is a Cartesian grid that is defined by specifying a bounding box and intervals. The input_mesh option permits a general hexahedral mesh to be used as the base mesh instead of a Cartesian grid. This option currently supports a serial and parallel Exodus files containing HEX8 elements with any number of blocks.
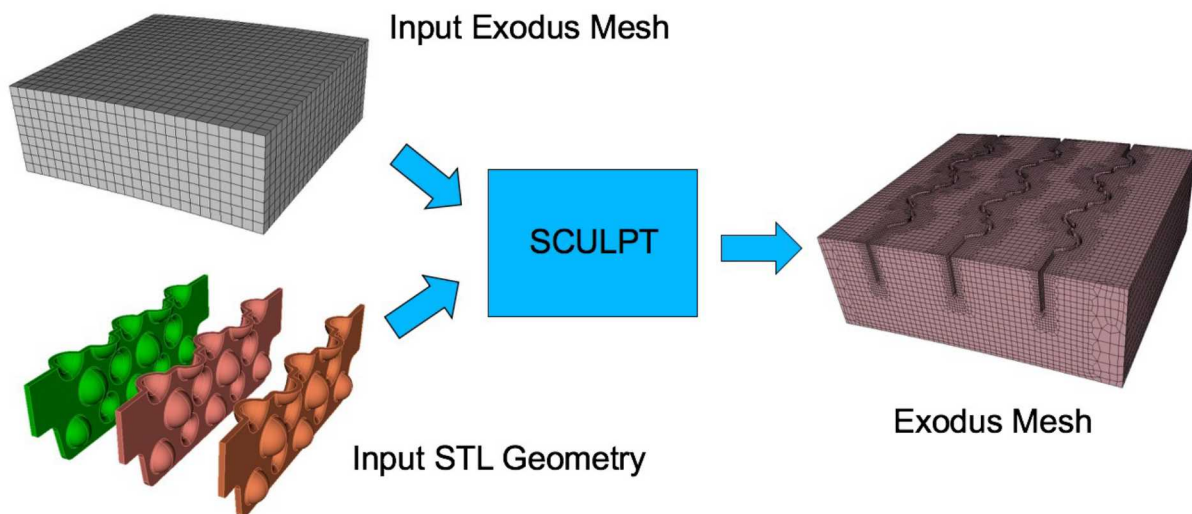
Figure 7-1. An exodus file is used as the base mesh for Sculpt and STL files describe the geometry to be sculpted.

The input_mesh option can also be used in parallel. Sculpt currently requires the mesh to be decomposed prior to running sculpt. The SEACAS decomp tool can be used to pre-process any exodus mesh to break it into multiple meshes ready for use in sculpt. SEACAS is an open source library available on github. For example, when using four processors with sculpt, you would use the following command:

```
decomp -p 4 simple-mesh.g
```

The result would be the four meshes:

```
simple-mesh.g.4.0
simple-mesh.g.4.1
simple-mesh.g.4.2
simple-mesh.g.4.3
```

Once the base mesh has been decomposed, Sculpt can be run. In this case, the `input_mesh` option would use the root `simple-mesh.g` as the argument.

```
input_mesh = simple-mesh.g
```

If the `-j 4` option is used, sculpt will look for 4 meshes in the current working directory with the appropriate root and extension.

Four different options are supported for describing the geometry when using the `input_mesh` option:

- `stl_file`: A single file containing a water-tight faceted description of the geometry. Note that only the portion of the STL file completely contained within the base mesh will be represented in the final mesh.

- `diatom_file`: May contain analytic descriptions of geometric primitives and/or references to multiple STL files.

- `input_spn`: The materials of the cells in the spn file are mapped onto the elements of the input mesh using inverse distance-weighted interpolation. As with the stl and diatom files, only the portion of the spn file completely contained within the base mesh will be represented in the final mesh. The `input_mesh_blocks` option can be used in conjunction with the spn_file option to limit the scope of the mapping of material from the spn file to the mesh file. If this options is used, only elements in the specified blocks will get mapped to. For more details, see the `input_mesh_blocks` option.

- `Element Variables`: The geometry may also be described by element variables in the Exodus file. Element variables should represent material volume fractions where the sum of element variables for any one cell should be between 0.0 and 1.0. Any number of element variables may be used where each unique variable defined will describe an element block in the final Exodus mesh produced. if the sum of element variables is less than 1.0 for any one element, a void material will be assumed and removed from the base mesh unless the `mesh_void` option is used.

```
Limitations:
```

- An STL file and element variables cannot be used in the same input. If element variables are present in the Exodus file and an STL or Diatom file is used, the element variables will be ignored.

- If an input mesh is used, any Cartesian grid specifications will be ignored (ie. `nelx`, `xmin`, `xmax`).

- The `adapt_type` option will work only for an exodus input mesh that defines a mapped mesh. Adapt types `vfrac_average (4)` and `vfrac_difference (6)` are currently the only criteria supported with the `input_mesh` option.

7.14. BLOCKS OF INPUT BASE EXODUS MESH

```
Command: input_mesh_blocks      Block ids of Input Base Exodus mesh
```

```
Input file command:    input_mesh_blocks <arg>
Command line options: -imb <arg>
Argument Type:         integers > 0
```

Command Description:

This option is valid when specifying both `input_mesh` and `spn_file`. Using this option, the materials of the cells in the spn file are mapped onto only the elements of the specified blocks in the input_mesh file. The remaining blocks are treated as void. The behavior without this option maps the materials of the cells in the spn file onto elements of all blocks in the input_mesh file.

7.15. MATERIAL DEFINITION WITH INPUT MESH

```
Command: input_mesh_material     Material definition with input mesh
```

```
Input file command:    input_mesh_material <arg>
Command line options: -imm <arg>
Argument Type:         integers > 0
Input arguments: geometry (0)
                 blocks (1)
```

Command Description:

This option is valid when specifying an 'input_mesh' . Using this option, the material definition in the final mesh may be defined based on the material definitions on the geometry, or based on the block ids of the input mesh. For example, a diatom file defining geometry would have materials defined which are used to define the materials in the final mesh. The default is to use material definitions on the geometry. Possible options are:

- `geometry (0)`: Material defined by geometry

- `blocks (1)`: Material defined by blocks in input mesh

Behavior of interior faces differs when using the blocks option. For instance, interior faces are defined by block interfaces rather than STL or diatom geometry. Exterior faces, on the other hand, are still defined by STL or diatom geometry. When combined with the capture option, only exterior faces are captured.

7.16. INPUT BASE MESH DEFINED BY PAMGEN

Command: `input_mesh_pamgen`   Input Base mesh defined by Pamgen

```
Input file command:    input_mesh_pamgen <arg>
Command line options: -imp <arg>
Argument Type:         file name with path
```

Command Description:

Option to use Pamgen to create a base mesh for Sculpt. Pamgen is an open source meshing tool developed at Sandia for generating hexahedral meshes from geometric primitives. In addition to being a stand-alone meshing solution, it is a parallel tool that is integrated as an inline meshing tool for Sandia's shock physics simulation tool, Alegra. Pamgen has also been integrated in Sculpt as a solution for automatically defining a base mesh.

The `input_mesh_pamgen` option permits a mesh defined my Pamgen input parameters to define the base mesh. A limited set of brick and cylinder primitives are supported by Pamgen. The name of an ascii file containing the pamgen mesh definition is used as the argument for this option. The following is a simple example of a pamgen mesh description. It generates a partial cylinder with a span of 90 degrees and height of 1.0. Other parameters allow for specific interval and sizing specifications as well as block/material identification.

```
mesh
    radial trisection
        trisection blocks, 2
        zmin -0.00075
        numz 1
            zblock 1 1. interval 8
        numr 3
            rblock 1 2.0 interval 8
            rblock 2 3.0 interval 8
            rblock 3 4.0 interval 8
        numa 1
            ablock 1 90. interval 24
    end
    set assign
        nodeset, ilo, 100
        block sideset, ihi, 45, 2
    end
end
```

For a full description of Pamgen and input parameters see the following document:

David M. Hensinger, Richard R. Drake, James G. Foucar, Thomas A. Gardiner, "Pamgen, a Library for Parallel Generation of Simple Finite Element Meshes", Sandia Report SAND2008-1933 (2008)
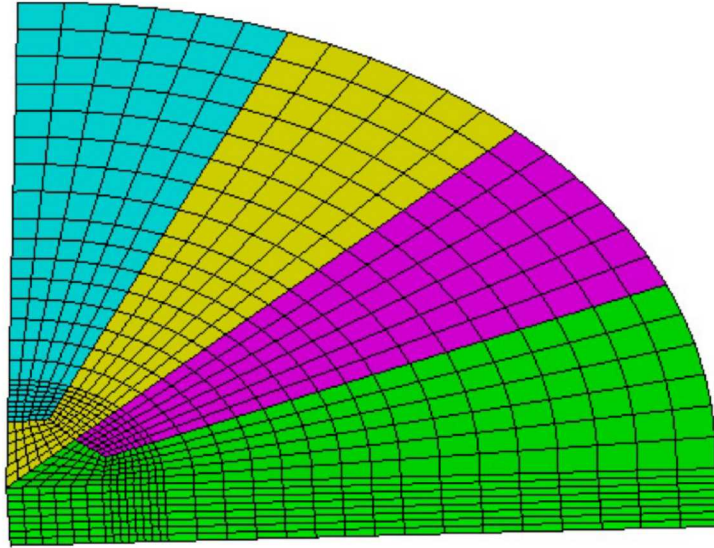
Figure 7-2. Base mesh generated by pamgen using the above input parameters. Colors represent 4 different processors when used in parallel mode.

Similar to the `input_mesh` option, the same geometry input options are available. They include `stl_file`, `diatom_file` and `input_spn`. See the `input_mesh` option for additional details and limitations.

8. MESH TYPE

Sculpt options for specifying the type of mesh that will be generated. The default mesh type that will be produced from Sculpt is an unstructured all-hex mesh that will attempt to conform as closely as possible to the input geometry. Sculpt will normally generate its mesh on the interior of the input geometry, however with the `mesh_void` option, it can also generate the mesh on the exterior of the geometry, out to the extent of the user-defined Cartesian overlay grid.

In addition to the default hex mesh, other types of meshes may be produced. This includes the stair-step mesh where the cells of the Cartesian grid inside or intersecting the geometry are used directly as the mesh without projections or smoothing. A triangle mesh may also be generated, which can be used as the basis for a facet-based geometry representation. Other methods include the capabilities to generate a hex-dominant mesh with hexes and tets as well as the ability to include degenerate elements.

```
Mesh Type  --type  -typ
  --stair               -str <arg> Generate Stair-step mesh
  --mesh_void           -V   <arg> Mesh void
  --htet                -ht  <arg> Convert hexes below quality threshold to tets
  --trimesh             -tri      Generate tri mesh of geometry surfaces
  --tetmesh             -tet <arg> Under Development
  --deg_threshold       -dg  <arg> Convert hexes below threshold to degenerates
  --max_deg_iters       -dgi <arg> Maximum number of degenerate iterations
  --htet_material       -htm <arg> Convert hexes in given materials to tets
  --htet_transition     -htt <arg> Transition method between hexes and tets
  --htet_pyramid        -htp <arg> Local transition pyramid
  --htet_tied_contact   -htc <arg> Local transition tied contact
  --htet_no_interface   -htn <arg> Local transition none
```

8.1. STAIR

Command: stair   Generate Stair-step mesh

Input file command: stair <arg>
Command line options: -str <arg>

```
Argument Type:          integer (0, 1, 2, 3)
Input arguments: none (0)
                 off (0)
                 on (1)
                 full (1)
                 interior (2)
                 fast (3)
```

Command Description:
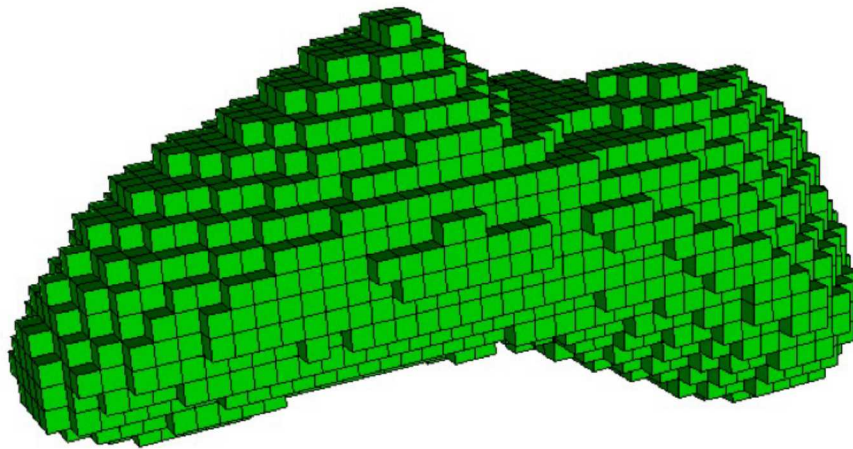
Figure 8-1. Example stair-step mesh on STL geometry.

The stair option generates a stair-step mesh where the cells of the Cartesian grid are used in the final mesh without projection or smoothing to the material interfaces. Cells selected from the Cartesian grid to be used in the final mesh will have volume fraction greater than 0.5. Several different options for the stair argument are available:

`off (0)`: Stair option is off (default)

`full (1)`: Stair-step mesh is generated, but additional processing is done to ensure material interfaces are manifold. This option may add or subtract cells from the basic mesh (where volume fraction > 0.5) to ensure no non-manifold connections between nodes and edges exist in the final mesh.

`interior (2)`: The exterior boundary will be smooth while internal material interfaces will be stair-step. This option also ensures manifold connections between elements.

`fast (3)`: Generates the final mesh based only on volume fraction criteria. No additional processing is done to ensure manifold connections between edges and nodes.

8.2.   MESH VOID

```
Command: mesh_void      Mesh void

Input file command:   mesh_void <arg>
Command line options: -V <arg>
Argument Type:        true/false or only
Input arguments: off (0)
                 false (0)
                 on (1)
                 true (1)
                 only (2)
```

Command Description:
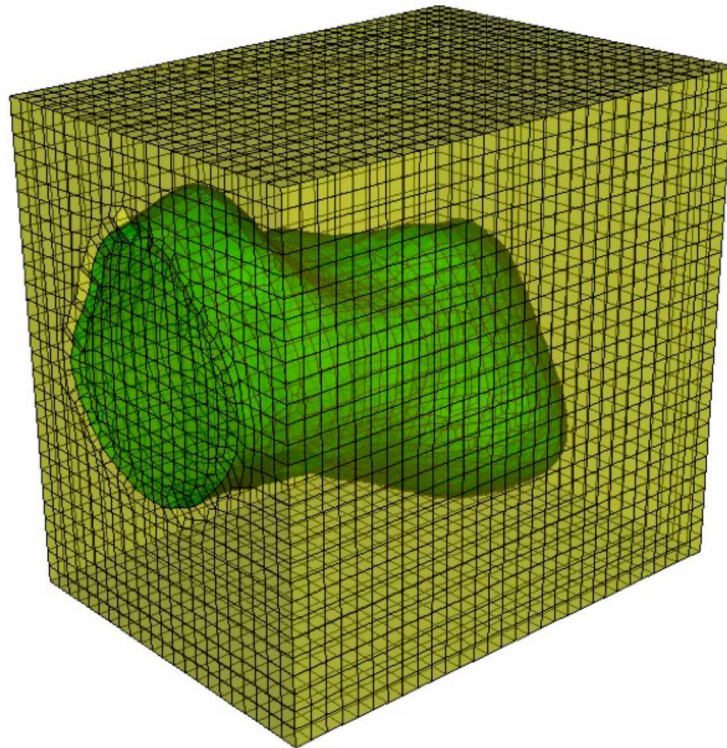
Figure 8-2. Mesh is generated in the void region surrounding the STL geometry.

The mesh_void accepts the following parameters:

off (0): No mesh is generated in the void region

on (1): Mesh is generated in the void region

only (2): Mesh is generated only in the void region and not in the material

If mesh_void option is set to on or only, then the void space surrounding the geometry will be treated as a separate material. Elements will be generated in the void to the extent of the Cartesian grid boundaries. If void_mat option is not used, the material ID of elements in the void region will be the maximum material ID in the model + 1.

8.3.   HTET

```
Command: htet      Convert hexes below quality threshold to tets

Input file command:   htet <arg>
Command line options: -ht <arg>
Argument Type:        floating point value (-1.0 -> 1.0)
```

Command Description:
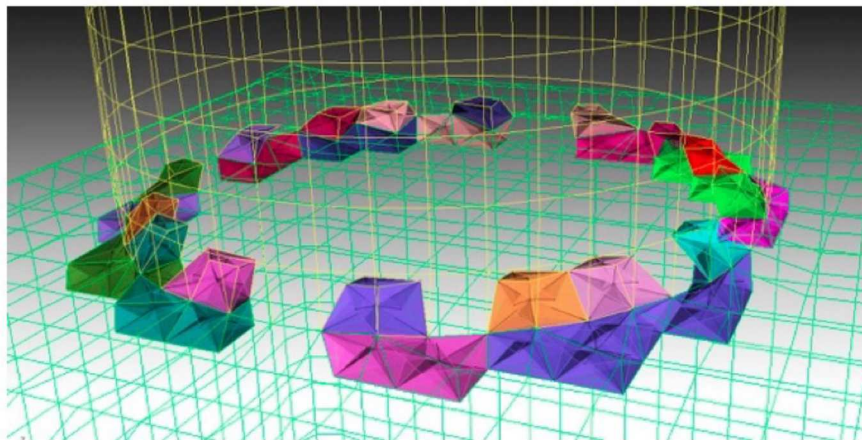
Figure 8-3. Tet elements generated where quality drops below threshold.

Automatically generate tets in place of poor quality elements. This option can be used to eliminate poor quality hex elements by replacing each hex that falls below the user defined Scaled Jacobian with 24 tets. The 24 tets are formed by inserting one node at the center of each face and one on the interior. Default value for htet is -1.0.

If an neighboring element is a hex, and will not be split, one may choose whether to use pyramid transitions or have hanging nodes. The default is to have hanging nodes with a tied contact condition being created. The transition type may be specified with the htet_transition command.

If tet blocks are created, their ids will be the material id plus an offset based on the maximum material id. Likewise, any pyramid blocks created will be offset as well, with their ids coming after hex block ids if there are no tets, or with their ids coming after tet blocks.

8.4.   TRIMESH

```
Command: trimesh      Generate tri mesh of geometry surfaces

Input file command:    trimesh
Command line options: -tri
```

Command Description:
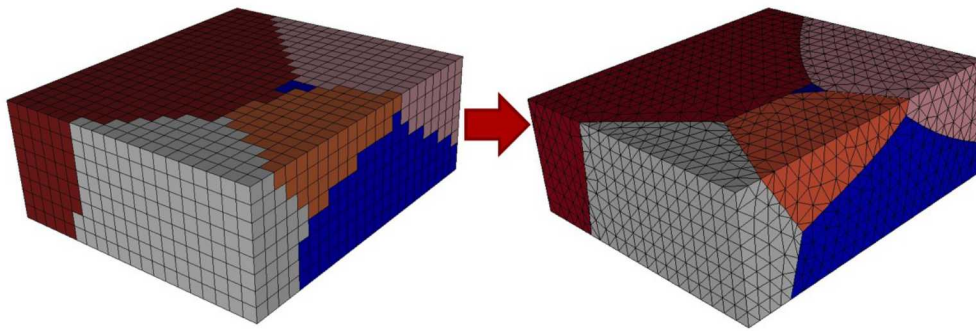
Figure 8-4. Trimesh generated from voxel microstructure data.

Generate a triangle mesh of the surface geometry. Surface geometry will be defined based on input grid resolution as well as user defined smoothing smoothing parameters. Resulting exodus mesh will contain only TRI elements. All TRI elements will be assigned to the same block in the exodus file.

This option is most often used in conjunction with the –write_geom option used to build a mesh-based geometry in Cubit. Use the following command in Cubit to import a Sculpt trimesh exodus file and s2g file (produced from –write_geom)

```
  import s2g <root filename>
```

See write_geom for more information on s2g files.

8.5.   TETMESH

```
Command: tetmesh      Under Development

Input file command:    tetmesh <arg>
Command line options: -tet <arg>
Argument Type:         none
Input arguments: off (0)
                 on (1)
```

```
        true (1)
        meshgems (2)
```

Command Description:

Under Development - uses space-filling tets as base grid. Size and extent is defined by bounding box options.

The `meshgems  (2)` option uses a third party tet mesher to place interior tets. Triangle mesh is defined by splitting quads on surface. Both tetmesh options are currently only implemented for serial execution.

8.6.   DEGENERATE (EDGE COLLAPSE) THRESHOLD

```
Command: deg_threshold     Convert hexes below threshold to degenerates

Input file command:    deg_threshold <arg>
Command line options: -dg <arg>
Argument Type:         floating point value (-1.0 -> 1.0)
```

Command Description:

Rock
1-edge collapse

Knife
1-face collapse

Axe
2-edge collapses

Wedge
2-edge collapses

Pyramid
3-edge collapses

Tetrahedron
4-edge collapses
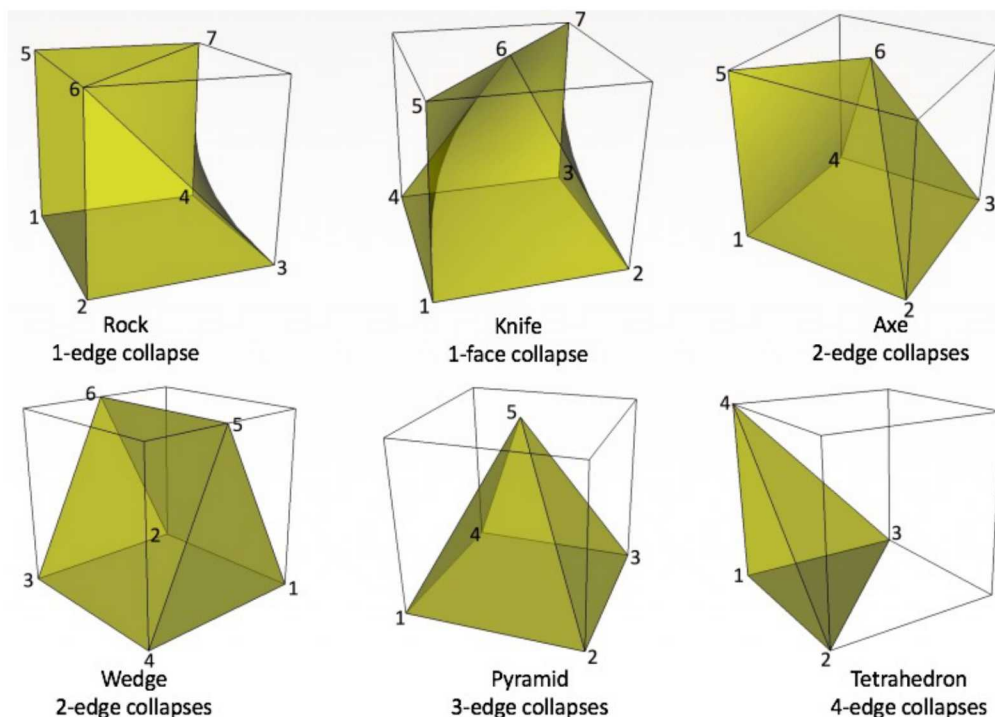
Figure 8-5. Examples of degenerates hexes where select edges have been collpased.

Some geometries will not permit a usable mesh with a traditional all-hex mesh. Sculpt includes the option to automatically and selectively collapse element edges to improve low-quality elements. The max_deg_iters and the deg_threshold values are used to control the creation of degenerates. Degenerate elements are treated as standard hex elements, but use repeated nodes in the eight-node connectivity array.

The deg_threshold value indicates scaled Jacobian threshold for edge collapses. Nodes at hexes below this threshold will be candidates for edge collapses, provided doing so will improve the minimum scaled Jacobian at the neighboring hexes. Default is -1.0.

8.7.   MAXMIMUM DEGENERATE ITERATIONS

```
Command: max_deg_iters      Maximum number of degenerate iterations

Input file command:   max_deg_iters <arg>
Command line options: -dgi <arg>
Argument Type:        integer >= 0
```

Command Description:

Maximum number of edge collapse iterations to perform to create degenerate hex elements. Default is 0. See also `deg_threshold`

8.8.   HTET MATERIAL

```
Command: htet_material      Convert hexes in given materials to tets

Input file command:   htet_material <arg>
Command line options: -htm <arg>
Argument Type:        integer >= 0
```

Command Description:

Generate tets in place hexes in a given material. This option can be given multiple times to specify multiple materials. Each hex in a material is replaced with 24 tets. The 24 tets are formed by inserting one node at the center of each face and one on the interior.

If an neighboring element is a hex, and will not be split, one may choose whether to use pyramid transitions or have hanging nodes. The default is to have hanging nodes with a tied contact condition being created. The transition type may be specified with the `htet_transition` command.

If tet blocks are created, their ids will be the material id plus an offset based on the maximum material id. Likewise, any pyramid blocks created will be offset as well, with their ids coming after hex block ids if there are no tets, or with their ids coming after tet blocks.

```
htet_material = 10
htet_material = 12
htet_transition = pyramid
htet_no_interface =  10 13
```
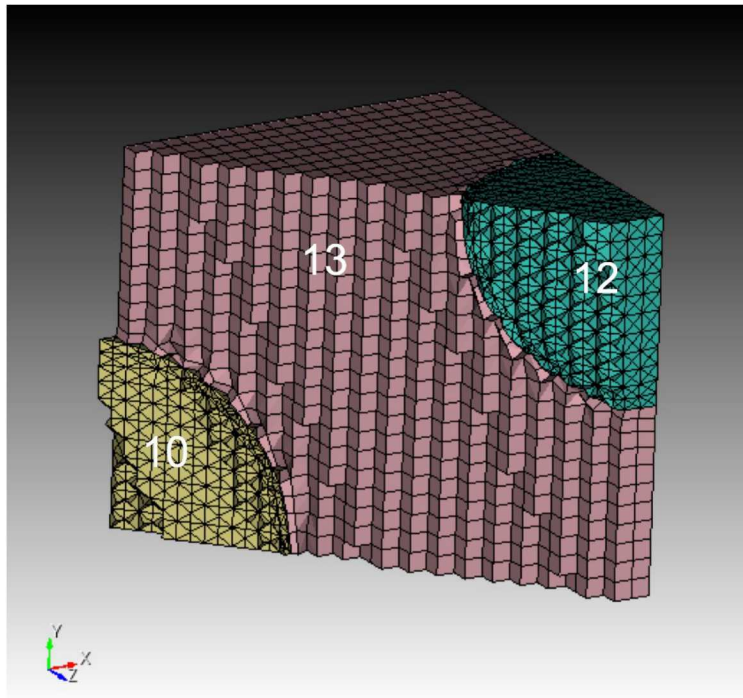

Figure 8-6. Simple example of the use of hybrid tet-hex capability using the above example input. Materials 10 and 12 use tet elements while 13 remains hexes. The default transition is to use pyramids, while the specific interface between 10 and 13 has no interface.

8.9.   HTET TRANSITION

```
Command: htet_transition      Transition method between hexes and tets

Input file command:   htet_transition <arg>
Command line options: -htt <arg>
Argument Type:        none/pyramid/tied_contact
```

```
Input arguments: none (0)
                 pyramid (1)
                 tied_contact (2)
```

Command Description:

When generating tets adjacent to hexes, the transition type between the two elements can be defined. Possible options are:

- `none (0)`: No transition between hex and tet

- `pyramid (1)`: Pyramid transition between hex and tet

- `tied_contact (2)`: Tied contact condition between hex and tet

If pyramid transition is specified, the hex may be split into 1 pyramids and 20 tets, 2 pyramids and 16 tets, 3 pyramids and 12 tets, and so forth. The mesh will remain conformal if pyramid transition is specified.

A tied contact condition can be defined to ensure continuity of the neighboring tets and hexes. To facilitate this, one additional nodeset and sideset will be generated and output to the exodus file if the gen_sidesets = variable (2) option is specified. The sideset and nodeset will be identified with the following IDs:

Sideset 10000 = the set of hex faces that interface a set of 4 tets.

Nodeset 1000 = the set of nodes at the interface between hexes and tets. One node per face in Sideset 10000 will be included.

8.10. LOCAL HTET TRANSITION PYRAMID

```
Command: htet_pyramid    Local transition pyramid

Input file command:    htet_pyramid <arg>
Command line options: -htp <arg>
Argument Type:         integer(s) >= 0
```

Command Description:

When generating tets adjacent to hexes, pyramid transitions can be specified for a given material or material interface. To specify a material interface, two material ids are given to specify pyramid transition between the two materials. To specify multiple materials or multiple material interfaces, this command may be used multiple times.

8.11. LOCAL HTET TRANSITION TIED CONTACT

```
Command: htet_tied_contact      Local transition tied contact


Input file command:    htet_tied_contact <arg>
Command line options: -htc <arg>
Argument Type:         integer(s) >= 0
```

Command Description:

When generating tets adjacent to hexes, tied contact transitions can be specified for a given material or material interface. To specify a material interface, two material ids are given to specify tied contact transition between the two materials. To specify multiple materials or multiple material interfaces, this command may be used multiple times.

8.12. LOCAL HTET TRANSITION NONE

```
Command: htet_no_interface      Local transition none


Input file command:    htet_no_interface <arg>
Command line options: -htn <arg>
Argument Type:         integer(s) >= 0
```

Command Description:

When generating tets adjacent to hexes, no transition can be specified for a given material or material interface. To specify a material interface, two material ids are given to specify no transition between the two materials. To specify multiple materials or multiple material interfaces, this command may be used multiple times.

9. BOUNDARY CONDITIONS

Sculpt options for specifying the methods for generating nodesets, sidesets and blocks on the mesh. Several automatic methods for generating nodesets and sidesets are provided in Sculpt using the gen_sidesets option. Where multiple blocks are required, Block IDs are normally defined using the material ID in the diatom file. Each STL file can be associated with a different block ID. If the mesh_void option is used, the ID for the block of elements in the void region can be set using the void_mat option.

For other input formats such as volume fraction microstructure data or Cartesian Exodus files, the Block IDs are defined by the individual formats.

```
Boundary Conditions   --boundary_condition  -bc
  --void_mat              -VM  <arg> Void material ID (when mesh_void=true)
  --gen_sidesets          -SS  <arg> Generate sidesets
  --free_surface_sideset -FS  <arg> Free Surface Sideset
  --match_sidesets         -mss <arg> Sidesets ids of matching pairs
```

9.1.   VOID MATERIAL ID

```
Command: void_mat      Void material ID (when mesh_void=true)

Input file command:    void_mat <arg>
Command line options: -VM <arg>
Argument Type:         integer > 0
```

Command Description:

When the mesh_void option is used, this value is the material (block) ID assigned to all elements in the void region. If void_mat option is not used, the material ID of elements in the void region will be the maximum material ID in the model + 1. Note that the void_mat may be the same as an existing material in another part of the model.

9.2. GENERATE SIDESETS

```
Command: gen_sidesets      Generate sidesets

Input file command:   gen_sidesets <arg>
Command line options: -SS <arg>
Argument Type:        integer (0, 1, 2, 3, 4, 5)
Input arguments: off (0)
                 fixed (1)
                 variable (2)
                 geometric_surfaces (3)
                 geometric_sidesets (4)
                 rve (5)
                 input_mesh_and_stl (6)
                 input_mesh_and_free_surfaces (7)
                 rve_variable (8)
```

Command Description:
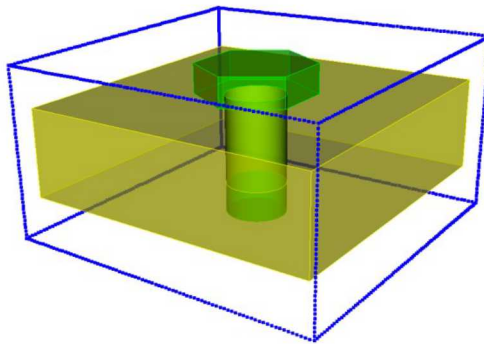
Figure 9-1. Geometry used in sideset examples below.

Generate exodus sidesets using one of the following options:

off (0): No sidesets will be generated

fixed (1): Exactly 3 sidesets will be generated according to the following:

- Sideset 1: All sides at the domain boundary. Sides will only be present in this sideset if the model intersects the enclosing bounding box or the void option is used.

- Sideset 2: All sides at the model boundary. Any side on the model that is not interior will be included. This should represent a full enclosure of the model if it does not intersect the domain boundary.

- Sideset 3: All sides at material interfaces. Includes sides on the interior where adjacent blocks are different.
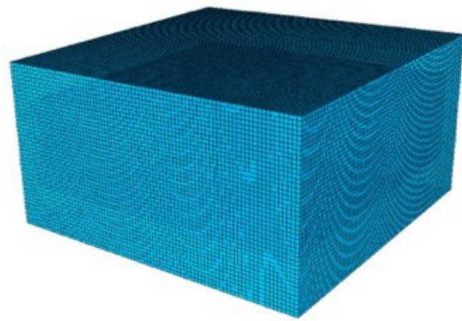
Sideset 1: all sides at the six sides of the original Cartesian grid. These are present only if the Void option is selected, or the geometry intersects the boundary
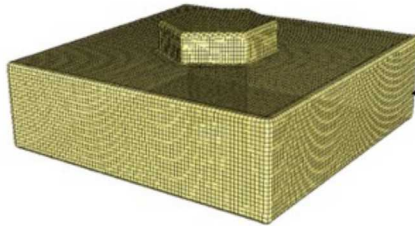
Sideset 2: all sides that are at a free surface of any block of elements
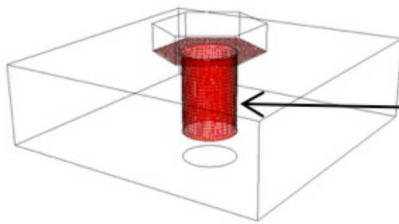
Sideset 3: all sides that define the interface between different blocks

Figure 9-2. Example of fixed(1) sidesets.

`variable (2)`: A variable number of sidesets will be generated with the following characteristics:

- Surfaces at the domain boundary

- Exterior material surfaces

- Interfaces between materials

Unlike Fixed sidesets, grouping of sides will be contiguous. A separate sideset will be generated for each set of contiguous sides.
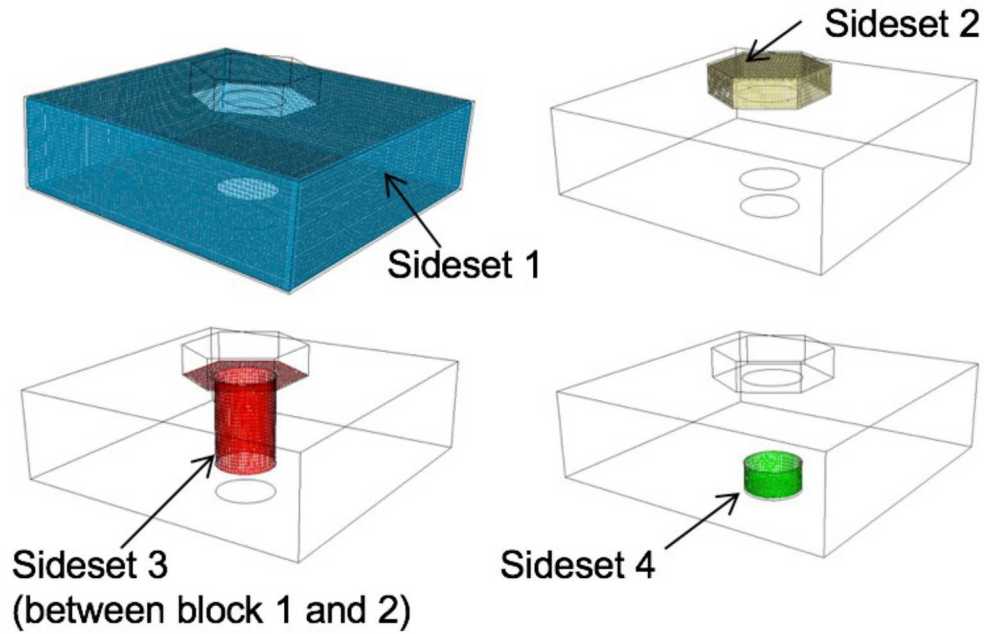
Figure 9-3. Example of variable(2) sidesets.

`geometric_surfaces (3):` Sidesets will be generated according to imported surface ID information. STL files may include an optional surface designation for any or all triangles in the file. Surface information may be written automatically from Cubit based on geometric surface IDs or sideset IDs. See the cubit sculpt parallel sideset option for more details. If present, one sideset will be generted for each surface designation in the STL file. Following is an example surface designation in an STL file. It would appear following all triangles.

```
surface 1
  0 1 2 3 4 5 6 7 8 9
  10 11 12 13 14 15 16 17 18 19
  20 21 22 23
endsurface 1
```

The id following the surface designation will be used as the sideset ID. Up to 10 triangle IDs, per line may be assigned to the surface. Triangle IDs are assigned based on order they appear in the STL file. Any number of surfaces may be defined. For this option, the assumption is that all triangles included in the STL files will be included in at least one surface designation.

In this example there a 17 sidesets generated.
One for each surface in the CAD model

CAD model with
surface IDs
labeled
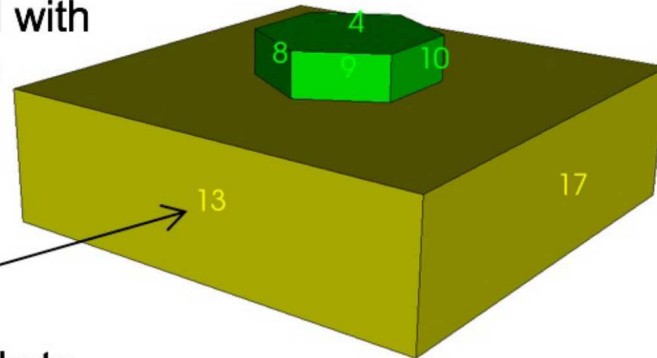
Sideset 13
Corresponds to
surface 13

Sideset 13
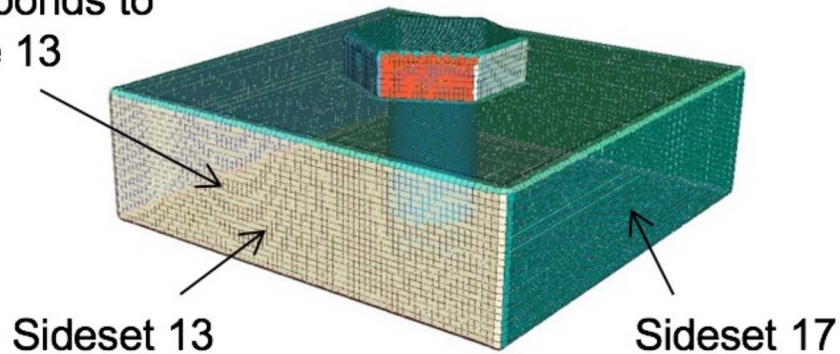
Sideset 17

Figure 9-4. Example of all geometric surfaces (3) defining sidesets.

geometric_sidesets (4): Similar to geometric_surfaces, except that only a portion of the triangles may be designated as sideset surfaces. This option is useful when using Cubit to identify specific surfaces as sidesets.
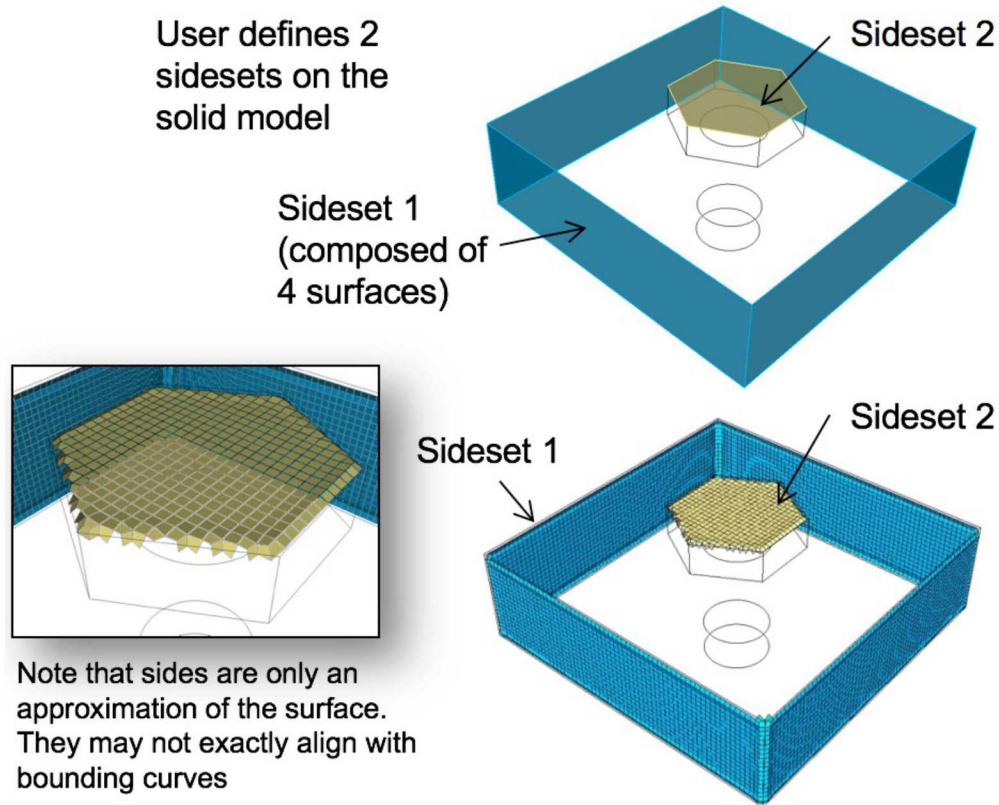
Figure 9-5. Example of selected geometric sidesets (4) in Cubit defining sidesets in Sculpt.

RVE (5): When using the full bounding box, such as representative volume elements (RVE) for microstructures, the nodesets and sidesets with IDs 1 to 6 are reserved for the six faces of the bounding box. They are assigned as follows:

```
Nodeset/Sideset ID        Contains nodes/faces
        1                 on minimum X domain boundary
        2                 on maximum X domain boundary
        3                 on minimum Y domain boundary
        4                 on maximum Y domain boundary
        5                 on minimum Z domain boundary
        6                 on maximum Z domain boundary
```

In addition, a nodeset and sideset will be generated on interior surfaces for each unique pair of adjacent material IDs. One final nodeset will also be generated along interior curves at all internal triple junctions (curves where at least 3 surfaces share a common curve).
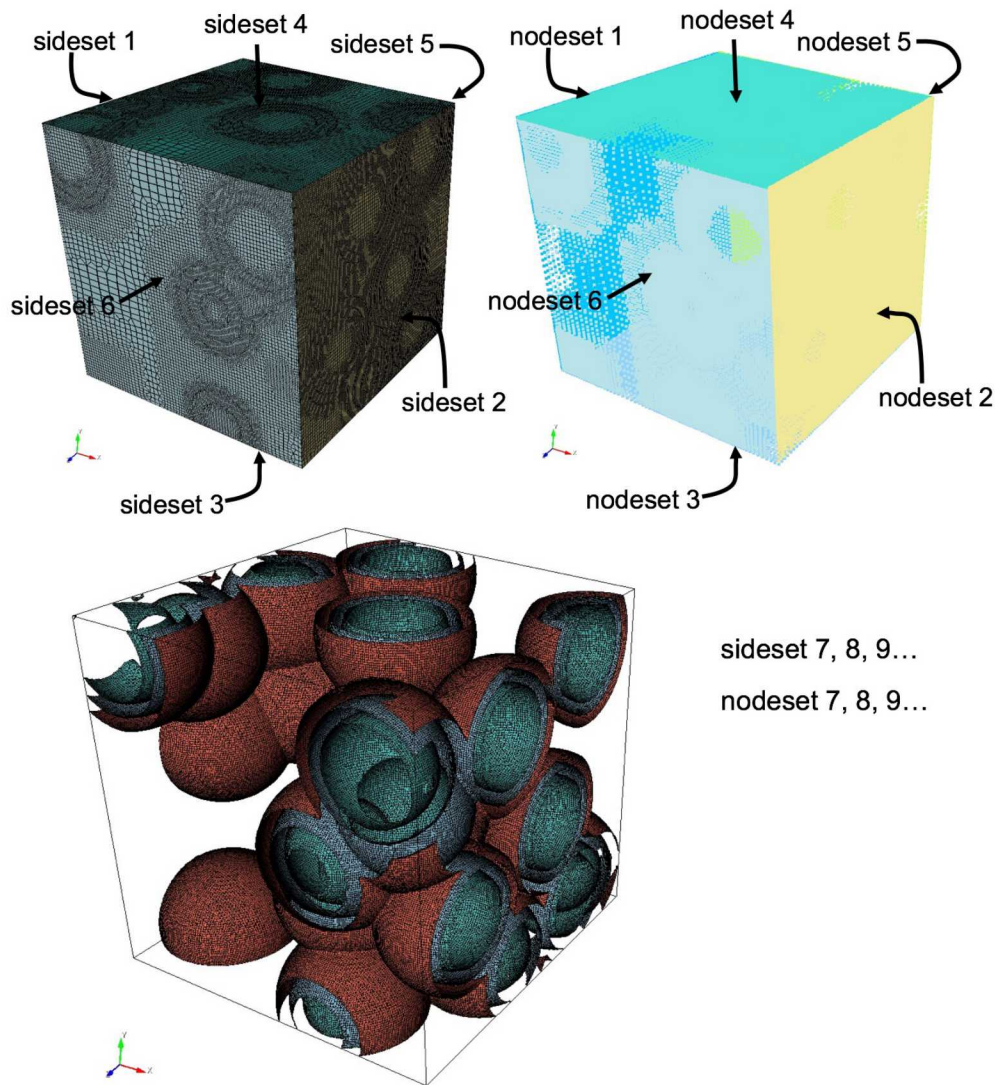
Figure 9-6. Example of automatically defined sidesets at domain boundaries of an RVE and at all interface surfaces between materials.

`input_mesh_and_stl` (6): Used with the `input_mesh` option where an exodus file is used as the base grid. Sidesets and nodesets defined in the input exodus mesh are transferred to the output mesh if the surface is not an interior surface. Sidesets defined in the augmented STL input file are transferred to the output mesh for interior surfaces. See also the `free_surface_sideset` option for prescribing a sideset on interior surfaces cut by the STL definition when using the `input_mesh` option.
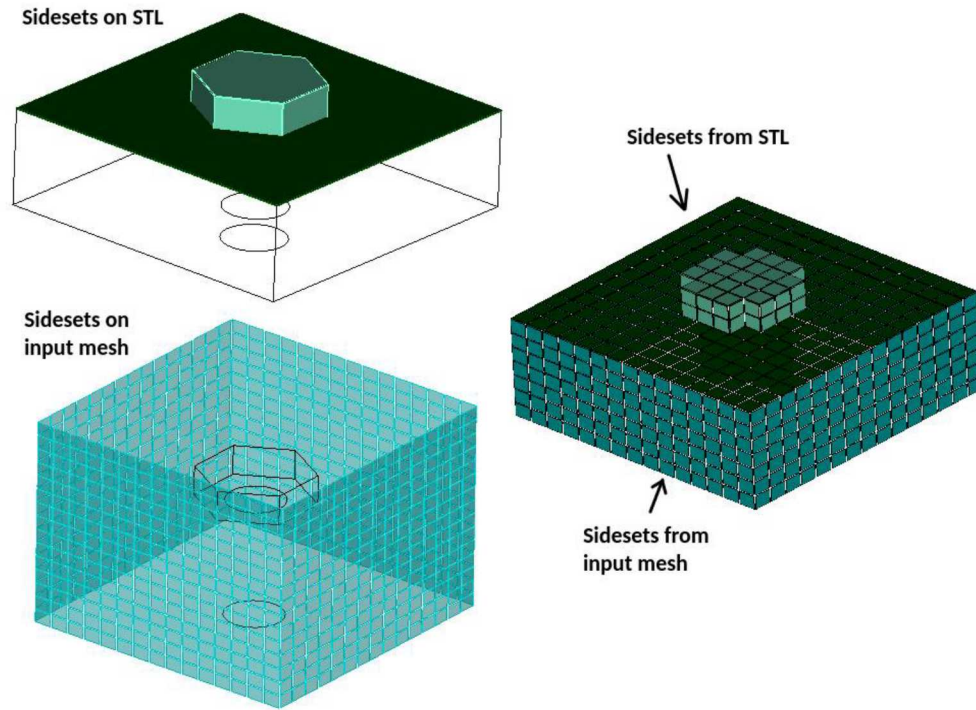
Figure 9-7. Example of sidesets defined in the input mesh and corresponding domain boundary sidesets in the output mesh.

`input_mesh_and_free_surfaces` (7): Used with the `input_mesh` option where an exodus file is used as the base grid. Sidesets and nodesets defined in the input exodus mesh are transferred to the output mesh if the surface is not an interior surface. Sidesets defined in the `free_surface_sideset` option are used to define sidesets for interior surfaces.
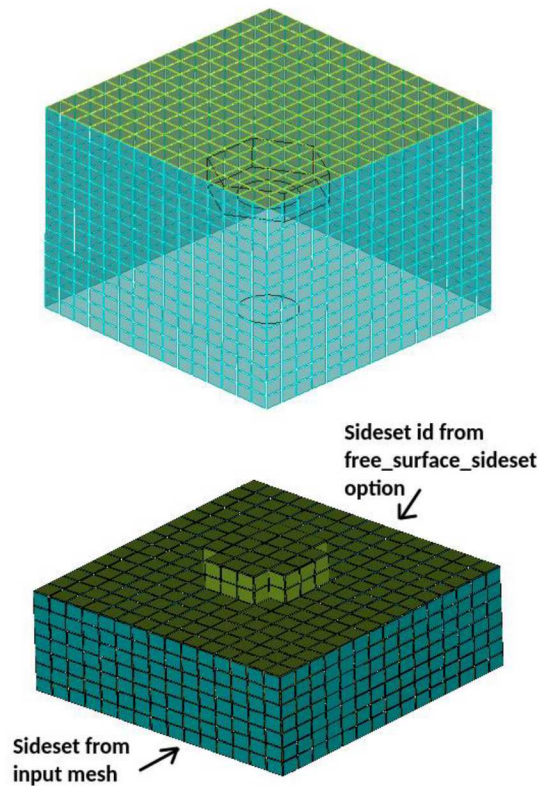
Figure 9-8. Example of sidesets defined in the input mesh and corresponding domain boundary sidesets in the output mesh.

rve_variable (8): Nodesets 1-6 and Sidesets 1-6 are defined at the boundaries as described in the gen_sidesets = rve (5) option. With the rve_variable option, additional nodesets and sidesets at material interfaces on the interior of the mesh are defined similar to the gen_sidesets = variable (2) option. Grouping of interior sides in a sidesets will be contiguous, where a separate sideset will be generated for each unique set of contiguous sides. Nodesets will be generated in a similar manner.

9.3. FREE SURFACE SIDESETS

Command: free_surface_sideset   Free Surface Sideset

Input file command: free_surface_sideset <arg>
Command line options: -FS <arg>
Argument Type: integer(s) >= 0

Command Description:

Given exodus sidesets are treated as interior surfaces for STL projection.

Used with the `input_mesh` option when using an exodus mesh as the base grid. This may be useful if the `capture` option is enabled and some of the STL surfaces are close in proximity to the boundaries of the input exodus mesh. When close in proximity, sculpt will by default not project those boundary nodes to the STL surface but keep them on the domain boundary. If a list of sideset IDs are given here, the sideset faces will be projected to the STL. The sideset IDs should refer to sidesets that are defined in the specified input_mesh exodus file.
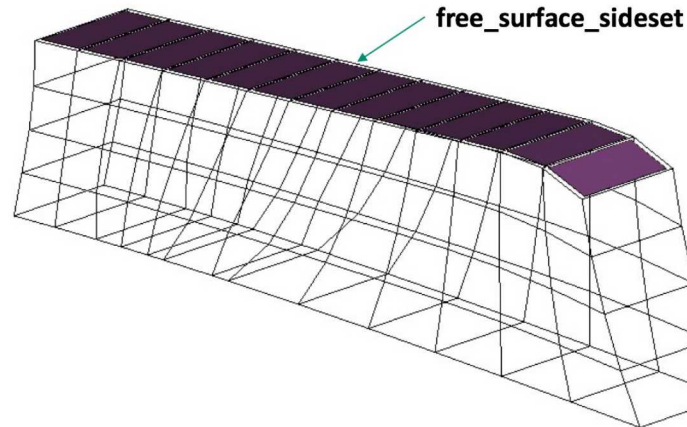
Figure 9-9. Example of free_surface_sideset defined on the top surface faces of an input mesh

9.4.   MATCH SIDESET IDS

Command: match_sidesets   Sidesets ids of matching pairs

Input file command: match_sidesets <arg>
Command line options: -mss <arg>
Argument Type: integer(s) >= 0

Command Description:

If used with an unstructured base grid (input mesh), this option allows the user to define a crack in the input mesh, where the faces of each vertical side (wall) of the crack are each in a different sideset. The faces at the bottom of the crack share a common edge (V-bottom) or face (square-bottom). Sculpt will match or equalize the volume fractions of the bottom cells on either side of the crack. This produces a uniform, higher quality mesh at the crack. The sidesets must be specified in a pairwise order. This option must be used with the `-input_mesh` (-im) option.

10. ADAPTIVE MESHING

Sculpt options for specifying adaptive meshing. Sculpt uses an initial overlay Cartesian grid that serves as the basis for the all-hex mesh. The default mesh size will roughly follow the constant size cells of the overlay grid. The adaptivity option allows the user to automatically split cells of the Cartesian grid based on geometric criteria, resulting in smaller cells in regions with finer details. The adapted grid is then used as the basis for the Sculpt procedure.
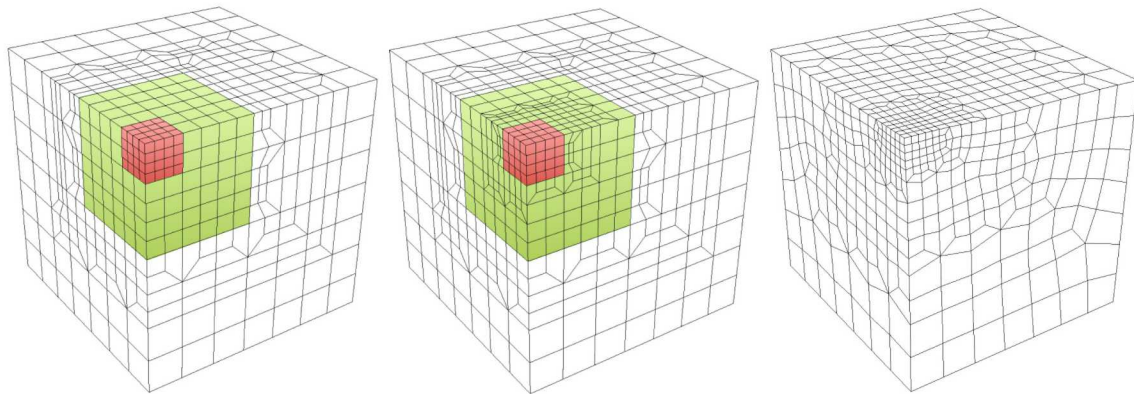
Figure 10-1. Adaptive mesh begins with constant size coarse Cartesian grid. Cells are recursively split based on geometry criteria and transitions added between levels. Projections and smoothing are performed to improve element quality.

Three options are used for controlling the adaptivity in sculpt: adapt_type, adapt_levels and adapt_threshold. The adapt_type option controls the method and geometric criteria used for deciding which cells to split in the grid, while the adapt_levels option controls the the maximum number of times any one cell can be split. Depending upon the adapt_type selected, the adapt_threshold is used as the specific geometric threshold value at which the decision is made to split any given cell.
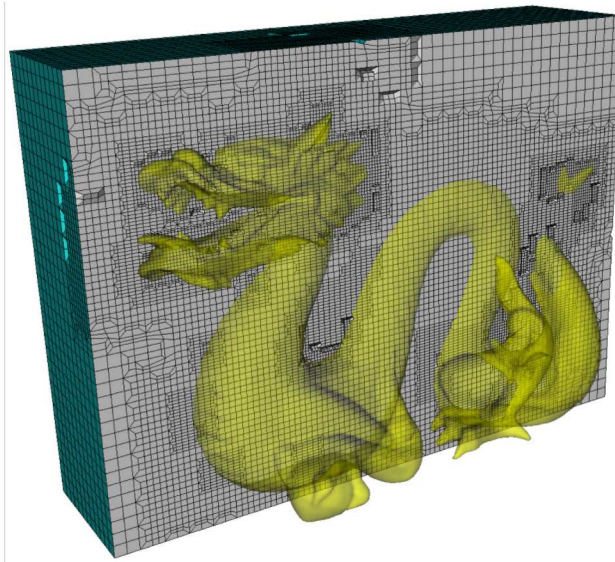
Figure 10-2. Initial cut-away view of adapted grid from dragon model before performing Sculpt operations.
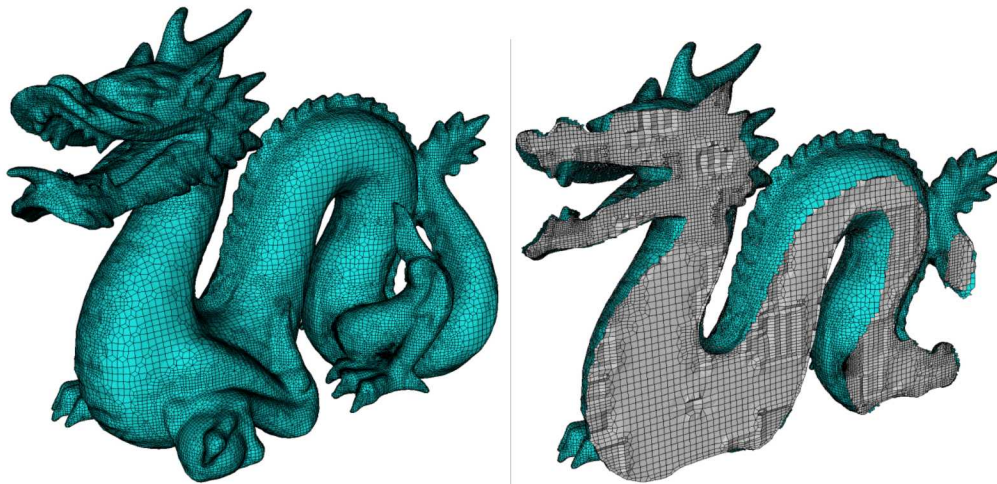
Figure 10-3. The final mesh of the dragon model and cutaway view of the mesh is shown with up to 4 levels of adaptive refinement.

```
Adaptive Meshing  --adapt  -adp
  --adapt_type          -A   <arg> Adaptive meshing type
  --adapt_threshold     -AT  <arg> Threshold for adaptive meshing
  --adapt_levels        -AL  <arg> Number of levels of adaptive refinement
  --adapt_export        -AE        Export exodus mesh of refined grid
```

10.1. ADAPTIVE REFINEMENT TYPE

```
Command: adapt_type      Adaptive meshing type

Input file command:    adapt_type <arg>
Command line options: -A <arg>
Argument Type:         integer (0, 1, 2, 3, 4, 5)
Input arguments: off (0)
                 facet_to_surface (1)
                 surface_to_facet (2)
                 surface_to_surface (3)
                 vfrac_average (4)
                 coarsen (5)
                 vfrac_diff (6)
                 vfrac_difference (6)
```

Command Description:

This option will automatically refine the mesh according to a user-defined criteria. Without this option, a constant cell size will be assumed everywhere in the model. To build the mesh, Sculpt uses an approximation to the exact geometry of the CAD model by interpolating mesh surfaces from volume fraction samples in each cell of the Cartesian grid. In general, the, higher the resolution of the Cartesian grid, the more sampling is done and the more accurate the mesh will represent the initial geometry. The adapt_type selected will control the criteria used for refining the mesh. If the criteria is not satisfied, the refinement will continue until a threshold indicated by the adapt_threshold parameter is satisfied everywhere, or the maximum number of levels (adapt_levels) is reached. The following criteria for refinement are available:

- off (0): Cartesian grid is defined only by nelx, nely, and nelz or cell_size which is used as the basis for the sculpt mesh.

  No refinement will be performed.

- facet_to_surface (1): This option will evaluate every location where an edge in the Cartesian grid intersects a triangle of the STL model and measures the closest distance to the approximated geometry. The cells adjacent to intersecting edges where the measured distance is greater than the adapt_threshold will be identified for uniform refinement. This is done for each refinement level where a new approximated geometry is then computed based upon the finer resolution grid. The refinement will continue until all measured distances are less than the adapt_threshold, or the maximum number of levels (adapt_levels) is reached. This option can only be used if input comes from an STL file. Microstructures and diatoms are currently not supported.
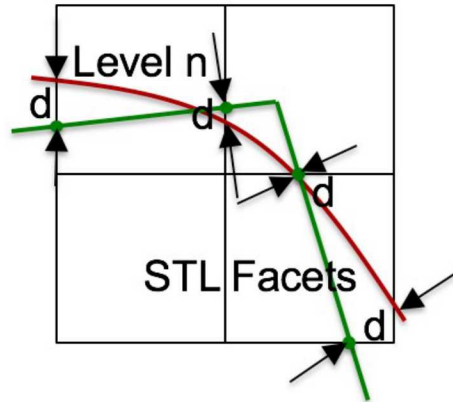
Figure 10-4. Distance from STL Facet to Approximated Geometry. The distance d is measured between the facets (green) where they cross the edges of the grid, to the closest point on the interpolated geometry. If d > adapt_threshold then the cell is split.

`surface_to_facet` (2): This criteria is similar to `facet_to_surface` (1) except that the locations selected for sampling are chosen from the vertices representing the approximated surfaces. The closest distance measured to any of the facets in the STL model is used as the criteria for refinement. Those cells at vertices where the distance measured exceeds the `adapt_threshold` are identified for refinement. This option is generally faster than 1, but may miss features if the initial resolution of the grid is too coarse. This option can also only be used if input geometry comes from an STL file. Microstructures and diatoms are currently not supported.
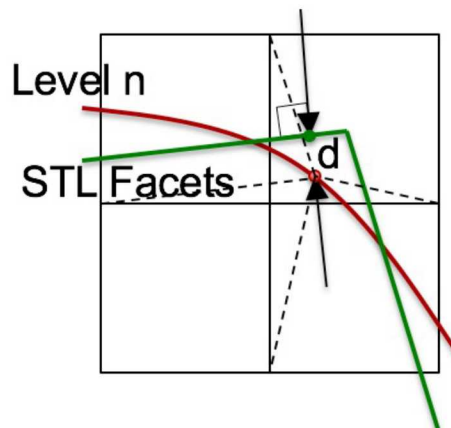
Figure 10-5. Distance from Approximated Geometry to STL Facet. The distance d is measured between points on the interpolated geometry corresponding to a projected point on the grid, to its closest point on one of the STL facets. If d > adapt_threshold then the cell is split.

`surface_to_surface` (3): This criteria will test each cell to compute the local interpolated

surface for the cell and compare with the surface interpolated for its eight subdivided child cells. If the distance between these two approximated surfaces is greater the the user defined `adapt_threshold`, then the cell will be uniformly refined. This option can be used with STL and diatom input geometry, but not with Microstructures.
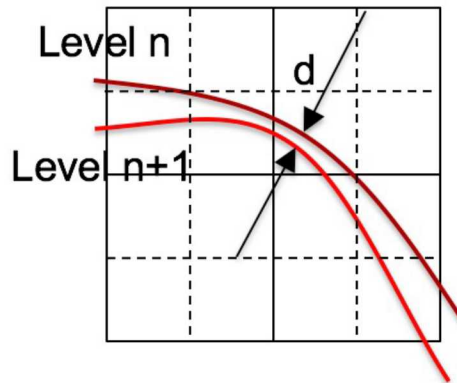
Figure 10-6. Distance Between Child and Parent Approximated Geometry. After computing the interpolated geometry for level n and level n+1, d is the distance between the two geometry representations. Cells where d > adapt_threshold are split.

`vfrac_average` (4): Each cell of the Cartesian grid is tested to determine if it should be subdivided into eight cells. The volume fraction of the parent cell is compared with the average volume fraction of its eight child cells. If the absolute difference between the average child volume fraction and its parent volume fraction is greater than the user defined `adapt_threshold` then the cell is uniformly refined. The `adapt_threshold` for this case should be a number between 0 and 1. A smaller number will be more sensitive to changes in geometry, usually resulting in more refinement at interfaces.
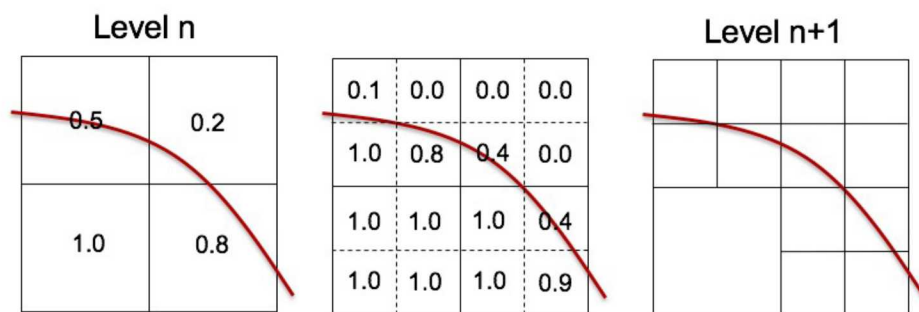
Figure 10-7. Difference of Cell Volume Fraction. Volume fractions are evaluated for the 8 child cells of a cell at level n. This example shows where one or more of the volume fractions at level n+1 of the lower left cells does not differ by more than a threshold d, so it remains unsplit.

`coarsen` (5): Given a dense set of data on a Cartesian Grid, Sculpt will begin at a coarse resolution and refine to capture changes in the data. It uses the adapt_levels option to determine

the coarseness of the initial grid. For example, a dense grid of LxMxN cells will begin with an initial resolution of L/2â x M/2â x N/2â, where a is the user defined adapt_levels value. Cells will be identified for refinement if the volume fraction of any material in a cell is greater than the user defined `adapt_threshold` and less than 1.0-adapt_threshold. This option is available only for `input_spn` and `input_micro` formats. It is most useful for cases where very dense data is initially provided which would be too fine to serve as an FEA mesh. This method will effectively coarsen the mesh on the interior and exterior of solids, but maintain a fine resolution at geometry boundaries.
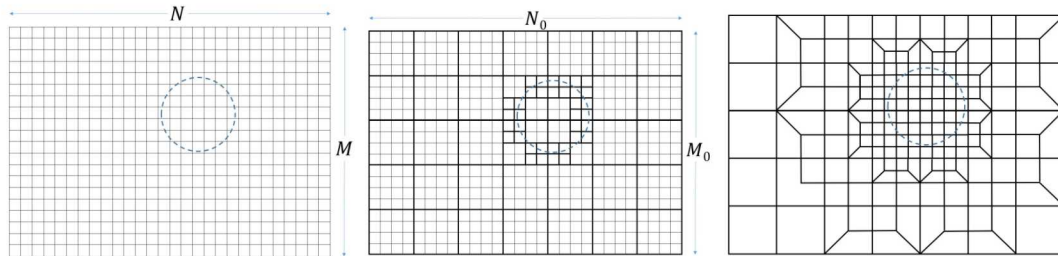
Figure 10-8. Refine to Dense Data (Coarsening). Initial grid at resolution N X M is coarsened to N0 X M0 based on the adapt_levels value. Coarse cells are then split similar to criteria in adapt_type = 4.

`vfrac_difference (6)`: Each cell of the Cartesian grid is tested to determine if it should be subdivided into eight cells. The volume fraction of the parent cell is compared with each of the volume fractions of its eight child cells. If the absolute difference between any of the child volume fractions and its parent volume fraction is greater than the user defined `adapt_threshold` then the cell is uniformly refined. The `adapt_threshold` for this case should be a number between 0 and 1. A smaller number will be more sensitive to changes in geometry, usually resulting in more refinement at interfaces.

To maintain a conforming mesh, transition elements will be inserted to transition between smaller and larger element sizes. Default for the `adapt_type` option is `off (0)` (or that no adaptive refinement will take place).

In all cases the initial Cartesian grid defined by xint, yint and zint or the `cell_size` value will be used as the basis for refinement and will define the approximate largest element size in the mesh.

10.2. ADAPTIVE REFINEMENT THRESHOLD

```
Command: adapt_threshold      Threshold for adaptive meshing


Input file command:    adapt_threshold <arg>
Command line options: -AT <arg>
Argument Type:         floating point value >= 0.0
```

Command Description:

This value controls the sensitivity of of the adaptivity. The value used should be based upon the adapt_type:

- `facet_to_surface (1)`

- `surface_to_facet (2)`

- `surface_to_surface (3)`

  For these options, the `adapt_type` selected represents an absolute distance between surfaces or facets. Where the distance exceeds `adapt_threshold` the nearby cell or cells will be identified for refinement. The smaller this number the more sensitive will be the adaptation and greater the resulting number of elements. If not specified, the `adapt_threshold` will be determined as follows:

  ```
  adapt_threshold = 0.25 * cell_size / adapt_levels\^2
  ```

- `vfrac_average (4)`

- `coarsen (5)`

  The `adapt_threshold` value in this case represents the maximum difference in volume fraction between a parent cell and the average of its eight child cells. This value should be between 0.0 and 1.0. The smaller the number, the more sensitive will be the adaptation and the greater the number of resulting elements. A default `adapt_threshold` of 0.01 is used if not specified.

Note that the user defined adapt_threshold may not be satisfied everywhere in the mesh if the value defined for adapt_levels is exceeded.

10.3. NUMBER OF ADAPTIVE LEVELS

```
Command: adapt_levels     Number of levels of adaptive refinement

Input file command:   adapt_levels <arg>
Command line options: -AL <arg>
Argument Type:        integer >= 0
```

Command Description:

The maximum number of levels of adaptive refinement to be performed. One level of refinement will split each Cartesian grid cell identified for uniform refinement into eight child cells. Two levels of refinement will split each cell again into eight, resulting in sixty-four child cells, three levels into 256, and so on. The maximum number of subdivision per cell is give as:

```
number of cells = 8\^adapt_levels
```

The minimum edge length for any cell will be given by:

```
min cell edge length = cell_size / adapt_levels\^2
```

The actual number of refinement levels used will be determined by whether all cells meet the `adapt_threshold`, or the adapt_levels value is exceeded. The default adapt_levels is 2. Note that setting the `adapt_levels` more than 4 or 5 can result in long compute times.

10.4. EXPORT REFINED CARTESIAN GRID

```
Command: adapt_export      Export exodus mesh of refined grid

Input file command:   adapt_export
Command line options: -AE
```

Command Description:

Export an exodus mesh containing the refined Cartesian grid. Interface reconstruction, boundary layer insertion and smoothing have not yet been applied to this mesh. It is the base mesh used as input to Sculpt. One file per processor will be exported in the form "vfrac_adapt.e.x.x". The exodus mesh produced will also contain the computed volume fractions for each material present in the model represented as element variables.

This option is primarily used for debugging the refinement option. However the mesh produced with this option can be used as the base mesh when used with the `input_mesh` option. For example, instead of Cartesian grid options, the input mesh may be specified as `input_mesh = vfrac_adapt.e.1.0`. Sculpt will use the refined mesh and the volume fraction element variables to build the final mesh.

11. SMOOTHING

Sculpt options for specifying how the mesh will be smoothed following mesh generation.

Sculpt 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. Default numbers of iterations and thresholds for each smoothing phase have been tuned for general use, however it may be worthwhile to adjust these parameters. 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. A fixed default of 2 iterations is applied to all hexes. Increasing the num_laplace parameter can improve some cases, especially convex shapes.

- `Optimization Smoothing`: Applied only to elements who's scaled Jacobian falls below the opt_threshold parameter (default 0.6) and their surrounding elements. This approach uses a more expensive optimization technique to improve regions of elements simultaneously. The max_opt_iters parameter can control the maximum number of iterations applied (default is 5). Iterations will terminate, however, if no further improvement is detected. Because this method optimizes node locations simultaneously, neighboring nodes with competing optimum can sometimes limit mesh quality.

- `Spot Optimization`: Also known as parallel coloring, is applied only to elements who's element quality falls below the pcol_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 using the max_pcol_iters to achieve desired results. Increasing the pcol_threshold and max_pcol_iters may yield improved results.

```
Smoothing  --smoothing  -smo
  --smooth              -S   <arg> Smoothing method
  --csmooth             -CS  <arg> Curve smoothing method
  --laplacian_iters     -LI  <arg> Number of Laplacian smoothing iterations
  --max_opt_iters       -OI  <arg> Max. number of parallel Jacobi opt. iters.
  --opt_threshold       -OT  <arg> Stopping criteria for Jacobi opt. smoothing
  --curve_opt_thresh    -COT <arg> Min metric at which curves won't be honored
  --max_pcol_iters      -CI  <arg> Max. number of parallel coloring smooth iters.
  --pcol_threshold      -CT  <arg> Stopping criteria for parallel color smooth
```

```
--max_gq_iters          -GQI <arg> Max. number of guaranteed quality smooth iters.
--gq_threshold          -GQT <arg> Guaranteed quality minimum SJ threshold
```

11.1. SMOOTH

```
Command: smooth        Smoothing method

Input file command:    smooth <arg>
Command line options: -S <arg>
Argument Type:         integer (0, 1, 2, 3)
Input arguments: off (0)
                 default (1)
                 on (1)
                 fixed_bbox (2)
                 no_surface_projections (3)
                 to_geometry (4)
```

Command Description:

Automatic adjustment of node locations following meshing to improve element quality. Controls the combined Laplacian and optimization smoothing procedures applied to volume and surface nodes (see csmooth for curve smoothing options) Uses the `laplacian_iters`, `max_opt_iters`, `opt_threshold`, `max_pcol_iters`, `pcol_threshold`, `mqx_gq_iters` and `gq_threshold` arguments to control the sensitivity and aggressiveness of the smoothing operations. In most cases, the default options for these parameters are sufficient, however increasing iterations or threshold values, while potentially causing longer run times, may result in improved mesh quality.

Smoothing will adjust the location of nodes on surfaces, projecting them to an approximated surface representation defined by interface reconstruction from volume fractions. In addition to turning smoothing on and `off`, the surface projection characteristics can be adjusted using the `bbox_fixed` and `no_surface_projections` options.

- `off (0)`: No volume and surface smoothing is performed.

- `on/default (1)`: (Default) Combined Laplacian/Optimization (Hybrid) smoothing both surface and volumes. Automatic boundary buffer layer improvement is performed at interior surfaces intersecting the domain boundary.

- `fixed_bbox (2)`: Uses standard hybrid smoothing procedure, however nodes at the the domain boundaries will be projected to one of the six planes of the bounding box. This option turns off the automatic boundary buffer improvement.

- `no_surface_projections (3)`: Uses the `fixed_bbox` method, however interior surfaces are not projected. This can result in smoother interior surface representations for microstructures models. This is effective in smoothing noisy surface data, but can potentially reduce overall volume. This method is default for microstructures file formats.

- `to_geometry (4)`: This option is currently under development. When used with the `capture` option, smoothing will also move nodes to the closest geometry entity. It must currently be used with `capture` to ensure that curves and surfaces are first identified and associated with boundary mesh entities. This option will only work with STL or diatom input that contains STL geometry.

`Boundary Buffer Improvement`: Sculpt's smoothing procedures will use an automatic boundary buffer improvement method. It will attempt to improve the quality of hexes where interior surfaces are close to tangent with the bounding box. This can result in nodes that may not lie precisely on the planes of the domain boundary. The `fixed_bbox (2)` and `no_surface_projections (3)` options will turn off the automatic boundary buffer improvement.

11.2. CURVE SMOOTHING

```
Command: csmooth      Curve smoothing method

Input file command:    csmooth <arg>
Command line options: -CS <arg>
Argument Type:         integer (0, 1, 2, ...6)
Input arguments: off (0)
                 circle (1)
                 hermite (2)
                 average_tangent (3)
                 neighbor_surface_normal (4)
                 vfrac (5)
                 linear (6)
```

Command Description:

The csmooth option controls the smoothing method used on curves. In most cases the default should be sufficient, however it may be useful to experiment with different options. The default curve smoothing option is `vfrac (5)`. The following curve smoothing options are available:

- `off (0)`: No curve smoothing will be performed.

- `circle (1)`: Nodes projected to a fitted circle defined current node and its two neighbors.

- `hermite (2)`: Nodes projected based on Hermite interpolation. Note that this method can only be used in serial (-j 1)

- `average_tangent (3)`: Nodes projected based on average tangent of neighbors. Note that this method may not be parallel serial consistent.

- `neighbor_surface_normal (4)`: Nodes projected based on neighboring surface normals and the resulting intersecting planes.

- `vfrac (5)`: (Default) Nodes projected to initial curve interface defined from the original volume fraction data.

- `linear (6)`: Nodes projected to the linear segment defined by the node and its two immediate neighbors.

11.3. LAPLACIAN ITERATIONS

```
Command: laplacian_iters      Number of Laplacian smoothing iterations

Input file command:    laplacian_iters <arg>
Command line options: -LI <arg>
Argument Type:         integer >= 0
```

Command Description:

Number of Laplacian smoothing iterations performed when Hybrid smoothing option is used. Default value is 2.

11.4. MAXIMUM OPTIMIZATION ITERATIONS

```
Command: max_opt_iters      Max. number of parallel Jacobi opt. iters.

Input file command:    max_opt_iters <arg>
Command line options: -OI <arg>
Argument Type:         integer >= 0
```

Command Description:

Indicates the maximum number of iterations of optimization-based smoothing to perform. May complete sooner if no further improvement can be made. Default is 5

11.5. OPTIMIZATION THRESHOLD

```
Command: opt_threshold     Stopping criteria for Jacobi opt. smoothing

Input file command:    opt_threshold <arg>
Command line options: -OT <arg>
Argument Type:         floating point value (-1.0 -> 1.0)
```

Command Description:

Indicates the value for scaled Jacobian where Optimization smoothing will be performed. Elements with scaled Jacobian less than opt_threshold and their neighbors will be smoothed. Default value is 0.6

11.6. CURVE OPTIMIZATION THRESHOLD

```
Command: curve_opt_thresh     Min metric at which curves won't be honored

Input file command:    curve_opt_thresh <arg>
Command line options: -COT <arg>
Argument Type:         floating point value (-1.0 -> 1.0)
```

Command Description:

Indicates the value for scaled Jacobian where if a node that falls on a curve has neighboring quads 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 this value should be set close to zero 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. Default for this value is 0.1.

11.7. MAXIMUM PARALLEL COLORING ITERATIONS

```
Command: max_pcol_iters     Max. number of parallel coloring smooth iters.

Input file command:    max_pcol_iters <arg>
Command line options: -CI <arg>
Argument Type:         integer >= 0
```

Command Description:

Maximum number of spot smoothing (also known as parallel coloring) iterations to perform. May complete sooner if no further improvement can be made. Default is 100. See also pcol_threshold.

11.8. PARALLEL COLORING THRESHOLD

```
Command: pcol_threshold     Stopping criteria for parallel color smooth

Input file command:    pcol_threshold <arg>
Command line options: -CT <arg>
Argument Type:         floating point value (-1.0 -> 1.0)
```

Command Description:

Indicates scaled Jacobian threshold for spot smoothing (also known as parallel coloring). A parallel coloring algorithm is used to uniquely identify and isolate nodes to be improved using optimization. Default is 0.2.

11.9. MAXIMUM GUARANTEED QUALITY ITERATIONS

```
Command: max_gq_iters     Max. number of guaranteed quality smooth iters.

Input file command:    max_gq_iters <arg>
Command line options: -GQI <arg>
Argument Type:         integer >= 0
```

Command Description:

Maximum number of guaranteed quality smoothing iterations to perform. Guaranteed quality smoothing performs a constrained Laplacian smoothing algorithm to adjust node locations. If the result of a smoothing operation results in adjacent element quality falling below the specified `gq_threshold` value, then move distance is cut until minimum threshold is achieved or the metric is improved. To achieve parallel consistency, a parallel coloring methodology is employed. The `max_gq_iters` defines the maximum number of parallel color iterations employed. Default is 0 (off).

Note that guaranteed quality can be utilized in conjunction with other smoothing methods (Laplacian, Optimization and Parallel Coloring), however to be effective it is normally used independent from other smoothing. For example, to use guaranteed quality the following is suggested:

```
laplacian_iters = 0
max_opt_iters = 0
max_pcol_iters = 0
max_gq_iters = 100
```

11.10. GUARANTEED QUALITY THRESHOLD

```
Command: gq_threshold      Guaranteed quality minimum SJ threshold
```

```
Input file command:    gq_threshold <arg>
Command line options: -GQT <arg>
Argument Type:         floating point value (-1.0 -> 1.0)
```

Command Description:

Indicates scaled Jacobian threshold for guaranteed quality smoothing. Default is 0.2. see also `max_gq_iters`

12. MESH IMPROVEMENT

Sculpt options for modifying the mesh to improve mesh quality.

Automatic smoothing provides an effective method for improving element quality. However there may be some cases that cannot be improved with smoothing alone. The options included in this section will apply changes to the underlying hex mesh or to the volume fraction data to increase the opportunity for smoothing to produce a good quality mesh.

```
Mesh Improvement  --improve  -imp
  --pillow               -p   <arg> Set pillow criteria (1=surfaces)
  --pillow_surfaces      -ps        Turn on pillowing for all surfaces
  --pillow_curves        -pc        Turn on pillowing for bad quality at curves
  --pillow_boundaries    -pb        Turn on pillowing at domain boundaries
  --pillow_curve_layers  -pcl <arg> Number of elements to buffer at curves
  --pillow_curve_thresh  -pct <arg> S.J. threshold to pillow hexes at curves
  --pillow_smooth_off    -pso        Turn off smoothing following pillow operations
  --capture              -c   <arg> Project to facet geometry <beta>
  --capture_angle        -ca  <arg> Angle at which to split surfaces <beta>
  --capture_side         -sc  <arg> Project to facet geometry with surface ID
  --defeature            -df  <arg> Apply automatic defeaturing
  --min_vol_cells        -mvs <arg> Minimum number of cells in a volume
  --defeature_bbox       -dbb        Defeature Filtering at Bounding Box
  --defeature_iters      -dfi <arg> Maximum Number of Defeaturing Iterations
  --thicken_material     -thm <arg> Expand a given material into surrounding cells
  --micro_expand         -me  <arg> Expand Microstructure grid by N layers
  --micro_shave          -ms        Remove isolated cells at micro. boundaries
  --remove_bad           -rb  <arg> Remove hexes with Scaled Jacobian < threshold
```

12.1. PILLOW

```
Command: pillow     Set pillow criteria (1=surfaces)

Input file command:   pillow <arg>
```

```
Command line options: -p <arg>
Argument Type:        integer (0, 1, 2, 3)
Input arguments: off (0)
                 surfaces (1)
                 curves (2)
                 domain_boundaries (3)
                 surfaces_no_smoothing (100)
                 curves_2_layers (212)
                 curves_3_layers (213)
                 curves_4_layers (214)
                 curves_5_layers (215)
                 curves_2_layers_no_smoothing (202)
                 curves_3_layers_no_smoothing (203)
                 curves_4_layers_no_smoothing (204)
                 curves_5_layers_no_smoothing (205)
```

Command Description:

For models that have more than one material that share an interface, unless the geometry is precisely aligned with the global axis, it is usually a good idea to turn on pillowing. Pillowing automatically inserts an additional layer of hexes at interface boundaries to improve mesh quality. Without pillowing you may notice inverted or poor quality elements at curve interfaces where 2 or more materials meet.

The pillow option will generate an additional layer of hexes at surfaces as a means to improve element quality near curve interfaces. This is intended to eliminate the problem of 3 or more nodes from a single hex face lying on the same curve. Use one or more of the following options to set up pillowing:

- `pillow_surfaces`: Pillow around all surfaces

- `pillow_curves`: Pillow bad quality at curves

- `pillow_boundaries`: Pillow at domain boundaries

- `pillow_curve_layers`: Number of element layers to buffer curves

- `pillow_smooth_off`: Turn OFF smoothing following pillow operations

See help on the above options for more information

12.2. PILLOW ALL SURFACES

```
Command: pillow_surfaces     Turn on pillowing for all surfaces

Input file command:   pillow_surfaces
Command line options: -ps
```

Command Description: Pillow option to insert a layer of hexes surrounding each internal surface in the mesh. Where two volumes share a common interface is defined as a surface. All hexes that have at least one of its faces on a surface are defined as the "shrink set" of hexes. A separate shrink set is defined for each unique surface. Hexes in the set are shrunk away from their hex neighbors not in the shrink set. A layer of hexes is then inserted surrounding all hexes in each set. This enforces the condition where no more than one hex edge will lie on any single curve thus allowing more freedom for the smoother to improve element quality.
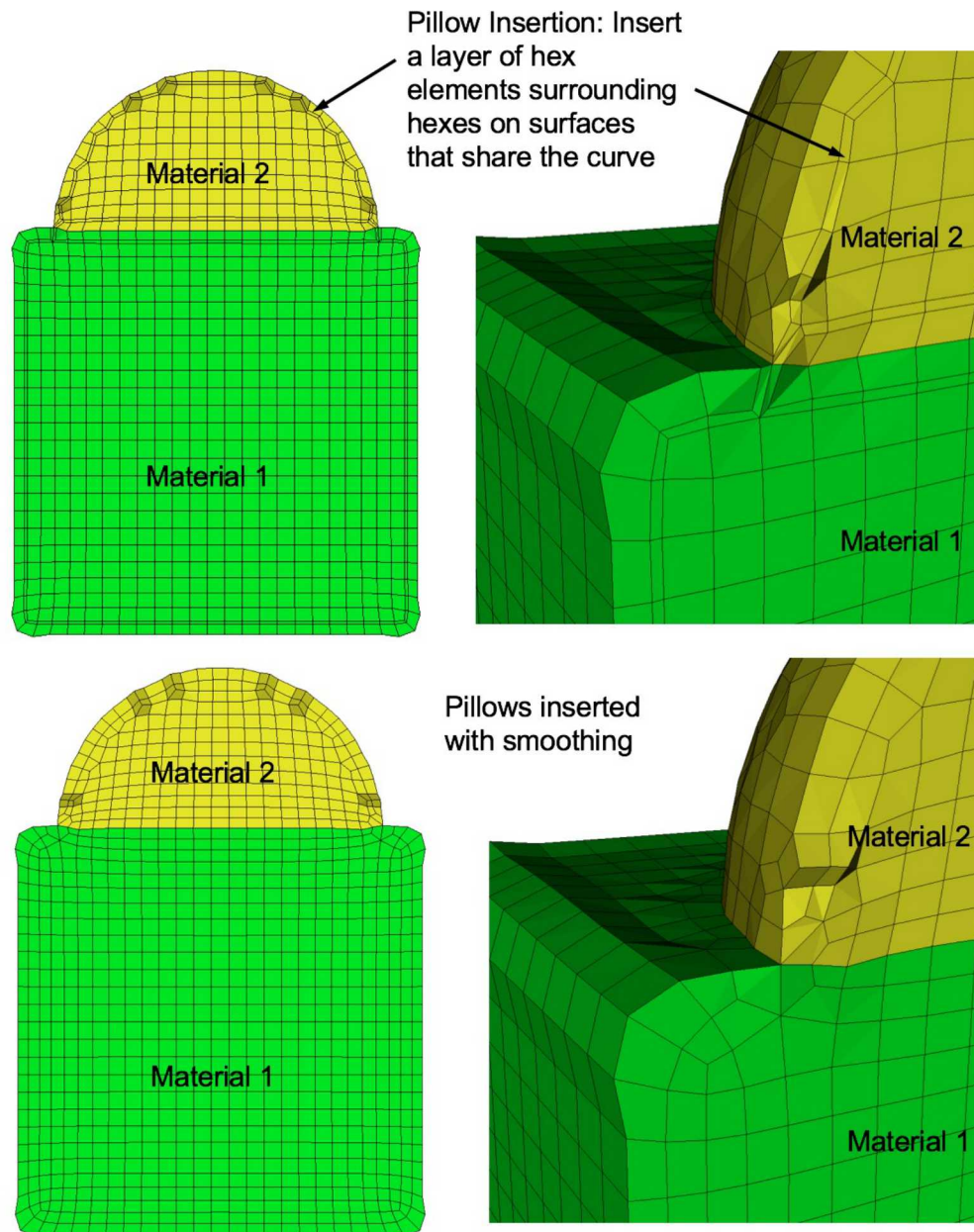

Figure 12-1. Example of surface pillowing, before and after smoothing

Surface pillowing is off by default. If both `pillow_curves` and `pillow_surfaces` options are used,

curve pillowing will be performed before surface pillowing. See the `pillow` option for more information on setting additional options for pillowing.

12.3. PILLOW BAD QUALITY AT CURVES

```
Command: pillow_curves      Turn on pillowing for bad quality at curves

Input file command:   pillow_curves
Command line options: -pc
```

Command Description: Pillow option to selectively pillow hexes at curves. Only hexes that have faces with 3 or more nodes on a curve will be pillowed. Additional buffer layers of hexes beyond the poor quads at the curves will be included in the pillow region. The number of buffer layers beyond the curve can be controlled with the `pillow_curve_layers`, where the default will be 3 layers.

Figure 12-2. Example of curve pillowing with four pillow_curve_layers, before and after smoothing

Curve pillowing is off by default. If both `pillow_curves` and `pillow_surfaces` options are used, curve pillowing will be performed before surface pillowing. See the `pillow` option for more information on setting additional options for pillowing.
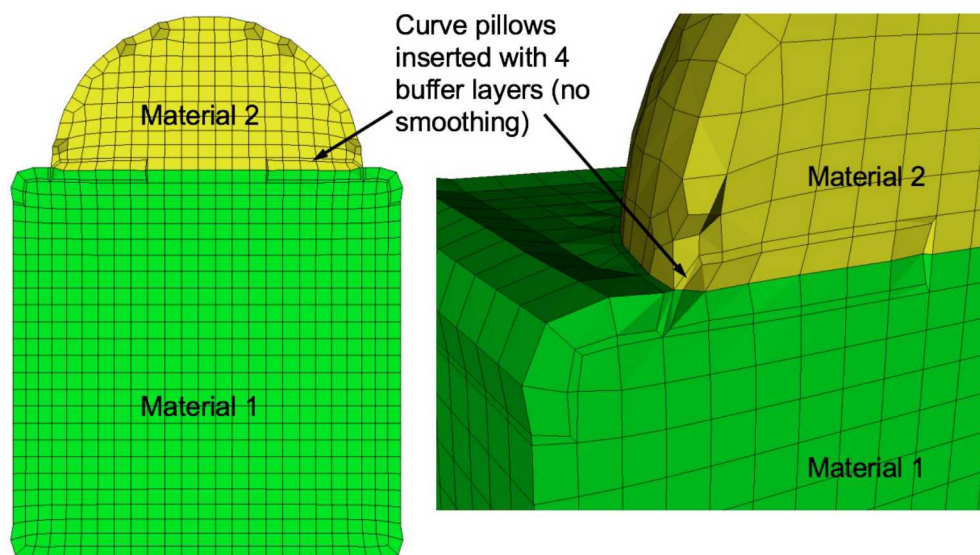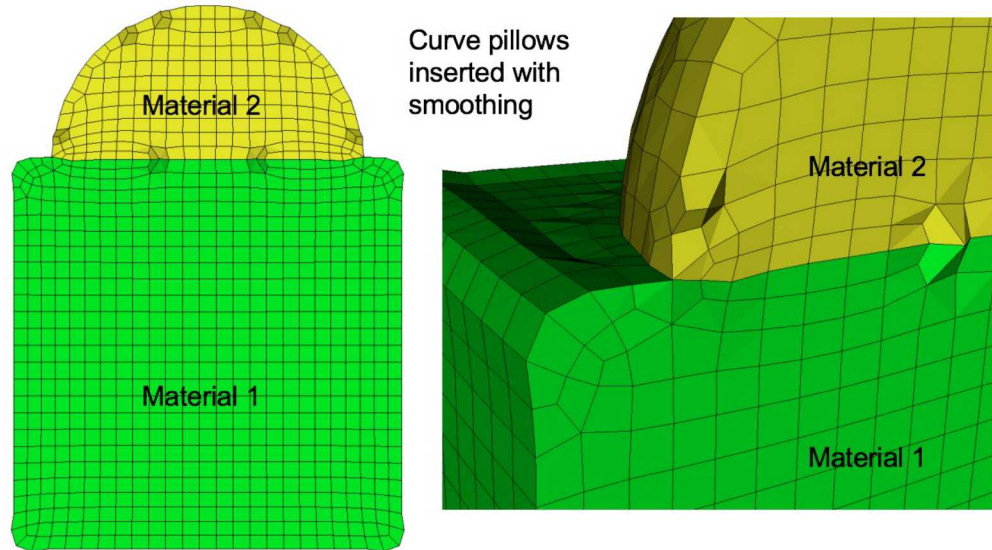

12.4. PILLOW AT DOMAIN BOUNDARIES

```
Command: pillow_boundaries      Turn on pillowing at domain boundaries

Input file command:   pillow_boundaries
Command line options: -pb
```

Command Description: Pillow option to insert pillow layers at domain boundaries of the initial Cartesian grid definition. One layer of hexes is inserted on each of the six faces of the Cartesian Domain. This option is useful where the void option is used to generate a mesh in the full Cartesian grid and where the adapt option has been used. Without this option, it is likely that hexes with two faces on the same domain boundary will occur if the adaptation extends to the boundary. Turning on the `pillow_boundaries` option should correct for these cases.
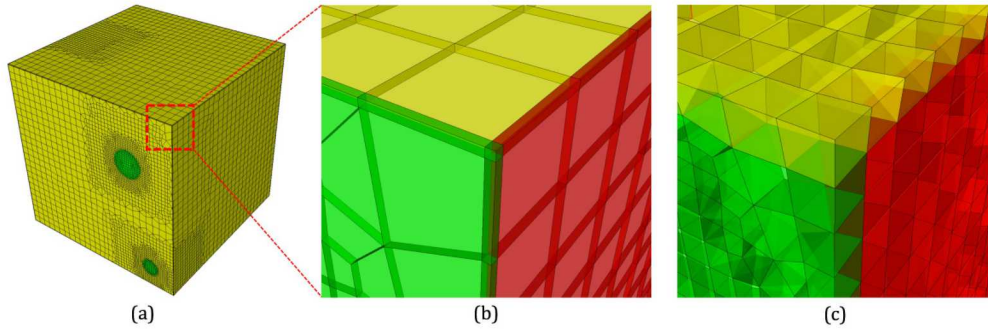
Figure 12-3. Example of pillowing at boundaries on a microstructure RVE. (b) before smoothing (c) after smoothing

Boundary pillowing is off by default. The `pillow_boundaries` option may be used in the same input as `pillow_surfaces` or `pillow_curves`. The `pillow_boundaries` option must also be used with the `mesh_void` option to ensure hexes will exist at the Cartesian domain boundary. See the `pillow` option for more information on setting additional options for pillowing.

12.5. NUMBER OF ELEMENT LAYERS TO BUFFER CURVES

```
Command: pillow_curve_layers      Number of elements to buffer at curves

Input file command:    pillow_curve_layers <arg>
Command line options: -pcl <arg>
Argument Type:         integer > 0
```

Command Description: Used for setting the number of buffer hex layers when the `pillow_curves` option is used. When `pillow_curves` is used a shrink set is formed from hexes that would otherwise have two or more edges on the same curve. This value will control the extent to which neighboring hexes will be included in the shrink set. The default `pillow_curve_layers` is 3. Setting this value lower will localize the modifications to the hex mesh, whereas, more layers will extend the region that is affected in correcting the poor quality at curves.

12.6. SCALED JACOBIAN THRESHOLD FOR CURVE PILLOWING

```
Command: pillow_curve_thresh      S.J. threshold to pillow hexes at curves
```

```
Input file command:     pillow_curve_thresh <arg>
Command line options:  -pct <arg>
Argument Type:          floating point value (-1.0->1.0)
```

Command Description: Used for setting the quality threshold for pillowing hexes at curves. When determining hexes to include in the shrink set, the `pillow_curves` option will look for hexes with more than two nodes of a hex on the same curve. If this condition is satisfied, it will test the mesh quality of quads on the adjacent surfaces that share the common curve. If at least 3 nodes are on a common curve and the Scaled Jacobian of any of the attached quads falls below the, `pillow_curve_thresh` scaled Jacobian metric, then the associated hexes will be included in the shrink set.

Default for `pillow_curve_thresh` is 0.3. Increasing this value will tend to increase the total number of hexes added to the mesh, but may result in better mesh quality after smoothing. Lowering this value may reduce the number of additional hexes but could potentially result in more hexes with poor or bad Scaled Jacobian metrics.

12.7. TURN OFF SMOOTHING FOLLOWING PILLOW OPERATIONS

```
Command: pillow_smooth_off     Turn off smoothing following pillow operations
```

```
Input file command:     pillow_smooth_off
Command line options:  -pso
```

Command Description: Controls the smoothing following pillow operations. To maximize element quality at pillowed hexes, smoothing is always performed after inserting the hex layers. The smoothing step may be omitted if `pillow_smooth_off` is set. This option can be useful for visualizing the pillow layers that have been inserted, but in most cases will generate poor quality or inverted elements.

12.8. CAPTURE

```
Command: capture      Project to facet geometry <beta>
```

```
Input file command:     capture <arg>
Command line options:  -c <arg>
Argument Type:          integer (0, 1, 2)
Input arguments: off (0)
                 on (1)
                 external_surfaces (2)
```

```
projections_only (3)
feature_angle_smooth (4)
topology_smooth (5)
```

Command Description:

This is an experimental option still in development. Nodes at the surfaces of a default sculpt mesh will not necessarily exactly lie on the geometric surfaces prescribed by the input STL geometry. While this characteristic can provide additional flexibility for defeaturing and element quality, there are cases where a more exact surface representation may be desired. The capture option attempts to address this by extracting sharp features and/or projecting nodes to the facet geometry.
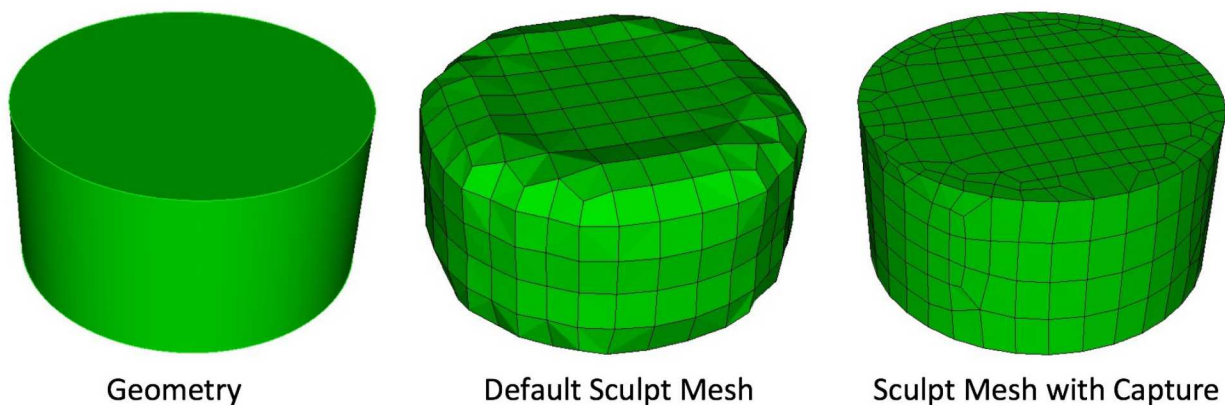

Geometry   Default Sculpt Mesh   Sculpt Mesh with Capture

Figure 12-4. Simple example illustrating the effect of the `capture` `= 5` option. Options `smooth = to_geometry` and `pillow_curves = true` are also used for this example.

Several options are currently being studied as possible solutions. They include the following:

0 = (off) Capture option is off. No attempt is made at capturing sharp features.

1 = (on) STL geometry is used as basis for feature capture. A user defined feature angle is used (capture_angle) to first generate groups of facets from the STL geometry based on capture_angle. Topological curves are defined based on projections to closest surface facets and edges. With default smoothing option, the surface nodes will be projected to the closest STL surfaces as a final step before exporting the exodus mesh. Consider using `smooth = to_geometry` option.

2 = (exterior_surfaces) Only exterior surfaces are captured. Uses the same procedure as described in capture = 1, except that interior surfaces (those with two adjacent volumes), will be ignored in the capture and projections stage.

3 = (projections_only) For this option, additional topology based on feature angle is not extracted. Only the final projection of surface nodes to the STL facets is done. Note that this option is useful for organic shapes that do not have sharp features, or where sharp features should be ignored.

4 = (feature_angle_smooth) This option uses the procedure outlined in `capture = 1`, except that the `smooth = to_geometry` is used by default. Note that `capture = 1` used with `smooth = to_geometry` should be identical to this option.

5 = (topology_smooth) Curve topology is defined similar to `capture = 1`, except that element face topology is first determined based on closest assigned facet. Curve topology is then extracted based on adjacent element face associativity. Surface node projections are only done for nodes that have unambiguous neighbor associativity. This provides for a tolerant approach to resolving topology that may result in defeaturing. (i.e. where the STL facet topology may be locally more complex than can be resolved by the prescribed resolution). This option also uses the `smooth = to_geometry` option as default for smoothing. Also note that `capture = 5` it is only currently available for serial execution (j=1)

12.9. CAPTURE ANGLE

```
Command: capture_angle      Angle at which to split surfaces <beta>
```

```
Input file command:    capture_angle <arg>
Command line options: -ca <arg>
Argument Type:         floating point value (0 -> 360)
```

Command Description:

This is an experimental option still in development. Feature angle for capture option.

12.10. CAPTURE SIDE

```
Command: capture_side      Project to facet geometry with surface ID
```

```
Input file command:    capture_side <arg>
Command line options: -sc <arg>
Argument Type:         integer > 0
```

Command Description:

Similar to the capture option, the capture_side option will project nodes to the initial triangle facets, however projections will be limited only to surface nodes closest to the surface ID specified by the argument. Note that the input STL file can identify and group facets according to a surface ID. However surface IDs are utilized only when using the gen_sidesets option with arguments 3 and 4. When using Cubit, the STL file written when using the sculpt parallel command with sideset options 3 and 4 will include surface identification for surfaces in the STL file. A workflow for using the capture_side option might include the following:

1. Generate or import CAD model into Cubit.

2. Identify and group selected surfaces into a single sideset with unique ID.

3. In the Sculpt GUI set the sideset generation option to 4.

4. Turn on the option: Do not Run Sculpt.

5. In your working directory edit the input file (.i).

6. Add the option capture_side = <id> to the input file.

7. Run sculpt in batch using the input file to control execution.

The result should be a mesh where surface nodes closest to the surfaces identified by the unique sideset ID will lie precisely on their closest surface.

12.11. DEFEATURE

```
Command: defeature       Apply automatic defeaturing

Input file command:    defeature <arg>
Command line options: -df <arg>
Argument Type:         integer (0, 1, 2, 3)
Input arguments: off (0)
                 filter (1)
                 collapse (2)
                 filter_and_collapse (3)
```

Command Description:

Option to automatically detect and remove small features. Primarily used for defeaturing microstructure data, however can be used with any input format. The following options are available:

- off (0): No defeaturing performed (default)

- filter (1): Filters the Cartesian grid data so that groupings of cells of a common material with less than min_vol_cells will be reassigned to the predominant neighboring material. If the min_vol_cells argument is not specified, the minimum number of cells in a volume will be set to 5. This has the effect of removing small volumes that would otherwise be generated. This option will also remove protrusions, where a cell surrounded on 4 or 5 sides by another material ID will be reassigned to the predominant neighboring material. This option is available with multiple processors.

  See also the defeature_iters and defeature_bbox options for additional control of the defeature = filter option. The compare_volume option can also be used to validate that changes made to material volumes are within acceptable limits.
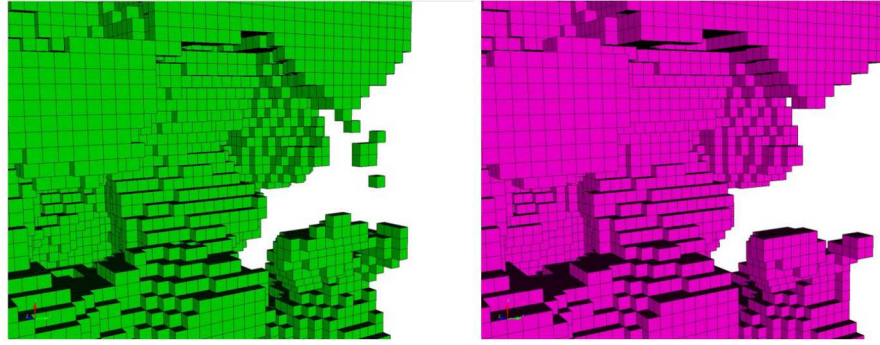
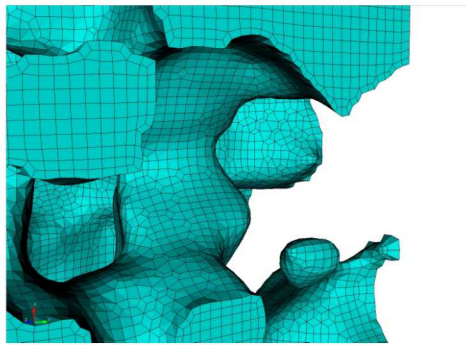**Figure 12-5. Example grid cells before and after defeaturing has been applied**

Figure 12-6. Final mesh after using defeaturing.

- `collapse (2)`: Curve and surface collapses are performed. This option is only available when used with the `trimesh` option. After geometry has been extracted and built from the volume fraction data curves containing exactly one mesh edge are collapsed into a single vertex. Surfaces that are identified with exactly 2 curves, each of which have 2 mesh edges are collapsed into a single curve. Only available as serial option (-j 1)
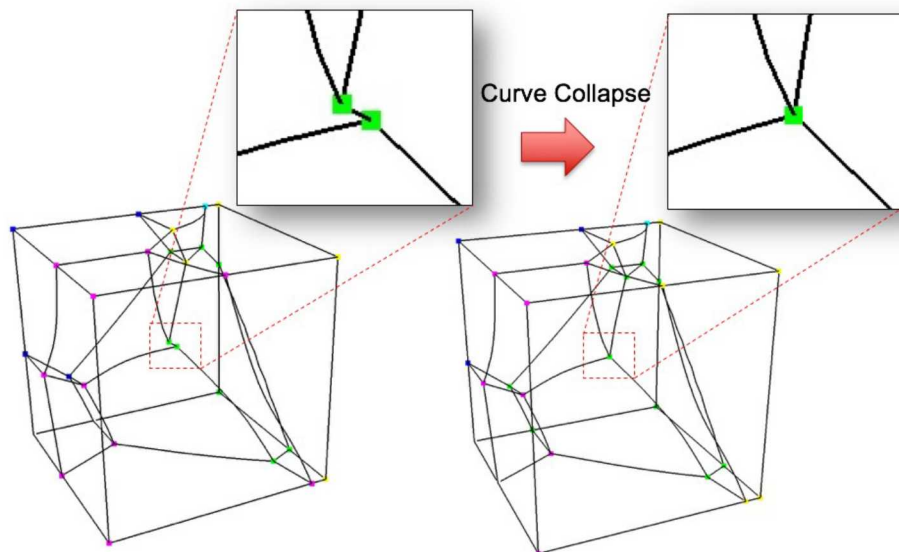
Figure 12-7. Example collapsing of small curve on microstructure model when using defeature=2 and trimesh option

- `filter_and_collapse (3)`: Performs both option `filter (1)` and `collapse (2)` on a trimesh. Only available as serial option (-j 1)

12.12. MINIMUM NUMBER OF CELLS IN A VOLUME

```
Command: min_vol_cells      Minimum number of cells in a volume
```

```
Input file command:    min_vol_cells <arg>
Command line options: -mvs <arg>
Argument Type:         integer >= 0
```

Command Description:

When used with `defeature` options `filter (1)` or `filter_and_collapse (3)`, specifies the minimum number of cells below which a volume will be eliminated. The cells of small volumes will be absorbed into the predominant material of the neighboring cells. If not specified and defeature options `filter (1)` or `filter_and_collapse (3)` are used, the `min_vol_cells` value will be set to 5.

12.13. DEFEATURE AT BOUNDING BOX

```
Command: defeature_bbox      Defeature Filtering at Bounding Box
```

```
Input file command:    defeature_bbox
Command line options: -dbb
```

Command Description:

The `defeature_bbox` option is used in conjunction with `defeature = filter (1)`. It is used to modify the defeature filter criteria at cells that are immediately adjacent to the Cartesian grid's domain boundary. It is most effective for microstructure data but can be used with any input format. The `defeature = filter (1)` option will remove protrusions identified by cells that are surrounded on 4 or 5 sides by another material. For cells that are at the domain boundary, cells will have missing adjacent cells on at least one face. If the `defeature_bbox=true` option is used, the missing adjacent cells are considered a different material and counted in the 4 or 5 surrounding cells with a different material. In contrast, the `defeature_bbox=false` option will not count the missing adjacent cells. Using the `defeature_bbox=true` has the effect of more aggressively modifying cells at the domain boundaries to avoid protrusions. The default for this option is `defeature_bbox=false`. It will be ignored if `defeature = filter (1)` is not used.

12.14. MAXIMUM NUMBER OF DEFEATURE ITERATIONS

```
Command: defeature_iters      Maximum Number of Defeaturing Iterations


Input file command:    defeature_iters <arg>
Command line options: -dfi <arg>
Argument Type:         integer >=0
```

Command Description:

Used with the `defeature` option. Controls the maximum number of iterations of defeature filtering that will be performed. Setting this value greater than the default of 10 can be useful for very noisy data where a significant number of iterations will need to be performed to resolve the geometry.

When performing non-manifold resolution, the defeature state of some of the cells may be effected. As a result, the defeaturing and non-manifold resolution procedures are performed in a loop until no further changes can be made. The `defeature_iters` sets the maximum number of defeature and non-manifold resolution procedures that will be performed. Note that if defeaturing reaches the maximum iteration value without completely resolving all non-manifold conditions, that subsequent sculpt procedures may not succeed. Set this value higher to allow the defeaturing and non-manifold resolution to run to completion. The `stair = 1` option can be used to interrogate the model to see where non-manifold conditions may still exist.

12.15. THICKEN A MATERIAL

```
Command: thicken_material      Expand a given material into surrounding cells


Input file command:    thicken_material <arg>
Command line options: -thm <arg>
Argument Type:         integer >= 0 floating point value (0.0->1.0)
```

Command Description:

Used with the `defeature` option. Add additional cells at the boundary of a given material. Takes two input values, a material and a volume fraction between 0 and 1. This option is useful for noisy input data that may not form contiguous volumes. Thickening a material may close small gaps making the material continuous. To perform the thicken operation, cells in adjacent materials are removed and reassigned to the indicated material. This option requires both a valid material ID and volume fraction value, where the volume fraction represents the amount of material to be added to each neighboring cell. For example:

```
thicken material = 1 0.2
```

```
thicken_material = 2 0.5
```
each neighboring cell to material 1 will change approximately 20 percent of its volume to be material 1. Other materials present in the cell will be decreased accordingly to maintain a sum of 1.0 for each cell. Additional material is accumulated in neighboring cells from each adjacent cell it shares with material 1, so that if for example a neighbor cell shares faces with three cells of material 1, it will add 0.6 (0.2 X 3) of material 1 volume fraction to the neighbor. If more than one `thicken_material` option is used, the thicken operation will be performed in the order they appear in the input. For the above example, material 1 would first be thickened, followed by material 2. If materials 1 and 2 are adjacent, thickening in this case, material 2 would take precedence, potentially removing cells from material 1 at their interface.
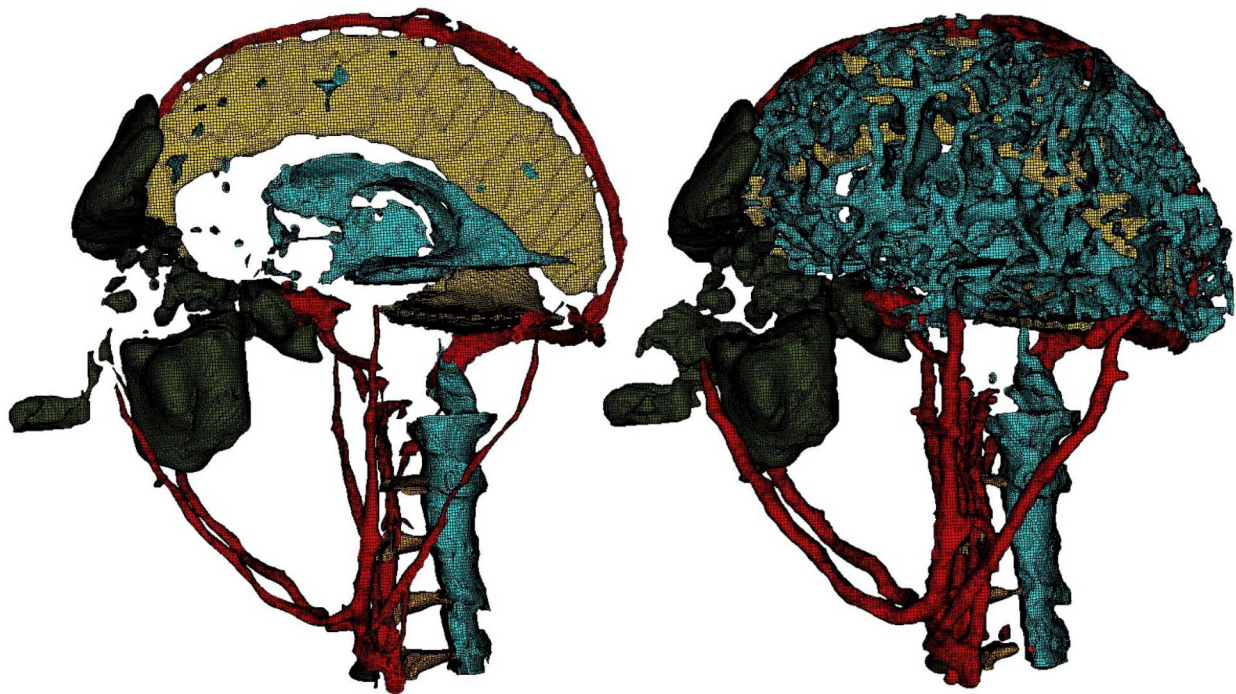

Figure 12-8. Bitmap input is used on a Cartesian base grid to generate the mesh for complex head and brain anatomy. Left: Some of the materials prior to applying the thicken_material option. Right: After applying the thicken_material option.

12.16. MICROSTRUCTURE EXPANSION

```
Command: micro_expand      Expand Microstructure grid by N layers

Input file command:   micro_expand <arg>
Command line options: -me <arg>
```

```
Argument Type:          integer >= 0
```

Command Description:

This option expands the Cartesian grid by a specified number of layers. It can be used with any of the following input options:

```
--input_micro
--input_cart_exo
--input_spn
```

In some cases the interior material interfaces may intersect the domain boundaries at small acute angles. When this occurs it may be difficult or impossible to achieve computable mesh quality at these intersections. To address this problem, one or more layers of hexes may be added to the Cartesian grid. The volume fractions from cells at the boundary are copied to generate additional layers. This has the effect of increasing the angle of intersection for any material interfaces intersecting the domain boundary. Usualy a value of 1 or 2 is sufficient to sufficiently improve quality.

Note that the resulting mesh in the expanded layers serves only to improve mesh quality and will only duplicate existing data at the boundaries. It may not reflect the actual material structure within the expansion layers.
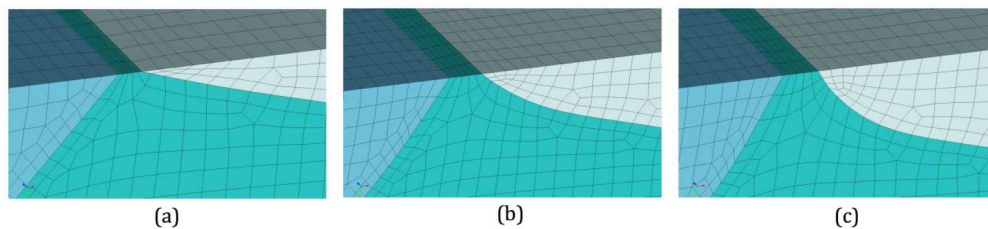

Figure 12-9. (a) Initial mesh (b) One expansion layer added (c) Two expansion layers added

12.17. MICROSTRUCTURE SHAVE

```
Command: micro_shave     Remove isolated cells at micro. boundaries

Input file command:   micro_shave
Command line options: -ms
```

Command Description:

This option potentially modifies the outermost layer of Cartesian cells of a microstructures file. It will identify isolated cells where the assigned material is unique from all of its surrounding cells at the boundary. When this occurs, the cell material is reassigned to the dominant nearby material.

This option is useful if it is noted that a cell structure just barely grazes the exterior planar boundary surface. Poor quality elements can often result with this condition. The micro_shave option will, in effect, remove material from the cell structure, but will result in better quality elements by removing the intersection region with the boundary.

micro_shave can be used with any of the following input options:

```
--input_micro
--input_cart_exo
--input_spn
```

12.18. REMOVE BAD ELEMENTS BELOW THRESHOLD

```
Command: remove_bad      Remove hexes with Scaled Jacobian < threshold

Input file command:    remove_bad <arg>
Command line options:  -rb <arg>
Argument Type:         floating point value -1.0 >= 1.0
```

Command Description:

Remove hexes below the specified scaled Jacobian metric.

13. MESH TRANSFORMATION

Sculpt options for applying transformations to the mesh following mesh generation. For cases where the initial geometry description may not be at the desired scale or bounds, the transformation options provide the ability to apply transformations to the node locations following the mesh generation procedure. This can be effective for microstructure models, where the size and location may be defined by the given intervals of the data.

```
Mesh Transformation  --transform  -tfm
  --xtranslate          -xtr <arg> Translate final mesh coordinates in X
  --ytranslate          -ytr <arg> Translate final mesh coordinates in Y
  --ztranslate          -ztr <arg> Translate final mesh coordinates in Z
  --scale               -scl <arg> Scale final mesh coordinates by constant
```

13.1. TRANSLATE MESH COORDINATES IN X

```
Command: xtranslate      Translate final mesh coordinates in X

Input file command:   xtranslate <arg>
Command line options: -xtr <arg>
Argument Type:        floating point value
```

Command Description:

Translate all mesh coordinates written to Exodus file by X delta distance.

13.2.   TRANSLATE MESH COORDINATES IN Y

```
Command: ytranslate      Translate final mesh coordinates in Y

Input file command:   ytranslate <arg>
Command line options: -ytr <arg>
Argument Type:        floating point value
```

Command Description:

Translate all mesh coordinates written to Exodus file by Y delta distance.

13.3.   TRANSLATE MESH COORDINATES IN Z

```
Command: ztranslate      Translate final mesh coordinates in Z

Input file command:   ztranslate <arg>
Command line options: -ztr <arg>
Argument Type:        floating point value
```

Command Description:

Translate all mesh coordinates written to Exodus file by Z delta distance.

13.4.   SCALE MESH COORDINATES BY CONSTANT

```
Command: scale      Scale final mesh coordinates by constant

Input file command:   scale <arg>
Command line options: -scl <arg>
Argument Type:        floating point value
```

Command Description:

Scale all mesh coordinates written to Exodus file by a constant scalar value.

14. BOUNDARY LAYERS

Sculpt options for defining boundary layers in the mesh. Boundary layers are thin hex layers that can be defined at surfaces, extending either inward or outward from a material. The user may specify the number and thickness of the hex layers as well as the material ID of the layers. Layer thicknesses should normally be "thin" with respect to the size of the cells. Layers will not intersect, so should be defined on surfaces where nearby layers will not overlap. Boundary layers are specified based upon a material ID, where hex layers will be placed at surfaces where the material interfaces with other materials, or at free surfaces.
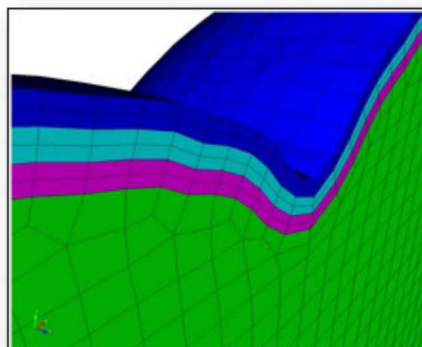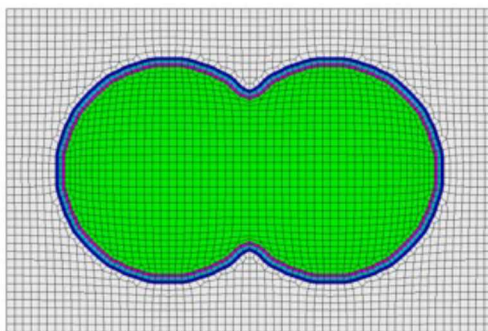

Figure 14-1. Example of boundary layers.

Figure 14-2. Boundary layers defined at the surfaces of a material.

```
Boundary Layers  --boundary_layer  -bly
```

```
--begin                 -beg <arg> Begin specification blayer or blayer_block
--end                   -zzz <arg> End specification blayer or blayer_block
--material              -mat <arg> Boundary layer material specification
--num_elem_layers       -nel <arg> Number of element layers in blayer block
--thickness             -th  <arg> Thickness of first element layer in block
--bias                  -bi  <arg> Bias of element thicknesses in blayer block
```

14.1. BOUNDARY LAYER BEGIN

```
Command: begin     Begin specification blayer or blayer_block

Input file command:   begin <arg>
Command line options: -beg <arg>
Argument Type:        blayer, blayer_block
```

Command Description:

Defines the beginning of a specification block. Must be closed with "end" argument. Currently supports the following specifications:

`blayer`
Defines a boundary layer specification. Layers of hex elements are placed at the interface of a given material. Valid argumnts used within a blayer specification include: material, and begin blayer_block.

`blayer_block`
Defines a set of element layers within a given blayer definition that share a common material ID. Valid arguments used within a blayer_block specification include: material, num_elem_layers, thickness and bias.

`Example:`
The following example shows a boundary layer specification in a sculpt input file. In this example, two boundary layer blocks are defined at the interface of materials 1 and 2. Two material blocks with ID 3 and 4 are generated with 1 and 2 element layers respectively.

```
BEGIN BLAYER
  MATERIAL = 1 2
  BEGIN BLAYER_BLOCK
    MATERIAL = 3
    NUM_ELEM_LAYERS = 1
    THICKNESS = 0.1
  END BLAYER_BLOCK
  BEGIN BLAYER_BLOCK
```

```
          MATERIAL = 4
          NUM_ELEM_LAYERS = 2
          THICKNESS = 0.2
          BIAS = 1.3
        END BLAYER_BLOCK
      END BLAYER
```
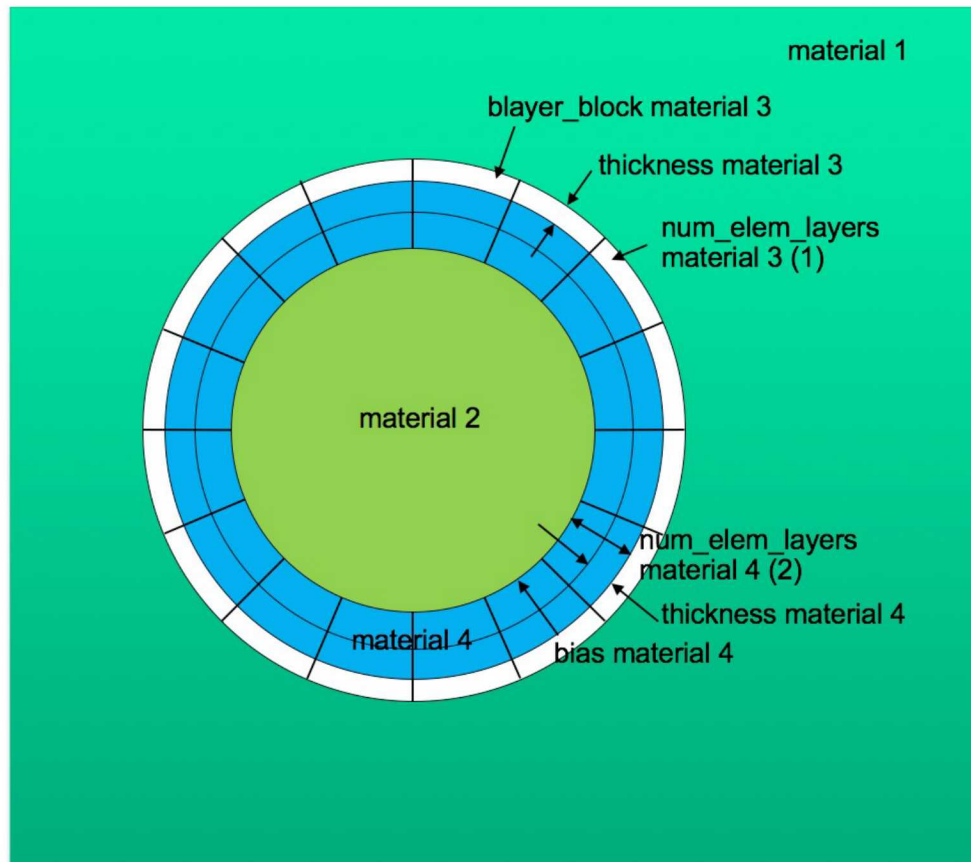

Figure 14-3. Example schema for boundary layers corresponding to input file below.

14.2. BOUNDARY LAYER END

```
Command: end      End specification blayer or blayer_block

Input file command:    end <arg>
Command line options: -zzz <arg>
Argument Type:         blayer, blayer_block
```

Command Description:

Defines the end of a specification block. Must be preceded with "begin" argument. Currently supports arguments blayer and blayer_block.

14.3. BOUNDARY LAYER MATERIAL

```
Command: material     Boundary layer material specification

Input file command:   material <arg>
Command line options: -mat <arg>
Argument Type:        integer > 0
```

Command Description:

Defines a material ID in a boundary layer specification. When used within a BLAYER specification, it references one or two existing materials in the input where boundary layers will be generated. If a single material is specified, hex layers will be generated at all interfaces of the designated material with any adjacent material. If two material IDs are specified, layers will be generated only at interfaces where the two materials are adjacent.

In most cases, the material ID(s) in the BLAYER specification refer to material IDs defined in the diatom file for specific geometry inserts such as STL files or diatom primitives. It can also be defined as the void material ID (VOID_MAT) or a material in a volume fraction description such as input_vfrac, input_micro, input_cart_exo or input_spn.

When used within a BLAYER_BLOCK specification, it refers to a new block that will be generated for which all elements in the blayer_block will be assigned. Normally it refers to a unique material ID that is not already referenced in the input. Where the material ID is already used, elements in the blayer block will be added to the existing material.

A material ID must be defined for both a BLAYER and BLAYER_BLOCK. This value does not have a default.

14.4. NUMBER OF ELEMENT LAYERS IN BOUNDARY LAYER

```
Command: num_elem_layers     Number of element layers in blayer block

Input file command:   num_elem_layers <arg>
Command line options: -nel <arg>
Argument Type:        integer > 0
```

Command Description:

Number of element layers to be defined within a BLAYER_BLOCK specification. num_elem_layers must be defined for all BLAYER_BLOCKs.

14.5. BOUNDARY LAYER THICKNESS

```
Command: thickness      Thickness of first element layer in block

Input file command:    thickness <arg>
Command line options: -th <arg>
Argument Type:         floating point value
```

Command Description:

Thickness of the first layer defined in a BLAYER_BLOCK. Value is an absolute distance. No default is provided and must be defined for all BLAYER_BLOCKs

---

14.6. BOUNDARY LAYER BIAS

```
Command: bias      Bias of element thicknesses in blayer block

Input file command:    bias <arg>
Command line options: -bi <arg>
Argument Type:         floating point value
```

Command Description:

Bias factor applied to additional layers of a BLAYER_BLOCK. Used in conjunction with the THICKNESS parameter (thickness of first layer) it defines a multiplier for the thickness for subsequent element layers defined within the same BLAYER_BLOCK. Default BIAS is 1.0 and is optional.

REFERENCES

[1] M. S. Ebeida, A. Patney, J.D. Owens, and E. Mestreau. Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. *International Journal for Numerical Methods in Engineering*, 88:974–985, 2011.

[2] Y. Ito, A. Shih, and B.K. Soni. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *International Journal for Numerical Methods in Engineering*, 77:1809–1833, 2009.

[3] L. Marechal. Advances in octree-based all-hexahedral mesh generation: Handling sharp features. *Proceedings of the 18th International Meshing Roundtable*, pages 65–84, 2009.

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

[5] 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.

[6] Steven J. Owen and Clifton R. Dudley. Degenerate hex elements. *Procedia Engineering, Proceedings 23rd International Meshing Roundtable*, 82:301–312, 2014.

[7] 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.

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

[9] Steven J. Owen and Jason F. Shepherd. Embedding features in a cartesian grid. *Proceedings 18th International Meshing Roundtable*, pages 117–138, 2009.

[10] 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.

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

[12] 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.

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

[14] 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. *Sandia National Laboratories SAND2019-3478, http://cubit.sandia.gov*, 2019.

[15] Robert Schneiders, F. Schindler, and F. Weiler. Octree-based generation of hexahedral element meshes. *Proceedings 5th International Meshing Roundtable*, pages 205–216, 1996.

[16] Jason F. Shepherd. Topologic and geometric constraint-based hexahedral mesh generation. *PhD. Thesis, University of Utah*, 2007.

[17] J. Yin and C. Teodosiu. Constrained mesh optimization on boundary. *Engineering with Computers*, 24:231–240, 2008.

[18] H. Zhang and G. Zhao. Adaptive hexahedral mesh generation based on local domain curvature and thickness using a modified grid-based method. *Finite Elements in Analysis and Design*, 43:691–704, 2007.

[19] 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.

[20] Y. Zhang, T. J. R. Hughes, and C. L. Bajaj. Automatic 3d mesh generation for a domain with multiple materials. *Proceedings 16th International Meshing Roundtable*, pages 367–386, 2007.

[21] Y. Zhang, X. Liang, and G. Xu. A robust 2-refinement algorithm in octree and rhombic dodecahedral tree based all-hexahedral mesh generation. *Proceedings 21st International Meshing Roundtable*, pages 155–172, 2013.

APPENDICES

O. EXAMPLES: USING CUBIT AS A FRONT-END TO SCULPT

The following examples use Cubit's graphical user interface as a front-end to the sculpt application. Each of the examples to follow use this simple geometry that can be generated in Cubit. Execute these commands in Cubit prior to performing the example `Sculpt Parallel` command line operations

```
sphere rad 1
sphere rad 1
vol 2 mov x 2
cylinder rad 1 height 2
vol 3 rota 90 about y
vol 3 mov x 1
unite vol all
```
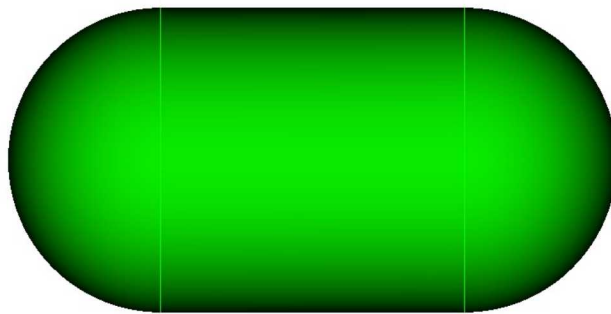
Figure O-1. Geometry created in Cubit from the above commands and used for the following examples.

A. BASIC SCULPT

This example illustrates use of Sculpt with all default options. So that we can view the result, we will also use the `overwrite`, `combine` and `import` options.

```
sculpt parallel
draw block all
```

The result of this operation is shown in Figure O-2. For this example, behind the scenes, Cubit built an input file for Sculpt, ran it on 4 processors, combined the resulting 4 meshes, and subsequently imported the resulting mesh into Cubit. Note that Volume 1 remains "unmeshed" and we have created a free mesh that is not associated with a volume. The result of any Sculpt command is always an unassociated free mesh.
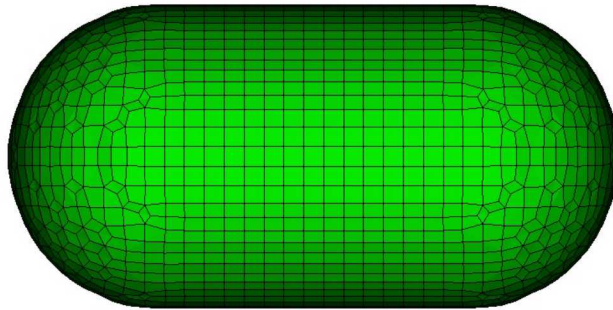

Figure O-2. Free mesh generated from sculpt command

B. SIZE AND BOUNDING BOX

This example illustrates the use of the `size` and `box` options

```
delete mesh
sculpt parallel size 0.1 box location position -1.5 0 -1.5
    location position 1 1.5 0
draw block all
```

In this case we have used the `size` option to define the base cell size for the grid. We have also used the `box` option to define a bounding box in which the mesh will be generated. Any geometry falling outside of the bounding box is ignored by Sculpt. Figure O-3 shows the mesh generated with this command.
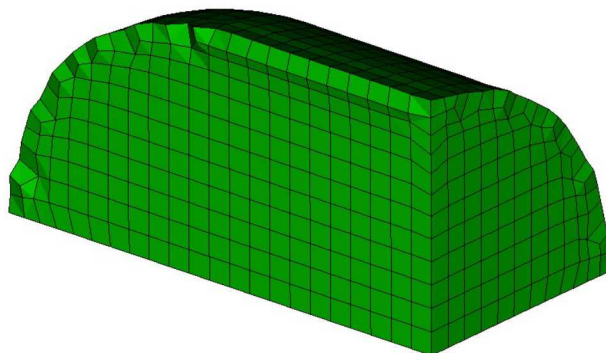

Figure O-3. Sculpt `box` option limits the extent of the generated mesh.

C. MESHING THE VOID

In this example we illustrate the use of the `void` option:

```
delete mesh
sculpt parallel size 0.1 box location position -1.5 0 -1.5
     location position 1 1.5 0 void
draw block all
```

The result is shown in figure O-4 Notice that this example is precisely the same as the last with the exception of the addition of the void option. Mesh is generated in the space surrounding the volume out to the extent of the bounding box. In this case, an additional material block is defined and automatically assigned an ID of 2. The nodes and element faces at the interface between the two blocks are shared between the two materials.
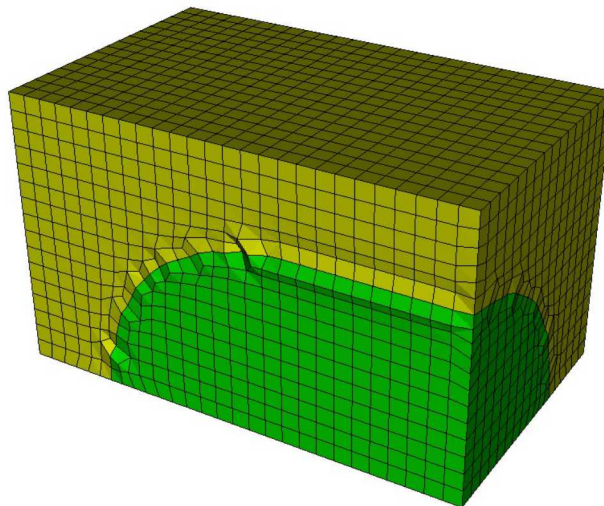
Figure O-4. Sculpt `void` operation generates mesh outside the volume.

D. AUTOMATIC SIDESET DEFINITION

In this example we illustrate the use of the `sideset` option.

Generating sidesets on the free mesh with Cubit: Sideset boundary conditions can be manually created on the resulting free mesh from Sculpt using the standard `Sideset <sideset_id> Face <id_range>` syntax. The `Group Seed` command is also useful in grouping faces based on a feature angle to be used in a single sideset.

Generating sidesets in Sculpt: Sculpt also provides several options for defining sidesets as part of the Sculpt run. The following illustrates one option:

```
delete mesh
sculpt parallel size 0.1 box location position -1.5 0 -1.5
    location position 1 1.5 0 void sideset 2
list sideset all
draw sideset all
```

Once again we use the same syntax but add the `sideset 2` option to automatically generate a series of sidesets. The `list` command should reveal that 10 sidesets were defined for this example with IDs 1 to 10. Figure O-5 shows the result of the `draw` command showing all of the sidesets in different colors. Note that for the `sideset 2` option, sidesets are created with the following criteria:

1. Interfaces between materials

2. Exterior surfaces

3. Surfaces at the domain boundary
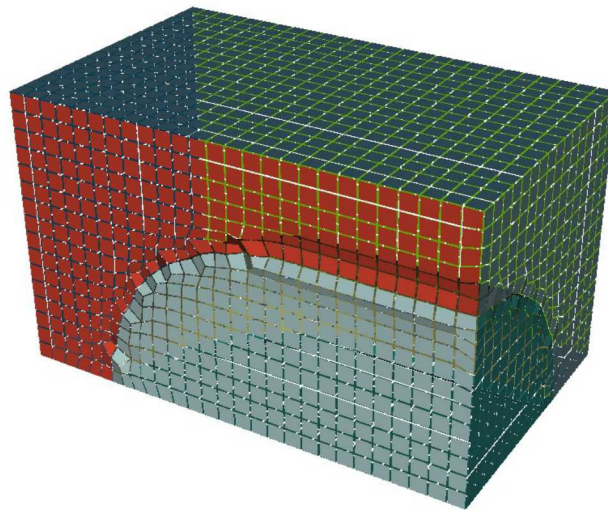
Figure O-5. Automatic sidesets created using Sculpt

E. RUNNING SCULPT STAND-ALONE

This example illustrates how to set up the files necessary to run Sculpt as a stand-alone process. This can be done on the same desktop machine or moved to a larger cluster machine more suited for parallel processing.

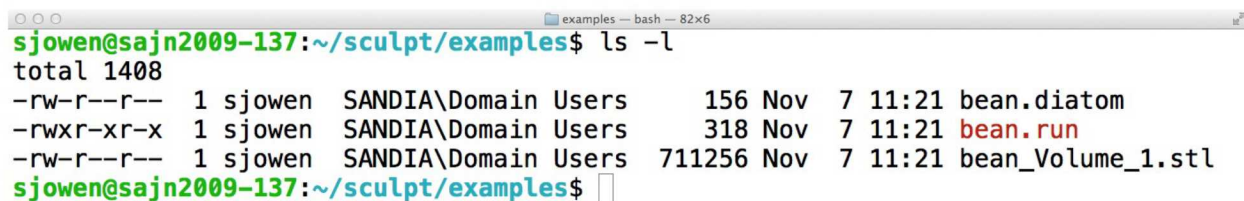Begin by setting your working directory in Cubit to a location that is convenient for placing example files:

```
cd "path/to/my/sculpt/examples"
```

Next we issue the basic `sculpt parallel` command to mesh the volume

```
delete mesh
sculpt parallel processors 8 fileroot "bean" over no_execute
```

In this case, we used the `no_execute` option which does not invoke the Sculpt application. Instead it will write a series of files to the working directory. The `fileroot` option defines the base file name for the files that will be written; in this case "bean". We also use the `processors` option to set the number of processors to be used to 8.

To see the files that Cubit placed in the working directory, bring up a terminal window on your desktop and change directories to the current working directory (ie. `cd path/to/my/sculpt/examples`). A directory listing should reveal 3 files as shown in Figure O-6.

Figure O-6. Directory listing of files written from Cubit

The following describes the purpose of each of the resulting files:

- `bean.diatom`: *Diatoms* is a file format used by Sandia's CTH and Alegra analysis programs that includes a rich constructive solid geometry definition. A series of directives for constructing and orienting primitives to build a complete solid model can be used. Included in the Diatom description is an STL import option. While any standard Diatom description may be used as input to Sculpt, for Cubit's purposes, only the STL option is used. This file contains a listing of all STL files that will be used as input to Sculpt.

- `bean.run`: The `.run` file contains the unix command line for running sculpt. Note that the file permissions have been set to execute to allow this file to be used as a unix script. Figure O-7 shows the `.run` file for this example. Note that the command uses `mpiexec` and the `psculpt` executables, along with their full path. These paths may need to be edited when running on a different machine. It also includes the default parameters for setting the sizes, bounding box and smoothing parameters that have been computed by Cubit.

- `bean_Volume_1.stl`: The STL file is a copy of the geometric model. In our case, it is a representation of the cylinder and sphere object we have been working with. The STL format is a set of triangles that describe the surfaces of the object. One STL file will be generated for each Volume. If the Block option is used, then one file for each Block would be created.

To run sculpt on the same machine, from the terminal window in your current working directory you would issue the following command:

```
●●●                    examples — vim — 73×8                       ↗
/usr/local/bin/mpiexec -np 8 /Users/sjowen/cubit/psculpt/camal-build/bin/
Release/psculpt -x 37 -y 21 -z 21 -t -1.414369 -u -1.370318 -v -1.370318
-q 3.414369 -r 1.370318 -s 1.370318   -S 1 -SI 12 -LI 2 -OT 0.600000
   -e /Users/sjowen/sculpt/examples/bean.diatom_result -d /Users/sjowen/sc
ulpt/examples/bean.diatom
~
~
```
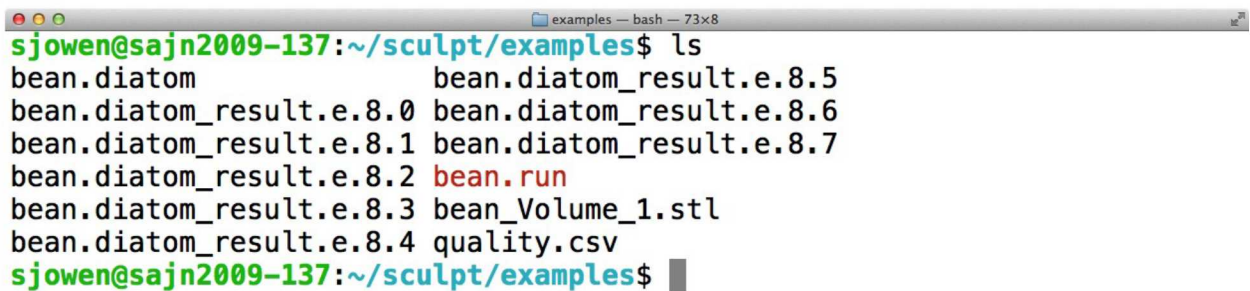
Figure O-7. Unix command line for running Sculpt generated by Cubit

```
    ./bean.run
```

If Sculpt is to be run on a different machine, copy the files in the working directory to the other machine and issue the same command. Remember to change the path to the `mpiexec` and `psculpt` executables to match those on the new machine. For running on cluster machines that have scheduling of resources, check with your system administrator for how to submit a job for running.

After running Sculpt, Figure 8 shows the resulting files that would be written to the current working directory.

```
●●●                    examples — bash — 73×8                      ↗
sjowen@sajn2009-137:~/sculpt/examples$ ls
bean.diatom              bean.diatom_result.e.8.5
bean.diatom_result.e.8.0 bean.diatom_result.e.8.6
bean.diatom_result.e.8.1 bean.diatom_result.e.8.7
bean.diatom_result.e.8.2 bean.run
bean.diatom_result.e.8.3 bean_Volume_1.stl
bean.diatom_result.e.8.4 quality.csv
sjowen@sajn2009-137:~/sculpt/examples$ ▮
```

Figure O-8. Eight Exodus files were generated and placed in working directory

Note that 8 exodus files have been generated, 1 from each processor. These files can be used by themselves or used as-is for use in a simulation, or they can be combined into a single file. The exodus files produced by Sculpt include all appropriate parallel communication information as defined by the Nemesis format. Nemesis is an extension of Sandia's Exodus II format that also includes appropriate parallel communication information.

To combine the resulting exodus files into a single file, we can use the `epu` tool. Epu should be included in your Cubit distribution, but may require you to set up appropriate paths for it to be recognized. To run `epu` on this model, use the following command from a unix terminal window:

```
    epu -p 8 bean.diatom_result
```

The result should be a single file with the name `bean.diatom_result.e`. The mesh in this file can then be imported into Cubit. Switch back to your Cubit application and from the command line type the following command:

```
import mesh "bean.diatom_result.e" no_geom
```

The result should be the same mesh we generated previously that is shown in Figure O-2.

F. MESHING MULTIPLE MATERIALS WITH SCULPT

This example illustrates using Sculpt to mesh models with multiple materials. To begin with, we will modify our current model by adding some additional volumes. Use the following commands:

```
delete mesh
cylinder rad 0.5 height 3
cylinder rad 0.5 height 3
vol 5 mov x 2
```

The resulting geometry should look like the image in Figure O-9
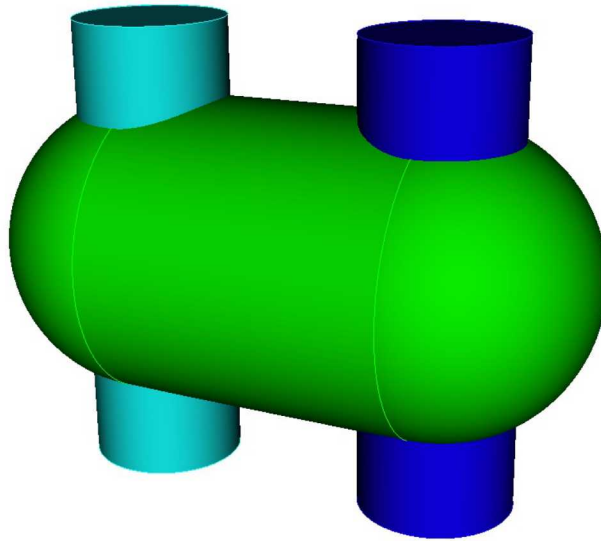

Figure O-9. Geometry used to demonstrate multiple materials with Sculpt

Use this geometry to generate a mesh using Sculpt.

```
sculpt parallel size 0.075
draw block all
```

The resulting mesh should look like the image in O-10.

Notice that one mesh block per volume was created. We should also note that no boolean operations were performed prior to building the mesh with Sculpt. In fact, volumes 4 and 5 were significantly
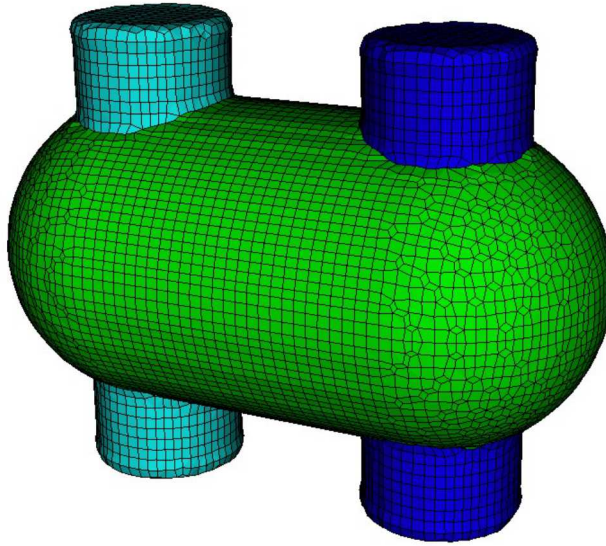
Figure O-10. Mesh generated on multiple materials

overlapping volume 1. This would be an invalid condition for normal Cubit meshing operations. Figure O-11 shows a cut-away image of the mesh using the Cubit Clipping Plane tool.

We should also note that `imprint/merge` operations typically needed for traditional meshing operations in Cubit, were also not required. While it is usually best to avoid overlaps to avoid ambiguities in the topology, Sculpt is able to generate a mesh giving precedence to the most recently defined materials. Merging is performed strictly by geometric proximity. Volumes closer than about one half the user input size will normally be automatically merged.

Next, we will examine the mesh quality of the free mesh. The following command will display a graphical representation of the Scaled Jacobian metric.

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

The result is shown in Figure O-12. Note the elements (colored red) at the interface between materials are unacceptable for simulation. This is caused by the Sculpt algorithm projecting nodes to a common curve interface shared by the materials.

In most cases, the poor element quality at material interfaces can be improved by using the `pillow` option. Adding this option will direct Sculpt to add an additional layer of elements surrounding each surface. To see the result of pillowing, issue the following commands:

```
delete mesh
sculpt parallel size 0.075 over combine import pillow 1
quality hex all scaled jacobian draw mesh
```
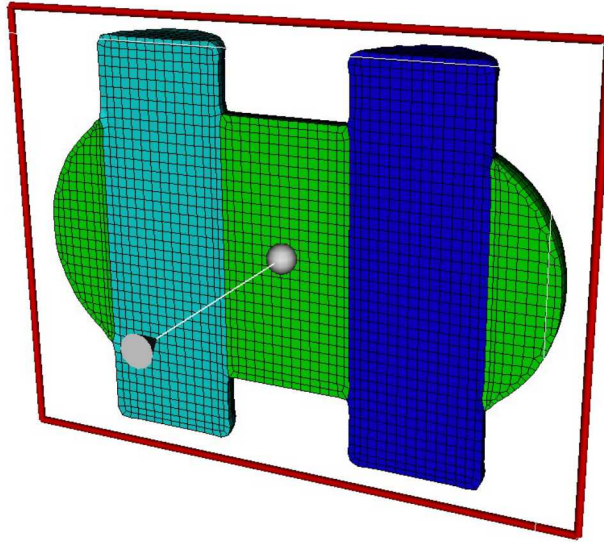
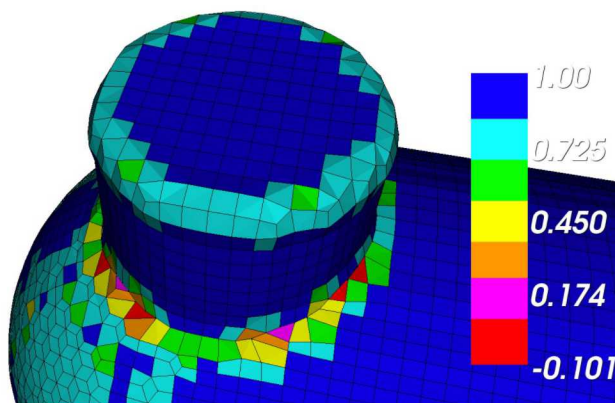**Figure O-11. Cut-away of mesh generated on multiple materials**

Figure O-12. Mesh quality of multi-material mesh

The resulting mesh is showed in Figure O-13. Note the improved mesh quality at the shared curve interface. A closer look at the mesh, shown in Figure O-14. reveals the additional layer of hexes surrounding each surface that allows for improved mesh quality when compared with Figure O-11. When generating meshes with multiple materials that must share common interfaces, the `pillow` option is usually recommended.
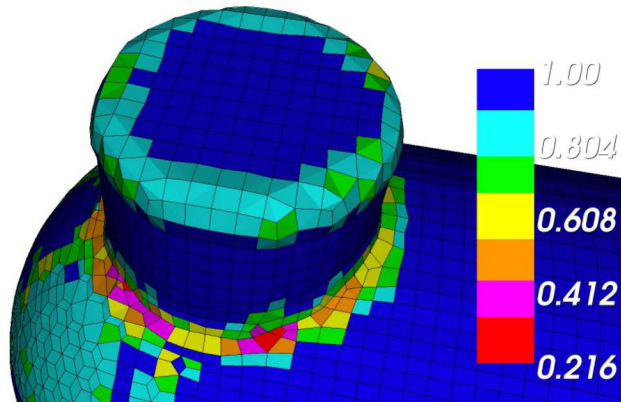
Figure O-13. Mesh quality of multi-material mesh using pillow option
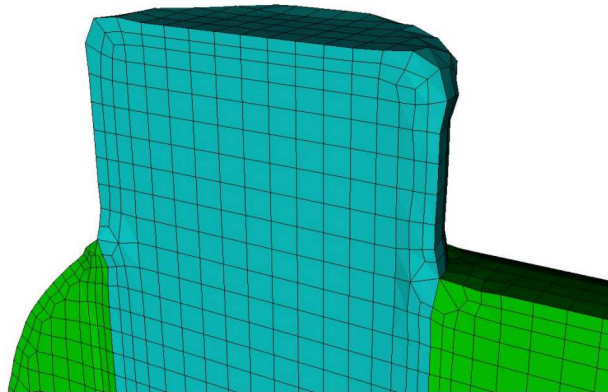

Figure O-14. Cutaway of mesh reveals the additional layer of hexes surrounding each surface when the pillow option is used

P. EXAMPLES: USING THE COMMAND-LINE SCULPT APPLICATION

The following illustrate simple use cases of the Sculpt application. To use these examples, copy the stl and diatom files from appendix Q to your working directory and name them `brick1.stl`, `brick2.stl` and `bricks.diatom` respectively.

A. MESHING A SINGLE STL VOLUME

```
sculpt -j 4 -stl brick1.stl -cs 0.5
```

Runs sculpt with 4 processors with geometry input from `brick1.stl`. Uses a base Cartesian cell size of 0.5. The bounding box and all other parameters will be defaulted. The result should be the 4 exodus files:

```
brick1.stl_results.e.4.0
brick1.stl_results.e.4.1
brick1.stl_results.e.4.2
brick1.stl_results.e.4.3
```

These files can be combined into a single file using the SEACAS tool epu

```
epu -p 4 brick1.stl_results
```

The result of this operation should be a single file:

```
brick1.stl_results.e
```

To view the resulting mesh in Cubit, use the import free mesh command. For example:

```
import mesh "brick1.stl_results.e" no_geom
```
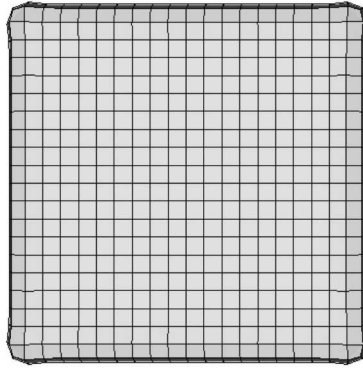
Figure P-1. Resulting mesh from example of single STL volume

B. MESHING MULTIPLE STL VOLUMES

```
mpiexec -np 4 psculpt -x 46 -y 26 -z 26 -t -6.5 -u -6.5 -v -6.5
    -q 16.5 -r 6.5 -s 6.50 -d bricks.diatom
```

In this case we use mpiexec to start 4 processes of psculpt. We explicitly define the number of Cartesian intervals and the dimensions of the grid. Rather than using the -stl option, we use the -d option which allows us to specify the diatom file, bricks.diatom. This file allows us to specify multiple stl files, where each one represents a different material. In this case we use both brick1.stl and brick2.stl, which are called out in bricks.diatom.
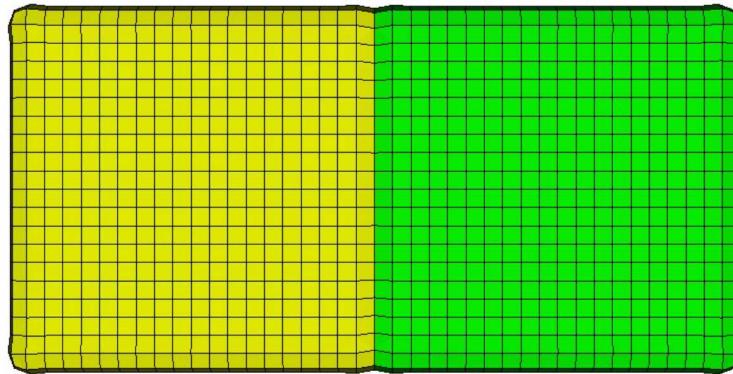

Figure P-2. Resulting mesh from example of two STL volumes

Q. EXAMPLE FILES

A. BRICK1.STL

```
solid Body 1
  facet normal 0.000000e+00 0.000000e+00 1.000000e+00
    outer loop
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex -5.000000e+00 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 0.000000e+00 1.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex -5.000000e+00 5.000000e+00 5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal -0.000000e+00 0.000000e+00 -1.000000e+00
    outer loop
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 -1.000000e+00 0.000000e+00
    outer loop
      vertex -5.000000e+00 -5.000000e+00 5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
```

```
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 -1.000000e+00 -0.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
    outer loop
      vertex -5.000000e+00 5.000000e+00 5.000000e+00
      vertex -5.000000e+00 5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
    outer loop
      vertex -5.000000e+00 -5.000000e+00 5.000000e+00
      vertex -5.000000e+00 5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 1.000000e+00 0.000000e+00
    outer loop
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 1.000000e+00 0.000000e+00
    outer loop
      vertex -5.000000e+00 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 1.000000e+00 -0.000000e+00 0.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
    endloop
  endfacet
```

```
    facet normal 1.000000e+00 -0.000000e+00 0.000000e+00
      outer loop
        vertex 5.000000e+00 5.000000e+00 5.000000e+00
        vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
        vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      endloop
    endfacet
endsolid Body 1
```

B. BRICK2.STL

```
solid Body 1
  facet normal 0.000000e+00 0.000000e+00 1.000000e+00
    outer loop
      vertex 1.500000e+01 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex 1.500000e+01 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 0.000000e+00 1.000000e+00
    outer loop
      vertex 1.500000e+01 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
    outer loop
      vertex 1.500000e+01 -5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 1.500000e+01 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal -0.000000e+00 0.000000e+00 -1.000000e+00
    outer loop
      vertex 1.500000e+01 5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 -1.000000e+00 0.000000e+00
    outer loop
```

```
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 1.500000e+01 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 -1.000000e+00 -0.000000e+00
    outer loop
      vertex 1.500000e+01 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex 1.500000e+01 -5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
    outer loop
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 1.000000e+00 0.000000e+00
    outer loop
      vertex 1.500000e+01 5.000000e+00 5.000000e+00
      vertex 1.500000e+01 5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
    endloop
  endfacet
  facet normal 0.000000e+00 1.000000e+00 0.000000e+00
    outer loop
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
      vertex 1.500000e+01 5.000000e+00 -5.000000e+00
      vertex 5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet
  facet normal 1.000000e+00 -0.000000e+00 0.000000e+00
    outer loop
      vertex 1.500000e+01 -5.000000e+00 5.000000e+00
      vertex 1.500000e+01 -5.000000e+00 -5.000000e+00
      vertex 1.500000e+01 5.000000e+00 5.000000e+00
```

```
      endloop
    endfacet
    facet normal 1.000000e+00 -0.000000e+00 0.000000e+00
      outer loop
        vertex 1.500000e+01 5.000000e+00 5.000000e+00
        vertex 1.500000e+01 -5.000000e+00 -5.000000e+00
        vertex 1.500000e+01 5.000000e+00 -5.000000e+00
      endloop
    endfacet
endsolid Body 1
```

C. BRICKS.DIATOM

```
diatoms
  package 'Brick1'
    material 1
    iterations 3
    insert stl
      FILE = 'brick1.stl'
    endinsert
  endp
  package 'Brick2'
    material 2
    iterations 3
    insert stl
      FILE = 'brick2.stl'
    endinsert
  endp
enddia
```

DISTRIBUTION

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Steven J. Owen	01543	sjowen@sandia.gov
W. Roshan Quadros	01543	wrquadr@sandia.gov
Michael Skroch	01543	mjskroc@sandia.gov
Corey D. Ernst	01543	cdernst@sandia.gov
Clinton S. Stimpson	01543	cjstimp@sandia.gov
Matthew L. Staten	01543	mlstate@sandia.gov
Martin W. Heinstein	01545	mwheins@sandia.gov
Micheal W. Glass	01545	mwglass@sandia.gov
Judith A. Brown	01516	judbrow@sandia.gov
Hojun Lim	01864	hnlim@sandia.gov
Kevin N. Long	01554	knlong@sandia.gov
James W. Foulk III	08363	jwfoulk@sandia.gov
Nathan W. Moore	01344	nwmoore@sandia.gov
Corbett Battaile	01864	ccbatta@sandia.gov
Theron Rogers	01864	trodger@sandia.gov
Micheal Powell	01443	micpowe@sandia.gov
James Brian	01443	jbcarle@sandia.gov
Chad Hovey	05421	chovey@sandia.gov
Trevor Hensley	09326	thensle@sandia.gov
Allen C Robinson	01443	acrobin@sandia.gov
Russell Daniel Teeter	01555	rdteete@sandia.gov
Byoung Yoon Park	08862	bypark@sandia.gov
Technical Library	01177	libref@sandia.gov