User Manual for the PROTEUS Mesh Tools

Revision 2.0

Nuclear Engineering Division
About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory’s main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE’s SciTech Connect (http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandra, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: orders@ntis.gov

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.
User Manual for the PROTEUS Mesh Tools

Revision 2.0

prepared by
Micheal A. Smith and Emily R. Shemon
Nuclear Engineering Division, Argonne National Laboratory

September 19, 2016
ABSTRACT

PROTEUS is built around a finite element representation of the geometry for visualization. In addition, the PROTEUS-SN solver was built to solve the even-parity transport equation on a finite element mesh provided as input. Similarly, PROTEUS-MOC and PROTEUS-NEMO were built to apply the method of characteristics on unstructured finite element meshes. Given the complexity of real world problems, experience has shown that using commercial mesh generator to create rather simple input geometries is overly complex and slow. As a consequence, significant effort has been put into place to create multiple codes that help assist in the mesh generation and manipulation.

There are three input means to create a mesh in PROTEUS: UFMESH, GRID, and NEMESH. At present, the UFMESH is a simple way to generate two-dimensional Cartesian and hexagonal fuel assembly geometries. The UFmesh input allows for simple assembly mesh generation while the GRID input allows the generation of Cartesian, hexagonal, and regular triangular structured grid geometry options. The NEMESH is a way for the user to create their own mesh or convert another mesh file format into a PROTEUS input format.

Given that one has an input mesh format acceptable for PROTEUS, we have constructed several tools which allow further mesh and geometry construction (i.e. mesh extrusion and merging). This report describes the various mesh tools that are provided with the PROTEUS code giving both descriptions of the input and output. In many cases the examples are provided with a regression test of the mesh tools. The most important mesh tools for any user to consider using are the MT_MeshToMesh.x and the MT_RadialLattice.x codes. The former allows the conversion between most mesh types handled by PROTEUS while the second allows the merging of multiple (assembly) meshes into a radial structured grid. Note that the mesh generation process is recursive in nature and that each input specific for a given mesh tool (such as .axial or .merge) can be used as “mesh” input for any of the mesh tools discussed in this manual.
# TABLE OF CONTENTS

Abstract ............................................................................................................................................ i  
Table of Contents ............................................................................................................................ ii  
List of Figures ................................................................................................................................ iii  
List of Tables .................................................................................................................................. iv  
1. Introduction ............................................................................................................................... 5  
2. MT_ApplyACRtoMesh: A Mesh Tool to Restrict the Element Block Size ............................. 8  
3. MT_BuildAxialMesh.x: A Mesh Tool to Extrude 2D meshes to 3D meshes ........................ 10  
4. MT_ChangeFEbasis.x: A Mesh Tool to Alter the Basis Order of a Mesh .............................. 12  
5. MT_DeformMesh.x: A Mesh Tool to Deform 3D Meshes .................................................... 14  
6. MT_Extrude2Dto3D.x: A Mesh Tool to Extrude 2D meshes to a 3D slice ........................... 17  
7. MT_FixCUBITHex27Issue.x: A Mesh Tool to Fix CUBIT 27 Node Hexahedrons ............... 18  
8. MT_IdentifyVertices.x: A Mesh Tool to Identify Vertices in a Mesh ................................. 20  
9. MT_MeshRestrictForBGP.x: A Mesh Tool to Restrict Element Block Sizes ...................... 22  
10. MT_MeshToBGPmesh.x: A Mesh Tool to Create a Pre-Partitioned Mesh ................. 23  
11. MT_MeshToHDF5.x: A Mesh Tool to Create a VISIT readable HDF5 File ....................... 25  
12. MT_MeshToMesh.x: A Mesh Tool to Convert Between Mesh Formats ....................... 26  
13. MT_MeshToVTK.x: A Mesh Tool to Create a VISIT Readable VTK File ....................... 27  
14. MT_RadialLattice.x: A Mesh Tool to Merge Meshes on a Radial Grid ......................... 28  
15. MT_Refine2Dmesh.x: A Mesh Tool to Automatically Refine a 2D Mesh ....................... 32  
16. MT_ModifyMesh.x: A Mesh Tool to Rotate and Translate a Mesh .................................. 33  
17. MT_DataToVTK.x: A Mesh Tool to Plot Solution data Via a VTK File ......................... 35  
18. Summary ................................................................................................................................. 37  
References ..................................................................................................................................... 38  
Appendix A. UFmesh: A User Friendly Mesh Format ................................................................. 39  
Appendix B. Grid: A File Format Specification for Structured Grid Input ............................ 48  
Appendix C. NEMESH: A PROTEUS File Format for Finite Element Mesh Input ............... 59
LIST OF FIGURES

Figure 1. ATR Mesh from CUBIT (left) with 20 cm³ (middle) and 1 cm³ (right) “acvolume” Restrictions .............................................................. 9
Figure 2. Example MT_BuildAxialMesh.x Extrusion of a 2D Pin-cell Mesh. ...................... 11
Figure 3. Example MT_ChangeFEBasis.x Change from a Linear (left) to Quadratic (right) Mesh ................................................................. 12
Figure 4. Example MT_DeformMesh.x Deformation Before and After Meshes. ............... 14
Figure 5. Example MT_Extrude2Dto3D.x Change from a 2D (left) to a 3D (right) Mesh. .... 17
Figure 6. Example MT_FixCUBITHex27Issue.x Correction of a Bad Hex 27 Element (left). 18
Figure 7. Example MT_IdentifyVertices.x Box Selection of a Pin in the ATR Mesh (right). 20
Figure 8. Example MT_IdentifyVertices.x Output ......................................................... 21
Figure 9. Example MT_MeshToBGPmesh.x Vertex Rank (left) and Element Rank (right). 24
Figure 10. Example MT_MeshToVTK.x Coloring of the ATR Mesh ................................ 27
Figure 11. Example MT_RadialLattice.x Mesh Demonstrating Complete Problem Creation. 31
Figure 12. Example MT_Refine2Dmesh.x Barycenter Refinement of a 2D Mesh ................ 32
Figure 13. Example MT_DataToVTK.x Plot of an Element Vector Field ......................... 35
Figure 14. Example MT_DataToVTK.x Data File Containing Seven Fields ....................... 36
Figure 15. Face (outside) and Sector (inside) Numbering for Cartesian (left) and hexagonal (right) Geometry ..................................................... 43
Figure 16. Example Cartesian Assembly UFmesh File ...................................................... 44
Figure 17. Example Hexagonal Assembly UFmesh File ..................................................... 44
Figure 18. Example Hexagonal Homogeneous Pin-cell UFmesh File .............................. 45
Figure 19. Example Hexagonal Sector and Face Meshing Control UFmesh File ............... 45
Figure 20. Example Cartesian Sector and Face Meshing Control UFmesh File ............... 46
Figure 21. Example Hexagonal Pin-cell Lattice without a Duct Restriction on the Geometry 47
Figure 22. Example Two-Dimensional Cartesian Grid File ............................................. 54
Figure 23. Example Three-Dimensional Cartesian Grid File ......................................... 55
Figure 24. Example Three-Dimensional Hexagonal Grid File ....................................... 56
Figure 25. Example Three-Dimensional Regular Triangular Grid File ........................... 57
Figure 26. Grid Finite Element Meshing Schemes (Regular Triangular shows 6 nodes) ..... 58
LIST OF TABLES

Table 1. Mesh Tool Summary ................................................................. 6
Table 2. Command Line Options for the MT_ApplyACRtoMesh.x Mesh Tool ........................................... 8
Table 3. Command Line Options for the MT_BuildAxialMesh.x Mesh Tool ........................................ 10
Table 4. Control Input for the MT_BuildAxialMesh.x Mesh Tool .................................................... 10
Table 5. Command Line Options for the MT_ChangeFEbasis.x Mesh Tool ........................................ 12
Table 6. Mesh File Format Control Input for *.SwitchType ......................................................... 13
Table 7. Command Line Options for the MT_DeformMesh.x Mesh Tool ......................................... 14
Table 8. Control Input File Format for MT_MeshDeform.x ...................................................... 15
Table 9. Command Line Options for the MT_Extrude2Dto3D.x Mesh Tool .................................. 17
Table 10. Command Line Options for the MT_FixCUBITHex27Issue.x Mesh Tool ....................... 18
Table 11. Command Line Options for the MT_IdentifyVertices.x Mesh Tool .................................. 20
Table 12. Command Line Options for the MT_MeshRestrictForBGP.x Mesh Tool ....................... 22
Table 13. Command Line Options for the MT_MeshToBGPmesh.x Mesh Tool ............................ 23
Table 14. Command Line Options for the MT_MeshToHDF5.x Mesh Tool ................................... 25
Table 15. Example Color Map for the ATR Mesh ........................................................................ 25
Table 16. Command Line Options for the MT_MeshToMesh.x Mesh Tool .................................. 26
Table 17. Command Line Options for the MT_MeshToVTK.x Mesh Tool .................................... 27
Table 18. Command Line Options for the MT_RadialLattice.x Mesh Tool ......................................... 28
Table 19. Control Input for the MT_RadialLattice.x Mesh Tool ...................................................... 30
Table 20. Control Input of the hex01.axial File for the MT_RadialLattice.x Example ...................... 30
Table 21. UFmesh Input of the hex01.ufmesh File for the MT_RadialLattice.x Example .......... 31
Table 22. General Purpose Mesh Merge Control Input .................................................................... 31
Table 23. Command Line Options for the MT_Refine2Dmesh.x Mesh Tool ................................... 32
Table 24. Command Line Options for the MT_ModifyMesh.x Mesh Tool .................................... 33
Table 25. Command Line Options for the MT_MeshToVTK.x Mesh Tool .................................... 35
Table 26. Keyword Input Options for the UFmesh File Format ...................................................... 42
Table 27. Keyword Input Options for the GRID File Format ..................................................... 52
1. Introduction

The PROTEUS code system [1] has a built in finite element library which provides multiple options. The finite element library is defined by its data structure setup which consists of a serial instance called NTmesh and a parallel instance called PNTmesh. The latter of these contains the parallel details of a given mesh which maps the locally visible mesh information into the global quantities on the parallel system. For PROTEUS-SN, one needs to provide a mesh as input and while mesh generation tools such as CUBIT [2] are available, they can be considerably complicated to use for simple geometries. In this report, we cover all of the mesh utility tools that are provided with the PROTEUS package which can be used to reduce the difficulties of using the commercial mesh tools.

There are three input means to create a mesh in PROTEUS: UFMESH, GRID, and NEMESH. At present, the UFMESH is a simple way to generate two-dimensional Cartesian and hexagonal fuel assembly geometries. The description of the UFmesh input is given in Appendix A. The GRID input is a simple way to generate conventional homogenized geometries for structured grid codes like PROTEUS-NODAL and contains options for Cartesian, hexagonal, and regular triangular geometry options. The input description of GRID is given in Appendix B and because it does its visualization using the finite element method, one can use the GRID input to construct a finite element mesh of the domain. The NEMESH is a way for the user to create their own mesh or convert another mesh file format into a PROTEUS input format. The input description of NEMESH is given in Appendix C. Other than these three input options, one must provide a mesh obtained from a commercial mesh generator in the EXODUS format for which we provide a convertor with PROTEUS [1].

These input options form the base of any mesh generation tool that is described in this manual. Given that one has an input mesh format acceptable for PROTEUS, Table 1 lists the mesh tools available to the user to assist in further mesh construction. As can be seen, there are numerous codes, some of which are more important than others. We discuss each code in a separate section along with its input. For visualization, we suggest using the VISIT [3] software which has an HDF5 [4] plugin that is compatible with the PROTEUS code package [1]. As an alternative, one can use the VTK file format which is supported by all tools to look at any step of the mesh construction or manipulation process. Each time a VTK file is exported, an additional “regionmap” file is created which is meant to easily be modified into a colormap file. The MT_MeshToVTK.x code explains how to setup a colormap file each time it is executed without a valid colormap file and we suggest consulting that section of the report for details of this mesh code. Note that the mesh generation process is recursive in nature and that each input specific for a given mesh tool (such as .axial or .merge) can be used as mesh input for any of the mesh tools.
<table>
<thead>
<tr>
<th>PROTEUS Mesh Tool</th>
<th>Purpose</th>
<th>Common Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT_ApplyACRtoMesh.x</td>
<td>Constrain the number of elements placed into each block to a fixed volumetric size</td>
<td>When using the sub-group API or coupling, this tool allows the mesh to be partitioned into smaller pieces</td>
</tr>
<tr>
<td>MT_BuildAxialMesh.x</td>
<td>Take a 2D mesh and build a 3D extruded mesh</td>
<td>Most often used to get around the poor performance of 3D mesh generators</td>
</tr>
<tr>
<td>MT_ChangeFEbasis.x</td>
<td>Take any mesh and alter the FE basis. This introduces (or removes) the extra points needed to define a higher order mesh</td>
<td>Test the impact of spatial basis order on the solution accuracy. Can also be used to test for bad elements in a mesh</td>
</tr>
<tr>
<td>MT_DeformMesh.x</td>
<td>Take any 3D mesh and deform it in X, Y, and Z directions according to user-specified strain functions</td>
<td>Quickly generate a simplistic deformed configuration to test reactivity worth with PROTEUS-SN</td>
</tr>
<tr>
<td>MT_Extrude2Dto3D.x</td>
<td>Take a 2D mesh and create a one element tall 3D mesh</td>
<td>Typically used as optional prestep to BuildAxialMesh</td>
</tr>
<tr>
<td>MT_FixCUBITHex27Issue.x</td>
<td>Fix badly generated hexahedron-27 elements created by CUBIT</td>
<td>All CUBIT meshes with this type of element and curvilinear geometry need to be corrected</td>
</tr>
<tr>
<td>MT_IdentifyVertices.x</td>
<td>Identify the set of vertices that lie within a given box of the geometric domain</td>
<td>Usually used to identify specific vertices for a fixed source definition</td>
</tr>
<tr>
<td>MT_MeshRestrictForBGP.x</td>
<td>Reduce the number of elements in each block to reduce the communication costs on Blue Gene/P</td>
<td>Only appropriate for Blue Gene/P and thus outmoded</td>
</tr>
<tr>
<td>MT_MeshToBGPmesh.x</td>
<td>Create a parallel partitioned mesh which reduces the load time on Blue Gene/P</td>
<td>Appropriate for any large scale computing machine</td>
</tr>
<tr>
<td>MT_MeshToHDF5.x</td>
<td>Take a PROTEUS mesh file and generate the native HDF5 format readable by CUBIT</td>
<td>Typically used to check the HDF5 setup of a given user mesh</td>
</tr>
<tr>
<td>MT_MeshToMesh.x</td>
<td>Take a PROTEUS mesh file in one format and convert it to one of the formats available in PROTEUS</td>
<td>Typically used to check a UFMESH or GRID mesh before using in PROTEUS</td>
</tr>
<tr>
<td>MT_MeshToVTK.x</td>
<td>Take a PROTEUS mesh file and generate a VTK file with a specific region color assignment</td>
<td>Using the regionmap output, one can easily construct the colormap table which allows easy visualization in VISIT</td>
</tr>
<tr>
<td>PROTEUS Mesh Tool</td>
<td>Purpose</td>
<td>Common Use</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MT_RadialLattice.x</td>
<td>Take a series of meshes and merge them in a Cartesian or hexagonal grid pattern</td>
<td>To build a complete 3D reactor geometry given a set of assembly meshes</td>
</tr>
<tr>
<td>MT_Refine2Dmesh.x</td>
<td>Perform a triangular mesh refinement on a given 2D mesh</td>
<td>Only usable on 2D meshes and only done to study spatial mesh refinement issues</td>
</tr>
<tr>
<td>MT_ModifyMesh.x</td>
<td>Perform a Rotation or Translation of a mesh</td>
<td>Used to translate meshes produced by CUBIT so that merge can easily be used</td>
</tr>
<tr>
<td>MT_DataToVTK.x</td>
<td>Take an element or vertex sized vector field from a data file and create a VTK file that allows it to be plotted</td>
<td>Used to circumvent HDF5 problems and allow other physics codes to plot solutions or error distributions on a given mesh</td>
</tr>
</tbody>
</table>
2. MT_ApplyACRtoMesh: A Mesh Tool to Restrict the Element Block Size

The focus of this section is the MT_ApplyACRtoMesh code which only has command line input. Table 2 shows the command line input options that are required for this mesh tool to operate which are relatively simple.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>acvolume</td>
<td>The max volume of all elements stored in any given block</td>
</tr>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

Example: ./MT_ApplyACRtoMesh.x 1.0 ATR.ascii ATR_1.0.ascii

The only input that needs to be explained in Table 2 is the “acvolume” input. In the base CUBIT [2] geometry, the ATR [6] is very complex and it is unrealistic to define every plate as a different region, especially if the composition is the same. Figure 1 can help to understand the impact that the acvolume has on the mesh where we give every storage block of elements a color assignment. From Figure 1, the original CUBIT mesh is observed to have 41 different regions (and thus the user must define 41 compositions to map to the ATR geometry). The use of the mesh tool with an acvolume setting of 20 cm³ increases the number of blocks to 4026 while the acvolume setting of 1 cm³ yields 21780 blocks. Clearly it is unmanageable to require the user to input 21780 compositions and the mapping information for it, let alone create a CUBIT geometry with 21780 regions. What actually happens inside of PROTEUS is that all of these new blocks are named the same as the originating blocks and thus the user’s original 41 compositions and assignments remain the same. Internally, each block is given its own cross section storage and thus this tool allows the user to set the granularity of the cross section assignment external to PROTEUS and visualize it much like we have done here.

It is important to note that with a fixed cross section methodology like that typically used for fast spectrum reactors, the use of this mesh tool is not needed. The sub-group API is where we anticipate this functionality to be most useful whether it be applied externally to PROTEUS like this mesh tool allows, or directly within PROTEUS which we are working on.
Figure 1. ATR Mesh from CUBIT (left) with 20 cm³ (middle) and 1 cm³ (right) “acvolume” Restrictions.
3. **MT_BuildAxialMesh.x: A Mesh Tool to Extrude 2D meshes to 3D meshes**

The focus of this section is the MT_BuildAxialMesh code which has both command line input and a control input file. Table 3 shows the command line input options that are required for this mesh tool to operate. Table 4 shows the description and an example of the control input for this mesh tool where the NTmesh recognized input for this file is *.axial or “control.axial.”

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The control input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

**Example:** `./MT_BuildAxialMesh.x ATR.ascii ATR_1.0.ascii`

<table>
<thead>
<tr>
<th>Input Keyword</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESH_FILE &lt;input mesh file name&gt;</td>
<td>&lt;input mesh file name&gt;</td>
<td>Set the input mesh file name</td>
</tr>
<tr>
<td>BLOCK_ALIAS &lt;mesh region name&gt; &lt;new region name&gt;</td>
<td>Rename mesh regions</td>
<td></td>
</tr>
<tr>
<td>AXIAL_MESH &lt;lower Z&gt; &lt;Upper Z&gt; &lt;Meshes&gt; &lt;Block Name&gt;</td>
<td>Define the axial extrusion</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```
MESH_FILE unstructured.ascii
BLOCK_ALIAS REGION_000000001 FUEL
BLOCK_ALIAS REGION_000000002 MODERATOR
BLOCK_ALIAS REGION_000000003 CLADDING
AXIAL_MESH 0.0 2.0 1 AA
AXIAL_MESH 2.0 8.0 2 AB
AXIAL_MESH 16.0 23.2343 2 AC
AXIAL_MESH 23.2343 25.0 1 AD
AXIAL_MESH 8.0 16.0 2 AE
```

As can be seen, there are only three control inputs for this mesh tool one of which is naming the mesh input file. While we could have done this as another command line input, we kept it this way to have consistent coding with the radial lattice program discussed later in this report. This approach allows one to combine multiple mesh tools together without having to string together numerous command line input options. It is important to note that this mesh can be two-dimensional or a three-dimensional slice which has clearly defined upper and lower axial planes.

Another key input option in this control file is the ability to rename the region names. In both CUBIT and the exodus format, the various geometrical regions are given numbers rather than names and thus the user must track an additional index of numbers to cross reference the more
logical use of named regions and compositions. Using this mesh tool, one can replace the numbers generated by the mesh tool with names for downstream usage.

The final control input option is the axial mesh specification. The first two values are obviously the region size specification where the third one is the number of actual meshes to place in each region. The last input on this line is only two letters long and serves as the addendum to any of the existing blocks in the mesh. In the example shown, the regions in the domain that are renamed to “fuel” will upon the extrusion exist as FUELAA, FUELAB, FUELAC, etc… It is important to note that the region names in PROTEUS are limited to 16 characters and thus the alias operation allowed by BLOCK_ALIASES is limited to at most 14 characters. Figure 2 shows an example extrusion of a 2D pin-cell mesh where the region coloring is not necessarily meaningful as the region ordering is not necessarily preserved from the two-dimensional mesh.

![Figure 2. Example MT_BuildAxialMesh.x Extrusion of a 2D Pin-cell Mesh.](image-url)
4. MT_ChangeFEbasis.x: A Mesh Tool to Alter the Basis Order of a Mesh

The focus of this section is the MT_ChangeFEbasis.x code which only has command line input. Table 5 shows the command line input options that are required for this mesh tool to operate. Unlike the previous mesh tools, there is an optional command line option which we indicate with the brackets \[.\] Figure 3 shows an example of how the number of mesh vertices is altered by this mesh tool.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>New FE Order</td>
<td>The new order of the FE basis functions</td>
</tr>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
<tr>
<td>[Element Type]</td>
<td>The type of element to switch to: null, serendipity, lagrange, gausslobato, nonconform</td>
</tr>
<tr>
<td>[Mesh Check]</td>
<td>Optional check to verify the mesh transformation</td>
</tr>
</tbody>
</table>

**Example**: 
```
./MT_ChangeFEbasis.x 2 input.ascii out.pntmesh NULL yes
```

![Figure 3. Example MT_ChangeFEbasis.x Change from a Linear (left) to Quadratic (right) Mesh.](image)

From Figure 3, it appears as though the number of elements has increased dramatically, however, in this case VISIT plots all higher order elements as if they were made of more lower order elements and thus the additional elements are an indication that more vertices are present than those necessary just to define the original element. In our example, we use the mesh tool to
change from a linear mesh to a quadratic finite element mesh. For this problem, the quadratic mesh has 38979 vertices while in the linear mesh only had 10946 vertices noting that the number of actual elements is the same in both meshes.

The second and third command line options are standard mesh input and output file definitions. The fourth command line allows one to force all elements in the mesh to be of a certain type. This is a particular issue with the CUBIT mesh conversion where some elements in CUBIT are stored as Lagrangian while others are stored as serendipity. The NULL command is the default option which does not change the type of element. All other selected options will cause the outputted mesh to be fully converted (via consistent interpolation) to the specified element type. The fifth command line input option can be rather important as it performs a check of the incoming and outgoing meshes to ensure that the volume of the two meshes does not change. It also can be used to check for twisted or poorly defined elements before proceeding to the PROTEUS code.

As a final note, the element type and order conversion was promoted to a file input option called *.SWITCHTYPE to allow it to be an inline operation performed on any given mesh. Table 6 shows the file format keyword input options and an example input. In this case, the output mesh file name is controlled by the command line options of the utility code. Consistent with all of the other mesh file formats, this file format can be called as a recursive input option of the input files.

<table>
<thead>
<tr>
<th>Input Keyword</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT_MESH</td>
<td>&lt;the input mesh&gt;</td>
<td>Specify the input mesh to load that is to be converted</td>
</tr>
<tr>
<td>TARGET_TYPE</td>
<td>&lt;element type&gt; [order]</td>
<td>Specify the new element type</td>
</tr>
</tbody>
</table>

**Example**

```
INPUT_MESH ./my.cubit.ascii
TARGET_TYPE SERENDIPITY 1
! TARGET_TYPE LAGRANGE 0 ! Will retain the existing mesh basis order
! TARGET_TYPE GAUSSLOBATTO 2
! TARGET_TYPE NONCONFORM 1
```
5. **MT_DeformMesh.x: A Mesh Tool to Deform 3D Meshes**

The focus of this section is the MT_DeformMesh.x tool which deforms any 3D PROTEUS-formatted finite element mesh according to user-specified strain (expansion coefficient) functions. The utility requires an input file that specifies the original mesh, strain functions, and other optional data. The utility only moves vertices and does not change the number of vertices, number of elements, or connectivity data. There is no possibility of vertex overlap or formation of inverted elements due to the limited deformation functions permitted.

![Image of MT_DeformMesh.x deformation](image)

**Figure 4. Example MT_DeformMesh.x Deformation Before and After Meshes.**

The usage and command line options are given in Table 9. A description of the required input file is given in Table 10. A pinned coordinate $(x_c, y_c, z_c)$ is calculated by default (or overridden by user input) which defines an origin around which the deformation occurs. The pinned coordinate does not need to coincide with an actual vertex.

The utility calculates the displacement of each vertex by calculating the X-, Y-, and Z-strain function values (expansion coefficients) at the vertex location. The X-strain and Y-strain functions are permitted to vary piecewise linearly with Z. The user provides a set of Z-values and corresponding strains to define these piecewise linear functions. It should be noted *all points within the X-Y plane at a given Z-elevation have the same X and Y strain functions*, so all assemblies move together (outward or inward). The strain in Z is permitted to be constant (independent of X, Y, Z).

<table>
<thead>
<tr>
<th><strong>Table 7. Command Line Options for the MT_DeformMesh.x Mesh Tool</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command Line Option</strong></td>
</tr>
<tr>
<td>deform.inp</td>
</tr>
<tr>
<td>out.ascii</td>
</tr>
<tr>
<td>out.vtk</td>
</tr>
</tbody>
</table>

**Example:** `./MT_DeformMesh.x deform.inp out.ascii [out.vtk]`
Table 8. Control Input File Format for MT_MeshDeform.x

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_MESHFILE</td>
<td>Path to file</td>
<td>Input mesh</td>
</tr>
<tr>
<td>VOLUME_FILE (Optional)</td>
<td>Path to file</td>
<td>Output file containing volume ratios of original and deformed mesh. Default is deformed_volumes.out.</td>
</tr>
<tr>
<td>PINNED_COORD_X (Optional)</td>
<td>Real value $x_c$</td>
<td>X-coordinate value of the pinned coordinate location. Default value is the average of the global maximum and minimum X-coordinates, $x_c = (x_{\text{min}} + x_{\text{max}})/2$.</td>
</tr>
<tr>
<td>PINNED_COORD_Y (Optional)</td>
<td>Real value $y_c$</td>
<td>Y-coordinate value of the pinned coordinate location. Default value is the average of the global maximum and minimum Y-coordinates, $y_c = (y_{\text{min}} + y_{\text{max}})/2$.</td>
</tr>
<tr>
<td>PINNED_COORD_Z (Optional)</td>
<td>Real value $z_c$</td>
<td>Z-coordinate value of the pinned coordinate location. Default value is the global minimum Z-coordinate, $z_c = z_{\text{min}}$.</td>
</tr>
<tr>
<td>ZSTRAIN</td>
<td>Real value $\varepsilon_z$</td>
<td>Constant strain in the z-direction. Ex: ZSTRAIN $= 0.02$ indicates 2% expansion in the Z-direction from the pinned coordinate. Assign 0 for no expansion in z-direction. Ex: ZSTRAIN $= -0.02$ indicates 2% contraction.</td>
</tr>
<tr>
<td>Z_ARRAY</td>
<td>Real values $z_1, z_2, ..., z_N$</td>
<td>Space delimited list of Z-coordinates that spans the entire range of the mesh. It is recommended that the original mesh has axial planes at these Z-coordinates to avoid unexpected results.</td>
</tr>
<tr>
<td>XSTRAIN_ARRAY</td>
<td>Real values $\varepsilon_{x1}, \varepsilon_{x2}, ..., \varepsilon_{xN}$</td>
<td>Space delimited list of strains in the X-direction corresponding to each entry in Z_ARRAY. Strains are linearly interpolated between these values.</td>
</tr>
<tr>
<td>YSTRAIN_ARRAY</td>
<td>Real values $\varepsilon_{y1}, \varepsilon_{y2}, ..., \varepsilon_{yN}$</td>
<td>Space delimited list of strains in the Y-direction corresponding to each entry in Z_ARRAY. Strains are linearly interpolated between these values.</td>
</tr>
<tr>
<td>SKIPSOLVE (Optional)</td>
<td>YES/NO</td>
<td>YES tells the utility to read in the mesh and input file, print the mesh extents including default pinned coordinate, and quit. NO is the default for normal operation.</td>
</tr>
</tbody>
</table>

Example:

```
SOURCE_MESHFILE  ../base.ascii
VOLUME_FILE      deformed_volumes.out
! PINNED_COORD_X  center of bottom plane
! PINNED_COORD_Y  center of bottom plane
! PINNED_COORD_Z  center of bottom plane
ZSTRAIN        0.01
Z_ARRAY        0.0  110.54  169.63  194.95  200.0  280.36  330.61  345.68
XSTRAIN_ARRAY  0.0 -6.81E-02 -1.36E-01 -6.81E-02 -3.40E-02  3.40E-01  5.45E-01  6.81E-01
YSTRAIN_ARRAY  0.0 -6.81E-02 -1.36E-01 -6.81E-02 -3.40E-02  3.40E-01  5.45E-01  6.81E-01
```
The utility produces output including the deformed mesh, vtk file (if requested), and material assignment file entries with the required density change factor for convenience. Additionally, an output file listing the block name, initial volume, deformed volume, and volume ratio is printed for convenience. If the same region name was used for multiple areas in system, it is possible that the volumes of all regions with the same name will be conglomerated and only one total volume will be reported in this report file. However, these convenience prints can be ignored if using the deformed mesh with PROTEUS-SN version 3.1.0+ in which material adjustments are handled automatically simply by using the new deformed mesh simulation option.

For the advanced user, some theory is included here. A vertex \( v = (x, y, z)^T \) is moved by a displacement vector \( \vec{u}(x, y, z) \) to yield a new vertex location \( v' = (x', y', z')^T \):

\[
\begin{pmatrix}
x' \\
y' \\
z'
\end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} u_x(x, y, z) \\ u_y(x, y, z) \\ u_z(x, y, z) \end{pmatrix}
\]

(1)

The X, Y, and Z components of the displacement vector are proportional to the strain \( \varepsilon \) in each dimension which describe the stretching or contraction at that coordinate. The pinned coordinate \( \vec{v}_c = (x_c, y_c, z_c)^T \) in the system specifies the point which stays stationary (new origin for stretching/compaction). The displacement in direction \( k \) is calculated by multiplying the strain function in direction \( k \) by the distance from the pinned coordinate,

\[
\begin{align*}
    u_x(x, y, z) &= (x - x_c) \varepsilon_x(x, y, z) \\
    u_y(x, y, z) &= (y - y_c) \varepsilon_y(x, y, z) \\
    u_z(x, y, z) &= (z - z_c) \varepsilon_z(x, y, z)
\end{align*}
\]

(2)

The utility permits piecewise linear strain functions in X and Y and constant strain functions in Z. The X- and Y-strain functions are only dependent on Z. The following equations define the 3 strain function between \( z_1 \) and \( z_2 \) directly from quantities provided in the user input file:

\[
\begin{align*}
    \varepsilon_x(x, y, z) &= \frac{(\varepsilon_{12}^x - \varepsilon_{11}^x)}{(z_2 - z_1)} (z - z_1) + \varepsilon_{11}^x \\
    \varepsilon_y(x, y, z) &= \frac{(\varepsilon_{12}^y - \varepsilon_{11}^y)}{(z_2 - z_1)} (z - z_1) + \varepsilon_{11}^y \\
    \varepsilon_z(x, y, z) &= \varepsilon_z
\end{align*}
\]

(3)
6. **MT_Extrude2Dto3D.x: A Mesh Tool to Extrude 2D meshes to a 3D slice**

The focus of this section is the MT_Extrude2Dto3D.x code which only has command line input which is given in Table 9. Given that the command line input only consists of the input and output mesh file names, there is nothing really to discuss. This mesh tool is one of the simplest and only extrudes a 2D mesh into a 3D planer mesh. Figure 5 shows the transformation in the mesh performed by this mesh tool noting that the final axial dimension is always 1.0 cm thick where the lower axial plane is at 0 cm and the upper plane is at 1 cm. It is important to note that linear hexahedrons in VISIT are plotted as a series of triangles (right picture) when using the VTK format while the quadrilaterals are plotted as the correct boxes (left picture) in two-dimensions. This can be confusing, but note that when we changed the basis order, the higher order quadrilaterals are also plotted using triangles in VISIT thus giving the same appearance. There is no difference in the visualization of linear and quadratic hexahedrons in VISIT when using the VTK input.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

**Example:** ./MT_Extrude2Dto3D.x input.ascii out.pntmesh

![Figure 5. Example MT_Extrude2Dto3D.x Change from a 2D (left) to a 3D (right) Mesh.](image)
7. MT_FixCUBITHex27Issue.x: A Mesh Tool to Fix CUBIT 27 Node Hexahedrons

The focus of this section is the MT_FixCUBITHex27Issue.x code which only has command line input that is given in Table 10. Much like the extrusion mesh tool, the command line input only consists of the input and output mesh file names and there is nothing really to discuss.

This mesh tool is very simple and targets a problem in the CUBIT software where curvilinear hexahedron 27 (hex27) elements are generated with an invalid set of points. The PROTEUS code will detect these elements as having bad shapes and thus will refuse to run if they are present. The problem with these elements in CUBIT is that the final 7 points added to the hexahedron 20 element (Hex20) are computed using a barycenter algorithm. In the case of the 27th vertex which is supposed to lie at the center of the element, for a significantly curved element (cladding), the barycenter can lie outside of the element. Similarly, for the other 6 vertices, the points are placed on the local barycenter of the surface which is not on the actual surface. This mesh tool corrects these mistakes by interpolating them from the hex20 element such that it exactly preserves the volume of the hex20 meshes.

An example verification test problem is included in the regression test which shows how invalid hex27 meshes are corrected for which we show a single element alteration in Figure 6.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

**Example:** ./MT_FixCUBITHex27Issue.x input.ascii out.pntmesh

![Figure 6. Example MT_FixCUBITHex27Issue.x Correction of a Bad Hex 27 Element (left).](image)

It is important to note that this was done using the HDF5 file as VISIT does not support the hex27 element in VTK. We encourage users to use the MT_ChangeFEbasis.x mesh tool to check
any hexahedron 27 meshes coming out of CUBIT to determine if their mesh is affected and use the MT_FixCUBITHex27Issue.x mesh tool to correct them as necessary.
8. MT_IdentifyVertices.x: A Mesh Tool to Identify Vertices in a Mesh

The focus of this section is the MT_IdentifyVertices.x code which is used to identify a set of vertices within a box of a given mesh. Table 11 shows the command line input options that are required for this mesh tool to operate along with the example we display results for. Figure 7 shows the ATR mesh and a selected region of the mesh for which we want to know the vertices of.

Table 11. Command Line Options for the MT_IdentifyVertices.x Mesh Tool

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>X minimum</td>
<td>The lower X coordinate of the search box</td>
</tr>
<tr>
<td>X maximum</td>
<td>The upper X coordinate of the search box</td>
</tr>
<tr>
<td>Y minimum</td>
<td>The lower Y coordinate of the search box</td>
</tr>
<tr>
<td>Y maximum</td>
<td>The upper Y coordinate of the search box</td>
</tr>
<tr>
<td>Z minimum</td>
<td>The lower Z coordinate of the search box</td>
</tr>
<tr>
<td>Z maximum</td>
<td>The upper Z coordinate of the search box</td>
</tr>
</tbody>
</table>

Example: `./MT_IdentifyVertices.x ATR.ascii -14.6 -13.1 -4.8 -3.0 0.0 1.0`

Figure 7. Example MT_IdentifyVertices.x Box Selection of a Pin in the ATR Mesh (right).

Much like the other mesh tools, the command line input for this mesh tool is straightforward as it consists of the input mesh and the box boundaries that surround the target region of interest. Figure 8 shows the output in this case which lists the box of interest along with all vertices that lie within the box. Note how the vertex numbering is not sequential, but that there are significant
jumps in the numbering. The cited vertex number is the desired output as this vertex numbering is preserved in both serial and parallel implementations of PROTEUS. The most common use of this mesh tool is to identify a vertex to assign a point source to and with complicated meshes like the ATR [6], this can be considerably difficult. In this case, the VTK export of the mesh was used followed by VISIT to identify a box which surrounded the pin of interest.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex 678</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.4.59293020E+00</td>
</tr>
<tr>
<td>Vertex 678</td>
<td>within box at position -1.362443940E+01</td>
<td>-1.4.69536890E+00</td>
</tr>
<tr>
<td>Vertex 679</td>
<td>within box at position -1.361945600E+01</td>
<td>-1.4.64973025E+00</td>
</tr>
<tr>
<td>Vertex 679</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.4.54735193E+00</td>
</tr>
<tr>
<td>Vertex 750</td>
<td>within box at position -1.359253760E+00</td>
<td>-1.4.37131588E+00</td>
</tr>
<tr>
<td>Vertex 750</td>
<td>within box at position -1.358753000E+00</td>
<td>-1.4.32065858E+00</td>
</tr>
<tr>
<td>Vertex 751</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.3.22135979E+00</td>
</tr>
<tr>
<td>Vertex 822</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.3.94634278E+00</td>
</tr>
<tr>
<td>Vertex 822</td>
<td>within box at position -1.356063730E+01</td>
<td>-1.4.07293300E+00</td>
</tr>
<tr>
<td>Vertex 822</td>
<td>within box at position -1.355565310E+01</td>
<td>-1.3.99666957E+00</td>
</tr>
<tr>
<td>Vertex 823</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.3.89541742E+00</td>
</tr>
<tr>
<td>Vertex 894</td>
<td>within box at position -1.352837860E+00</td>
<td>-1.3.72330095E+00</td>
</tr>
<tr>
<td>Vertex 894</td>
<td>within box at position -1.352354600E+01</td>
<td>-1.3.67267904E+00</td>
</tr>
<tr>
<td>Vertex 895</td>
<td>within box at position -1.446940000E+00</td>
<td>-1.3.56952090E+00</td>
</tr>
<tr>
<td>Vertex 966</td>
<td>within box at position -1.349684130E+01</td>
<td>-1.3.39933051E+00</td>
</tr>
<tr>
<td>Vertex 966</td>
<td>within box at position -1.349185750E+01</td>
<td>-1.3.34871214E+00</td>
</tr>
<tr>
<td>Vertex 967</td>
<td>within box at position -1.349684130E+01</td>
<td>-1.3.24366642E+00</td>
</tr>
<tr>
<td>Vertex 1038</td>
<td>within box at position -1.346494530E+01</td>
<td>-1.3.07538212E+00</td>
</tr>
<tr>
<td>Vertex 3369</td>
<td>within box at position -1.447180810E+01</td>
<td>-1.3.63619338E+00</td>
</tr>
<tr>
<td>Vertex 3372</td>
<td>within box at position -1.352999097E+01</td>
<td>-1.3.73305959E+00</td>
</tr>
<tr>
<td>Vertex 3390</td>
<td>within box at position -1.446681190E+01</td>
<td>-1.4.50933931E+00</td>
</tr>
<tr>
<td>Vertex 3391</td>
<td>within box at position -1.447517950E+01</td>
<td>-1.4.31033461E+00</td>
</tr>
<tr>
<td>Vertex 3393</td>
<td>within box at position -1.447324960E+01</td>
<td>-1.4.20692040E+00</td>
</tr>
<tr>
<td>Vertex 3394</td>
<td>within box at position -1.359790990E+01</td>
<td>-1.4.40903939E+00</td>
</tr>
<tr>
<td>Vertex 3412</td>
<td>within box at position -1.446687050E+01</td>
<td>-1.4.18334310E+00</td>
</tr>
<tr>
<td>Vertex 3413</td>
<td>within box at position -1.447135220E+01</td>
<td>-1.4.98439971E+00</td>
</tr>
<tr>
<td>Vertex 3415</td>
<td>within box at position -1.358240910E+01</td>
<td>-1.4.28299280E+00</td>
</tr>
<tr>
<td>Vertex 3416</td>
<td>within box at position -1.356581940E+01</td>
<td>-1.4.09504620E+00</td>
</tr>
<tr>
<td>Vertex 3843</td>
<td>within box at position -1.446692480E+01</td>
<td>-1.3.85739743E+00</td>
</tr>
<tr>
<td>Vertex 3843</td>
<td>within box at position -1.447126000E+01</td>
<td>-1.3.66851044E+00</td>
</tr>
<tr>
<td>Vertex 3843</td>
<td>within box at position -1.355056740E+01</td>
<td>-1.3.95890505E+00</td>
</tr>
<tr>
<td>Vertex 3848</td>
<td>within box at position -1.353372780E+01</td>
<td>-1.3.76107922E+00</td>
</tr>
<tr>
<td>Vertex 3845</td>
<td>within box at position -1.446679100E+01</td>
<td>-1.3.53149978E+00</td>
</tr>
<tr>
<td>Vertex 3845</td>
<td>within box at position -1.447090900E+01</td>
<td>-1.3.32365309E+00</td>
</tr>
<tr>
<td>Vertex 3845</td>
<td>within box at position -1.351872460E+01</td>
<td>-1.3.63490709E+00</td>
</tr>
<tr>
<td>Vertex 3846</td>
<td>within box at position -1.350164420E+01</td>
<td>-1.3.43713564E+00</td>
</tr>
<tr>
<td>Vertex 3846</td>
<td>within box at position -1.350572170E+01</td>
<td>-1.3.20568490E+00</td>
</tr>
<tr>
<td>Vertex 3847</td>
<td>within box at position -1.447067670E+01</td>
<td>-1.3.00685425E+00</td>
</tr>
<tr>
<td>Vertex 3848</td>
<td>within box at position -1.348680700E+01</td>
<td>-1.3.31093314E+00</td>
</tr>
<tr>
<td>Vertex 3848</td>
<td>within box at position -1.346956220E+01</td>
<td>-1.3.11231367E+00</td>
</tr>
</tbody>
</table>

Figure 8. Example MT_IdentifyVertices.x Output.
9. MT_MeshRestrictForBGP.x: A Mesh Tool to Restrict Element Block Sizes

The focus of this section is the MT_MeshRestrictForBGP.x code which has a similar impact as the MT_ApplyACRtoMesh.x. Table 2 shows the command line input options that are required for this mesh tool to operate along with an example. The first two options are again the input and output mesh file names which do not need to be discussed. The third input is optional where the default is 10. The origin of this mesh tool is the Blue Gene/P architecture which had considerable memory constraints. The purpose of this tool is to restrict the memory size of the data stored in each element block. The memory size constraint is placed upon the element connectivity list that is stored in the mesh data structure and the outcome of the code is a mesh where any large blocks of element data are broken into smaller blocks. Unlike the MT_ApplyACRtoMesh.x tool, this code constrains the memory size rather than the volume of the block and thus this tool only applies to really large meshes. Because the base ATR mesh is a two-dimensional mesh, this code makes no change to the block setup even when using a 1 MB input constraint.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
<tr>
<td>[memory size limit]</td>
<td>The memory size limit (in MB) to apply where the default is 10 MB.</td>
</tr>
</tbody>
</table>

**Example:** `./MT_MeshRestrictForBGP.x ATR.ascii ATR.pntmesh 20`
10. MT_MeshToBGPmesh.x: A Mesh Tool to Create a Pre-Partitioned Mesh

The focus of this section is the MT_MeshToBGPmesh.x code which builds a pre-partitioned mesh for use on the Blue Gene/P computer. In most cases there is no need to apply this mesh tool although it can improve performance and reduce time in the mesh load process. Table 13 shows the command line input options that are required for this mesh tool to operate noting that two of them are optional.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file (should end with a .bgpmesh)</td>
</tr>
<tr>
<td>Spatial decomposition</td>
<td>The number of pieces to decompose the spatial mesh into</td>
</tr>
<tr>
<td>Ghosting type</td>
<td>Use one ring of “VERTEX” or “ELEMENT” around mesh</td>
</tr>
<tr>
<td>[Domains per process]</td>
<td>The number of domains assigned to each process</td>
</tr>
<tr>
<td>[vtk output file]</td>
<td>An option to export a VTK file showing the decomposition</td>
</tr>
</tbody>
</table>

Example: ./MT_MeshToBGPmesh.x ATR.ascii ATR.bgpmesh 8 VERTEX 1 ATR.vtk

As mentioned, the main point of this mesh tool is to pre-partition the mesh which was needed on the Blue Gene/P architecture as the memory resources were constrained and the time required to load and partition large meshes could be considerable. The first two command line inputs are similar to those of other mesh tools, but one should note that the .bgpmesh extension should be used on the output file. The third command line input is by far the most important as it specifies the number of pieces that the mesh is to be decomposed into. This must match the settings used in PROTEUS. The ghosting type is specific to the actual solution algorithm being implemented. For some solution methodologies one must obtain a single ring of ghosted vertex data while other methodologies require a single ring of element data. This input must be specified and we note that PROTEUS-SN requires the VERTEX setting.

The fifth command line input is optional and should always be set to 1. The basic intent of this input is to give more than one piece of the mesh to each process and originated from our research into the MOC solver. In the MOC case, it was necessary to assign several pieces of the mesh to each process to get a better load balance which one should intuitively understand will yield more communication. However, since the MOC algorithm is compute bound rather than communication bound, the assignment of more sub-domains actually improves the load balancing and thus yielded better parallel scaling.

The sixth command line input can be rather invaluable. Unlike the VTK files that are produced by a simple mesh conversion (aka MT_MeshToMesh.x), the parallel partitioning information is
stored on the VTK file produced by the MT_MeshtoBGmesh.x tool. Figure 9 shows the ATR.vtk file that results from the example input applied to the ATR mesh. As can be seen, there are 8 distinct colors assigned to the mesh. The left hand picture shows the parallel vertex assignment and the interpolation occurs because of ghosted elements between two adjacent processors. The right hand picture is more useful as it shows the element assignment to each processor. In the element assignment case, there is no interpolation along shared element borders but distinct boundaries.

![Figure 9. Example MT_MeshtoBGmesh.x Vertex Rank (left) and Element Rank (right).](image)
11. MT_MeshToHDF5.x: A Mesh Tool to Create a VISIT readable HDF5 File

The focus of this section is the MT_MeshToHDF5.x code which has command line input and an optional input file to assign specific colors to each region. Table 14 shows the command line input options that are required for this mesh tool to operate where the first two are again the mesh.

Table 14. Command Line Options for the MT_MeshToHDF5.x Mesh Tool

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file which should end with .h5</td>
</tr>
<tr>
<td>[region to color map file]</td>
<td>A file which assigns colors to each region name</td>
</tr>
</tbody>
</table>

Example: ./MT_MeshToHDF5.x ATR_ascii ATR.h5 colormap.inp

Table 15. Example Color Map for the ATR Mesh

| REGION_000000111       | 1  |
| REGION_000000112       | 1  |
| REGION_000000113       | 1  |
| REGION_000000114       | 1  |
| REGION_000000115       | 1  |
| REGION_000000116       | 1  |
| REGION_000000117       | 1  |
| REGION_000000118       | 1  |
| REGION_000000119       | 1  |
| REGION_000000120       | 1  |
| REGION_000000121       | 1  |
| REGION_000000122       | 1  |
| REGION_000000123       | 1  |
| REGION_000000124       | 1  |
| REGION_000000125       | 1  |
| REGION_000000126       | 1  |
| REGION_000000127       | 1  |
| REGION_000000128       | 1  |
| REGION_000000129       | 1  |
| REGION_000000130       | 1  |
| REGION_000000131       | 41 |
| REGION_000000133       | 40 |
| REGION_000000134       | 22 |
| REGION_000000135       | 23 |
| REGION_000000136       | 40 |
| REGION_000000137       | 30 |
| REGION_000000138       | 38 |
| REGION_000000139       | 38 |
| REGION_000000140       | 1  |
| REGION_000000141       | 7  |
| REGION_000000142       | 7  |
| REGION_000000143       | 20 |
| REGION_000000144       | 5  |
| REGION_000000145       | 31 |
| REGION_000000146       | 38 |
| REGION_000000147       | 22 |
| REGION_000000148       | 23 |
| REGION_000000149       | 40 |
| REGION_000000150       | 30 |
| REGION_000000151       | 38 |
| REGION_000000152       | 38 |
| REGION_000000153       | 1  |
| REGION_000000154       | 7  |
| REGION_000000155       | 7  |
| REGION_000000156       | 7  |
| REGION_000000157       | 7  |
| REGION_000000158       | 7  |
| REGION_000000159       | 7  |
| REGION_000000160       | 7  |
| REGION_000000161       | 7  |
| REGION_000000162       | 7  |
| REGION_000000163       | 7  |
| REGION_000000164       | 7  |
| REGION_000000165       | 7  |
| REGION_000000166       | 7  |

The colormap file is a structured input file and the HDF5 file that results can be used to look at the mesh and block numbering. An output example is deferred until later in this report.
12. MT_MeshToMesh.x: A Mesh Tool to Convert Between Mesh Formats

The focus of this section is the MT_MeshToMesh code which only has the command line input shown in Table 16. This particular mesh tool is quite useful for quickly converting between different mesh formats and testing out GRID, NEMESH, and UFMESH inputs.

Table 16. Command Line Options for the MT_MeshToMesh.x Mesh Tool

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

Example: ./MT_MeshToMesh.x input.ascii out.pntmesh
13. MT_MeshToVTK.x: A Mesh Tool to Create a VISIT Readable VTK File

The MT_MeshToVTK code is similar to the MT_MeshToHDF5 in that its purpose is to create a visualization of the mesh only. Table 17 shows the command line input options that are required for this mesh tool to operate. The color table description was given earlier in Table 15 and Figure 10 shows the impact of using the coloring upon the ATR mesh [6].

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file which should end with .vtk</td>
</tr>
<tr>
<td>[region to color map file]</td>
<td>A file which assigns colors to each region name</td>
</tr>
</tbody>
</table>

**Example:** ./MT_MeshToVTK.x ATR.ascii ATR.vtk colormap.inp

![Figure 10. Example MT_MeshToVTK.x Coloring of the ATR Mesh.](image)
14. **MT_RadialLattice.x: A Mesh Tool to Merge Meshes on a Radial Grid**

The MT_RadialLattice.x code is used to merge existing two-dimensional or three-dimensional meshes together on a structured grid and has both command line and a control input. Table 18 shows the command line input options for this mesh tool while Table 19 shows the control input for the radial lattice code noting that *.merge is the NTmesh recognized extension. For a thorough test of the various mesh tools and their integration, we utilize the merge routine to completely build a homogeneous reactor problem.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Input File</td>
<td>The control input for the mesh tool</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
<tr>
<td>[use P or A names]</td>
<td>A switch to preface the region name with a P(in) or A(ssembly). A is used by default.</td>
</tr>
<tr>
<td>[output vtk]</td>
<td>An option to build a VTK file</td>
</tr>
<tr>
<td>[debug]</td>
<td>Yes/No flag to generate debug output for the merging process</td>
</tr>
</tbody>
</table>

**Example:** `./MT_RadialLattice.x merge.inp merge.ascii P merge.vtk no`

The command line input is not much different from the other mesh tools. The optional P or A names will be discussed shortly. The debug input should not be used unless the user has intimate knowledge of the mesh merging algorithm. The output vtk eliminates the need to run this mesh tool (or any other) when creating a VTK file to visualize and check the mesh merging process. One easy way to do this is to look at the “NumberOfSurfaceTags” vector exported on the mesh. Any distinct mesh that is created by a user must have valid boundary conditions before merging even though they are not to be included in the final geometry. When the merging process fails, the mesh will appear to have internal boundary conditions that are not valid for neutronics work.

From Table 19 we can see that the radial lattice code has five keyword input lines. The “radial_lattice” keyword input is by far the most important and should appear first. This keyword input allows the user to specify the type of geometrical grid of which Cartesian and hexagonal are supported. Given that these are structured grids the user must also provide a pitch where in the Cartesian case, the x and y pitch are assumed identical in the merge program. One can use the MT_MeshModify.x program to alter the y location of meshes that do not obey this mesh merging rule. The last two numbers on this line specify the size of the grid data map and must be consistent with the gridline_data keyword input to work. In the example shown, we are going to build a hexagonal grid of meshes with a pitch of 8.0 cm and grid size of 13 x 13.

The next keyword input in Table 19 is geometry_data which is used to specify specific meshes to load. The first character input is an alias naming to make the gridline_data input easier to read.
In the example, we have four geometry_data input lines each of which load four such meshes and alias them as A1 through A4. It is important to note that in each case we load a MT_BuildAxialMesh.x control input file described earlier. The apply_check keyword can be used to enforce that each user defined mesh in the merge process is checked for holes along with a check on the final mesh. Table 20 shows the hex01.axial file referenced in the Table 19 example and Table 21 shows the hex01.ufmesh file referenced in Table 20. As one can see, the UFmesh is rather simplistic and one can understand how far more complicated geometries and meshes can be built.

To finalize the description of the merge input, we look at the gridline_data in Table 19 and see that we constructed a hexagonal core. We are not technically required to use the MT_RadialLattice.x to construct the final mesh as any of the mesh tools can be used so long as the input in Table 19 ends with a .merge extension. Figure 11 shows the resulting geometry and mesh that this example input builds. It is important to note that the pictures were made using CUBIT and a VTK file from MT_MeshToMesh.x. If one uses the actual MT_RadialLattice.x code, then the block region names can be altered. This is particularly important if one wants to assign different compositions to each assembly without having to build separate meshes for each assembly.

To use the auto-renaming scheme, one must simply provide the assignment files which was not done with the example shown in Table 19. In this case, the mapping between regions and compositions will be maintained, but when the copy process is done to duplicate the given assembly mesh in the lattice, the region names and composition names are updated with a Axxyy preface where “xx” is the radial X position and “yy” is the radial Y position. If the user specifies the optional P on the command line input, this will be Pxxxy corresponding to a specific pin position. If one uses the merge operation to build each assembly with pin names Pxxxy and then to merge assemblies, the final names will be AxxyyPxxxy and the compositions appearing in the final assignment file will also have the same information. If desired, one should use the MT_BuildAxialMesh code after creating each assembly mesh as it will append the two letters to the end to create names PxxxyAA, PxxxyAB, … consistent with the example shown in that section. Combining the assembly meshes would then lead to names AxxyyPxxxyyAA, AxxyyPxxxyyAB, … which uses the maximum size of the character string at this time.

After extensive usage, there was a considerable desire to add a general purpose mesh merge option. This is primarily because the preceding lattice type input requires additional coordinate calculations be done to determine the proper centroid of each mesh. Table 22 gives the input specification for the .mergexyz input and shows an example input file. There is no point in showing an example output as the input is just an equivalent way of writing the input shown in Table 19. One important aspect is that this input file format allows one to merge 3D meshes whereas the original lattice input only allows 2D mesh merging. One important aspect to note is that there is no ability to apply a “pin” or “assembly” renaming scheme like that shown in Table 18 as there is no lattice index to be applied.
### Table 19. Control Input for the MT_RadialLattice.x Mesh Tool

<table>
<thead>
<tr>
<th>Input Keyword</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_CHECK</td>
<td>&lt;yes/no&gt;</td>
<td>Apply rigorous checking to every user mesh and merged mesh</td>
</tr>
<tr>
<td>OUTPUT_FILE</td>
<td>&lt;new assignment output&gt;</td>
<td>Specify the output file for writing assignment data</td>
</tr>
<tr>
<td>RADIAL_LATTICE</td>
<td>&lt;type&gt; &lt;pitch&gt; &lt;x size&gt; &lt;y size&gt; &lt;Type&gt; = &lt;hexagonal&gt; or &lt;Cartesian&gt;</td>
<td>Specify the type of grid, its pitch, and number of x and y dimensions</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>&lt;alias&gt; &lt;input mesh&gt; [input assignment]</td>
<td>Specify an input mesh and assignment file and assign them both an alias</td>
</tr>
<tr>
<td>GRIDLINE_DATA</td>
<td>&lt;y position&gt; &lt;x1&gt; &lt;x2&gt; ... &lt;xN&gt;</td>
<td>Define the axial extrusion</td>
</tr>
</tbody>
</table>

**Example**

RADIAL_LATTICE Hex 8.0 13 13  
GEOMETRY_DATA A1 ./hex01.axial  
GEOMETRY_DATA A2 ./hex02.axial  
GEOMETRY_DATA A3 ./hex03.axial  
GEOMETRY_DATA A4 ./hex04.axial  
GRIDLINE_DATA 13 0 A3 A3 A3 A3 A3 0 0 0 0 0 0 0 0  
GRIDLINE_DATA 11 A3 A3 A3 A3 A3 0 0 0 0 0 0 0  
GRIDLINE_DATA 10 A3 A3 A3 A3 A3 0 0 0 0 0 0  
GRIDLINE_DATA 9 A3 A3 A2 A2 A2 A2 A2 A2 A3 A3 0 0  
GRIDLINE_DATA 8 A3 A3 A2 A2 A1 A1 A1 A1 A2 A3 A3 0  
GRIDLINE_DATA 7 0 A3 A3 A4 A4 A4 A4 A4 A4 A4 A4 0  
GRIDLINE_DATA 6 0 A3 A3 A2 A2 A1 A1 A1 A1 A2 A3 A3 0  
GRIDLINE_DATA 5 0 0 A3 A3 A2 A2 A2 A2 A2 A2 A3 A3 0  
GRIDLINE_DATA 4 0 0 0 A3 A3 A2 A2 A2 A2 A2 A2 A3 A3  
GRIDLINE_DATA 3 0 0 0 0 A3 A3 A4 A2 A2 A2 A4 A3 A3  
GRIDLINE_DATA 2 0 0 0 0 0 A3 A3 A3 A3 A3 A3 A3 0  
GRIDLINE_DATA 1 0 0 0 0 0 0 0 A3 A3 A3 A3 A3 0

### Table 20. Control Input of the hex01.axial File for the MT_RadialLattice.x Example

| MESH_FILE hex01.ufmesh | AXIAL_MESH 0.0 20.0 3 A | AXIAL_MESH 20.0 40.0 3 B | AXIAL_MESH 40.0 60.0 3 C | AXIAL_MESH 60.0 70.0 2 D | AXIAL_MESH 70.0 80.0 2 E | AXIAL_MESH 80.0 90.0 2 F | AXIAL_MESH 90.0 100.0 2 G | AXIAL_MESH 100.0 110.0 2 H | AXIAL_MESH 110.0 130.0 3 I | AXIAL_MESH 130.0 150.0 3 J | AXIAL_MESH 150.0 170.0 3 K |
Table 21. UFmesh Input of the hex01.ufmesh File for the MT_RadialLattice.x Example

<table>
<thead>
<tr>
<th>GRID_TYPE HEXAGONAL 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRID_PITCH 8.0</td>
</tr>
<tr>
<td>MESH_ROTATE 30.0</td>
</tr>
<tr>
<td>ELEMENT_ORDER 1</td>
</tr>
<tr>
<td>PINCELL_REGIONS A1 INNERCORE</td>
</tr>
<tr>
<td>PINCELL_MESSES A1 2</td>
</tr>
<tr>
<td>GRID_MAP 1 A1</td>
</tr>
<tr>
<td>PINCELL AZIMUTHAL 2</td>
</tr>
</tbody>
</table>

Figure 11. Example MT_RadialLattice.x Mesh Demonstrating Complete Problem Creation.

Table 22. General Purpose Mesh Merge Control Input

<table>
<thead>
<tr>
<th>Input Keyword</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_CHECK</td>
<td>&lt;yes/no&gt;</td>
<td>Apply rigorous checking to every user mesh and merged mesh</td>
</tr>
<tr>
<td>OUTPUT_FILE</td>
<td>&lt;new assignment output&gt;</td>
<td>Specify the output file for writing assignment data</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>&lt;X&gt; &lt;Y&gt; &lt;Z&gt; &lt;mesh&gt; [input assignment]</td>
<td>Specify an input mesh and its X,Y,Z offsets along with optional assignment</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>OUTPUT_FILE</th>
<th>./erase.assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY_DATA</td>
<td>1.0 0.0 0.0 ./data/cart_1.nemesh ./data/cart_1.assignment</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>2.0 0.0 0.0 ./data/cart_2.nemesh ./data/cart_2.assignment</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>3.0 0.0 0.0 ./data/cart_3.nemesh ./data/cart_3.assignment</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>4.0 0.0 0.0 ./data/cart_4.nemesh ./data/cart_4.assignment</td>
</tr>
<tr>
<td>GEOMETRY_DATA</td>
<td>5.0 0.0 0.0 ./data/cart_5.nemesh ./data/cart_5.assignment</td>
</tr>
</tbody>
</table>
15. MT_Refine2Dmesh.x: A Mesh Tool to Automatically Refine a 2D Mesh

The MT_Refine2Dmesh code only applies to two-dimensional linear finite element meshes at this time. Its purpose is to make a uniform refinement in a given mesh. This mesh code has very basic command line inputs as shown in Table 23. Figure 12 shows an example refinement of the a 2D mesh where the right hand picture is the result of the refinement. As can be seen, the algorithm places a single point at the barycenter of each existing finite element and sub-divides the element into triangles. This approach quickly leads to poor aspect ratio elements. An alternative idea is to split each surface of the existing element into two pieces thereby breaking each triangle into four elements and each quadrilateral into four elements. Given the rare use of this mesh tool, this alternative scheme has not been implemented. Based upon the poor mesh quality that results, one should consider using the mesh tools discussed in this report or the commercial mesh generator to produce a more refined mesh.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
</tbody>
</table>

**Example:** `./MT_Refine2Dmesh.x input.ascii out.pntmesh`

Figure 12. Example MT_Refine2Dmesh.x Barycenter Refinement of a 2D Mesh.
16. MT_ModifyMesh.x: A Mesh Tool to Rotate and Translate a Mesh

The MT_ModifyMesh code can be used to rotate a given mesh around the coordinate axis and translate it. This mesh code has rather complicated command line input as shown in Table 24. From this table, one should understand that there are actually four different command line input options that are accepted. To understand them, one must understand that the actual rotation for every vertex \( \hat{r} \) in the mesh given as:

\[
\hat{r}' = \hat{R} \cdot \hat{r}
\]  

(1)

In this form, the matrix \( \hat{R} \) is a 3x3 matrix which is generic in its usage, but should be normalized to unity in practice. Looking at the first example, the usage only specifies the rotation angle which yields a rotation matrix of the form:

\[
\hat{R} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

(2)

where the angle \( \theta \) is computed from the entry in degrees.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file</td>
</tr>
<tr>
<td>Rotation Angle</td>
<td>The X-Y plane counter clockwise rotation to apply</td>
</tr>
<tr>
<td>X movement</td>
<td>The translation to apply in the X direction</td>
</tr>
<tr>
<td>Y movement</td>
<td>The translation to apply in the Y direction</td>
</tr>
<tr>
<td>Z movement</td>
<td>The translation to apply in the Z direction</td>
</tr>
<tr>
<td>( R_{i,j} ) component</td>
<td>The ((i,j)) matrix value of the rotation matrix ( \hat{R} )</td>
</tr>
</tbody>
</table>

**Option 1**: \(<\text{input mesh}> \ <\text{output mesh}> \ <\text{rotation}>\)

**Option 2**: \(<\text{input mesh}> \ <\text{output mesh}> \ <\text{X move}> \ <\text{Y move}> \ <\text{Z move}>\)

**Option 3**: \(<\text{input mesh}> \ <\text{output mesh}> \ <\text{rotation}> \ <\text{X move}> \ <\text{Y move}> \ <\text{Z move}>\)

**Option 4**: \(<\text{input mesh}> \ <\text{output mesh}> \ <\text{R11}> \ <\text{R12}> \ <\text{R13}> \ <\text{R21}> \ldots \ <\text{R33}>\)

**Example 1**: ./MT_ModifyMesh.x input.ascii out.pntmesh 30

**Example 2**: ./MT_ModifyMesh.x input.ascii out.pntmesh 2 2 2

**Example 3**: ./MT_ModifyMesh.x input.ascii out.pntmesh 30 2 2 2

**Example 4**: ./MT_ModifyMesh.x input.ascii out.pntmesh .4 .2 0 .1 .3 0 0 0 1.
In the second input option and associated example, the input only focuses on the translation of a given mesh. The third input option combines the first two and thus allows a rotation followed by a translation of a given mesh. The fourth input option only deals with rotation where the user can specify a complete rotation matrix. This input option actually allows the user to not only arbitrarily rotate a mesh, but also skew the mesh when the vectors are not normalized. One should be very careful when using this approach as the overall volume may not be preserved.
17. MT_DataToVTK.x: A Mesh Tool to Plot Solution data Via a VTK File

The MT_DataToVTK code is similar to the MT_MeshToVTK file in that its purpose is to create a visualization of the mesh. Table 25 shows the command line input options that are required for this mesh tool to operate. The color table description was given earlier in Table 15 and Figure 13 shows an example of loading a flux vector field and plotting it on a mesh.

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>The mesh input file</td>
</tr>
<tr>
<td>Data File</td>
<td>The data file that is to be plotted</td>
</tr>
<tr>
<td>Output File</td>
<td>The mesh output file which should end with .h5</td>
</tr>
<tr>
<td>[Element/Vertex]</td>
<td>(E)lement or (V)ertex Flag Indicating Field Type</td>
</tr>
<tr>
<td>[region to color map file]</td>
<td>A file which assigns colors to each region name</td>
</tr>
</tbody>
</table>

**Example:** ./MT_MeshToVTK.x bench08.ascii bench08.data bench08.vtk e color.inp

Figure 13. Example MT_DataToVTK.x Plot of an Element Vector Field.

The primary difference in this mesh tool is the “Data File” of which Figure 14 shows an example. This file should have no header comment lines and the first column must contain the
element or vertex index position to place the vector data. As can be seen, the ordering of the data is not important, only that the index does not go out of the bounds of the mesh (i.e. number of mesh vertices or mesh elements). The vector data will be labeled CellWiseXXXX for element data and VertexWiseXXXX for vertex data where the XXXX is the column index (i.e. 0001 to 0007 in the example). The mesh tool uses the PROTEUS free format reader and thus one can include comments at the end of each line, but it will detect any input line of data that does not have the same number of columns as the first line and flag it as an error.

| 9569 | 2.929E-3 | 3.915E-3 | 2.160E-3 | 1.003E-3 | 7.756E-4 | 2.345E-3 | 6.232E-3 |
| 9570 | 2.847E-3 | 3.764E-3 | 2.066E-3 | 9.601E-4 | 7.391E-4 | 2.204E-3 | 5.769E-3 |
| 9590 | 2.761E-3 | 3.669E-3 | 2.008E-3 | 9.339E-4 | 7.170E-4 | 2.118E-3 | 5.488E-3 |
| 9591 | 2.786E-3 | 3.637E-3 | 1.991E-3 | 9.281E-4 | 7.128E-4 | 2.109E-3 | 5.466E-3 |
| 9592 | 3.387E-3 | 5.224E-3 | 3.018E-3 | 1.411E-3 | 1.116E-3 | 3.772E-3 | 1.098E-2 |
| 9593 | 3.373E-3 | 5.047E-3 | 2.898E-3 | 1.353E-3 | 1.069E-3 | 3.568E-3 | 1.031E-2 |
| 9594 | 3.310E-3 | 4.833E-3 | 2.754E-3 | 1.293E-3 | 1.010E-3 | 3.315E-3 | 9.466E-3 |
| 10765 | 3.253E-3 | 4.608E-3 | 2.603E-3 | 1.212E-3 | 9.514E-4 | 3.069E-3 | 8.644E-3 |
| 10766 | 3.179E-3 | 4.369E-3 | 2.446E-3 | 1.137E-3 | 8.885E-4 | 2.802E-3 | 7.739E-3 |
| 10767 | 3.061E-3 | 4.141E-3 | 2.301E-3 | 1.106E-3 | 8.311E-4 | 2.568E-3 | 6.962E-3 |
| 10768 | 2.865E-3 | 3.350E-3 | 1.738E-3 | 8.028E-4 | 6.023E-4 | 1.630E-3 | 3.860E-3 |
| 10769 | 2.672E-3 | 3.186E-3 | 1.645E-3 | 7.614E-4 | 5.668E-4 | 1.496E-3 | 3.393E-3 |
| 411 | 2.583E-3 | 3.069E-3 | 1.582E-3 | 7.348E-4 | 5.439E-4 | 1.413E-3 | 3.129E-3 |
| 412 | 2.590E-3 | 3.021E-3 | 1.559E-3 | 7.255E-4 | 5.373E-4 | 1.400E-3 | 3.104E-3 |
| 427 | 3.802E-3 | 5.883E-3 | 3.415E-3 | 1.596E-3 | 1.270E-3 | 4.366E-3 | 1.285E-2 |
| 428 | 3.700E-3 | 5.489E-3 | 3.158E-3 | 1.473E-3 | 1.168E-3 | 3.946E-3 | 1.149E-2 |
| 429 | 3.581E-3 | 5.070E-3 | 2.873E-3 | 1.338E-3 | 1.055E-3 | 3.473E-3 | 9.944E-3 |
| 430 | 3.422E-3 | 4.588E-3 | 2.562E-3 | 1.190E-3 | 9.327E-4 | 2.971E-3 | 8.318E-3 |
| 431 | 3.128E-3 | 4.108E-3 | 2.253E-3 | 1.044E-3 | 8.093E-4 | 2.460E-3 | 6.640E-3 |
| 432 | 3.000E-3 | 3.660E-3 | 1.946E-3 | 8.994E-4 | 6.855E-4 | 1.960E-3 | 4.987E-3 |
| 433 | 3.116E-3 | 4.151E-3 | 2.298E-3 | 1.071E-3 | 8.287E-4 | 2.448E-3 | 6.497E-3 |
| 434 | 3.043E-3 | 4.006E-3 | 2.204E-3 | 1.027E-3 | 7.909E-4 | 2.304E-3 | 6.021E-3 |
| 435 | 2.962E-3 | 3.921E-3 | 2.150E-3 | 1.001E-3 | 7.686E-4 | 2.220E-3 | 5.736E-3 |
| 436 | 2.951E-3 | 3.908E-3 | 2.142E-3 | 9.976E-4 | 7.656E-4 | 2.216E-3 | 5.723E-3 |
| 437 | 3.014E-3 | 3.968E-3 | 2.181E-3 | 1.015E-3 | 7.813E-4 | 2.289E-3 | 5.974E-3 |

Figure 14. Example MT_DataToVTK.x Data File Containing Seven Fields.
18. Summary

In the preceding sections, each mesh tool that is provided with PROTEUS is described. In all cases, command line input of some form is expected if not required. In the case of MT_BuildAxialMesh.x and MT_RadialLattice.x, an additional control input file is required that describes how the final mesh is to be constructed. For the MT_MeshToVTK.x and MT_MeshToHDF5.x codes, an additional region to color map file is optionally required to create a user defined color setup, but this can greatly assist in the visualization and understanding of the mesh materials.

All of these mesh tools are tested routinely with a series of meshes that have known results. In most cases they run in a few milliseconds with the only exception being the creation of meshes with \( \gg 1 \) million vertices. In these cases the run times can be on the order of a minute. Compared with a commercial mesh generator, these times are virtually negligible and further, given that the mesh tools provide user friendly naming conventions rather than abstract numbers, they are vastly more convenient to use. In most cases, the bulk of the time is spent writing the mesh or VTK file to the disk rather than performing operations. In time, we anticipate the UFmesh capability being a more rigorously used option for generating reactor lattice geometries as a consequence.
References

Appendix A. UFmesh: A User Friendly Mesh Format

The UFmesh format was recently added to the PROTEUS code to assist in the creation of meshes for pin-cells and better control the meshing behavior than what is practical in CUBIT. In this appendix, we cover the input options for the UFmesh and give several examples of how to create input geometries.

To begin, the UFmesh is a keyword-based input description where the keywords are given in Table 26. The current UFmesh capability only supports Cartesian and Hexagonal geometries where Figure 15 shows the reference face numbering and sector numbering for both of these geometries. To understand the input in Table 26, it is easiest to look at the example inputs provided in Figure 16 through Figure 20. In each figure we include the resulting plot of the region naming (arbitrary color) associated with each UFmesh file. It is extremely important to note that UFmesh only creates linear two-dimensional meshes of circular geometries and thus curvilinear surfaces are always represented using faceted surfaces. Thus all cylinders are linearly tessellated and the radius of each cylinder is adjusted to give the correct volume. While the input does allow the conversion to higher order basis functions (refer to the MT_ChangeFEbasis.x mesh tool for a description of this operation), the cylindrical surfaces are always based upon the linear tessellation. To create 3D meshes, one should study how to use the MT_BuildAxialMesh.x mesh tool to create a three-dimensional mesh from each two-dimensional UFmesh file. An example of this was given in the MT_RadialLattice.x mesh code section.

For any UFmesh file, the Grid_Type keyword input must be listed first as it specifies the type of geometry being constructed and how large it is. For example, Figure 16 and Figure 17 both define 5 x 5 grids while in Figure 18, a 1 x 1 grid problem is specified. For hexagonal geometries, one MUST use an odd number of I and J positions, which correspond to an assembly of hex pins laid out in rings. The corresponding pictures with the first two examples show a 5 x 5 grid while the third example has only a single hex. The second keyword input should be the Grid_Pitch as it must be known to define the size of pin-cells. This quantity is simply the pin pitch and should be set appropriately such that the grid of pins does not exceed the inner most duct wall (or assembly) pitch.

To specify an assembly pitch larger than the grid of pins, one must include the Assembly_Sizes keyword input. In Figure 16, a single value is given for Assembly_Sizes which indicates the final assembly pitch. In this case the assembly pitch is not that important, but it is error-checked to ensure that the grid of pin-cells is not larger than the assembly pitch.

In Figure 17, two values are given for Assembly_Sizes, the first value indicating the location of an assembly duct region named DUCT1, and the second value indicating the assembly pitch. From the associated picture, one can see a thin red region surrounding the domain which corresponds to the DUCT1 region that was defined. Looking closely at the resulting mesh, we see that the creation of this duct causes a warping of the pin-cell mesh algorithm for those pin-cells on the border such that the gaps between the pin-cells and the inner duct wall are filled. To
prevent poor quality meshes, one should not specify a duct region that is extremely far away from
the pin edges (which is the case for this example problem). It is suggested to include an additional
fake duct wall closer to the pin cells if necessary to get a good mesh.

The Assembly_Meshes input is required in Figure 17 to define the number of radial meshes in
the duct region. The Assembly_Regions input is also required in order to name the resulting duct
region. The Assembly_Meshes and Assembly_Regions inputs are not required in the Figure 16 or
Figure 17 examples because there is no duct present.

Both the Figure 16 and Figure 17 examples have pin-cells and thus include the Pincell_Radii,
Pincell_Regions, and Pincell_Meshes keywords. The order of these inputs is not important, only
that they appear after the Grid_Type keyword input. Referencing Table 26, one can see that the
first input on each of these keyword inputs is an alias name for the pin-cell itself. This is done to
identify the same pin cell across multiple keywords and also for later use in the Grid_Map input.
Focusing on Figure 16, one can see that three distinct pin-cells are defined. The first one, A1,
specifies that there are three rings named Fuel1, Fuel1, and Clad1 surrounded by a region named
moderator. The first two region names are set to be the same to demonstrate that boundary layer
meshing can be applied by simply including additional radii with identical region names. The
second pin-cell, A2, specifies 5 radial rings where the same Fuel1 and Moderator region names
from A1 were used but additional names for cladding and poison were added. Because all of the
pin-cells have the same central region names and moderator region names, the coloring in the
picture is identical between these regions. The last pin-cell in Figure 16 does not specify any
radial rings and is an example of defining a missing pin which can occur in some types of fuel
assemblies.

A quick comparison of the pin-cell information in Figure 17 to Figure 16 indicates an exact
duplication of the input. This was done to show the differences in output between the two
examples. A comparison of the mesh pictures provided in both example inputs shows a base 8
region meshing scheme in the Cartesian case and a 12 region geometrical meshing scheme. Note
that this matches the sector geometry definitions in Figure 15. The default value of
Pincell_Azimuthal input is 1 mesh per sector where the sectors are shown in Figure 15. The
addition of the Pincell_Azimuthal input in Figure 17 results in a much finer representation of the
pin-cells where each surface of the pin-cell is split into 8 meshes (each pin-cell surface has 2
sectors each of which is split into 4 meshes by the Pincell_Azimuthal input). This pattern can
more easily be seen in Figure 18 where Pincell_Azimuthal is set to 2 and 24 (6 surfaces x 2
sectors x 2 azimuthal segments) total meshes are created along the border of the hex pin-cell.

The last input to consider is the Grid_Map input which specifies the layout of pin-cells in the
grid. For the Cartesian case in Figure 16, the corners of the 5 x 5 grid are filled with the letter “O”
or word “NULL” which are interpreted by the input to be non-meshed regions of the domain. The
picture shows the same loading pattern as the ascii input noting that the grid_map input is reverse
ordered with respect to Y in order for the picture and grid map to match. This infers that the 1,1
index position of I-J is at the origin and that any higher values of I or J lead to increases in the x
or y position. The hexagonal input of Figure 17 shows a similar case, but here, there are six total pin-cells that are set as NULL to get the desired hexagonal pattern. If one does not specify any ducts and fills the complete lattice of pin-cells, then one obtains a full hex lattice of pins like that shown in Figure 21. Note that the Assembly_Sizes input was left off of this example in order to prevent it from creating an outer hexagonal boundary around the lattice of pins. If one provides these extra pin-cells and Assembly_Sizes input, then the code will fail with errors if the lattice of pins intersects the boundary or produce a very poor mesh if the lattice of pins does not intersect the innermost boundary (i.e. do not use this approach). It is important to note that one should avoid providing mesh input that produces very poor quality meshes for PROTEUS-SN.

The final two example problems demonstrate fine-grained control of the meshing scheme. Figure 19 shows how to use sector and face meshing controls for hexagonal geometries while Figure 20 shows how to use sector and face meshing controls for Cartesian geometries. Unlike the other examples, the Pincell_Azimuthal input has 12 inputs for Figure 19 and 8 inputs for Figure 20 (as opposed to a single input). In Figure 19, sectors 1 and 2 of each pin-cell are more finely meshed then the remaining sectors (4 versus a 1). Looking at the mesh picture itself, one sees that because we include a pin-cell lattice, the program is forced to make sectors 7 and 8 match the mesh settings in 1 and 2 such that the entire problem can be meshed. Note that this is just an example mesh and should not in practice be used.

To control the meshing along the assembly border such that assemblies can later be merged into a core layout, the Assembly_Border input in Figure 19 is specified to produce 16 meshes per surface. One should understand that without this input there would be 11 or 21 meshes per surface. The restriction to 16 meshes requires the addition of a fake “duct” region around the pin lattice which is triangle meshed rather than quad meshed. The triangle mesh can be very poor quality if the number of intervals is considerably different. Looking at Figure 20, we again use the Pincell_Azimuthal and Assembly_Border inputs to alter the behavior. In this case we see a much better quality mesh as the number of intervals is more comparable between the pins and the duct surface.

Overall, the UFmesh capability allows users to quickly define meshes for reactor problems. The only real question that remains that is not covered by the example problems is how to merge assembly meshes together that have different sizes and numbers of pins. To merge several assemblies together, we strongly recommend using the Assembly_Border combined with the Pincell_Azimuthal input to define the mesh.
Table 26. Keyword Input Options for the UFmesh File Format

<table>
<thead>
<tr>
<th>Key Word</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid_type</td>
<td>&lt;name&gt; &lt;I size&gt; &lt;J size&gt;</td>
<td>Define the type of geometry being setup and its grid size in both I and J directions. Name = {hexagonal, Cartesian} Note: I and J must be odd for hexagonal type grids</td>
</tr>
<tr>
<td>Grid_pitch</td>
<td>&lt;I pitch&gt; &lt;J pitch&gt;</td>
<td>Defines the spatial pitch of the grid being constructed (pin-cell pitch)</td>
</tr>
<tr>
<td>Element_order</td>
<td>&lt;integer order&gt;</td>
<td>Defines the basis order of the finite element mesh</td>
</tr>
<tr>
<td>Mesh_rotate</td>
<td>&lt;rotation angle&gt;</td>
<td>Defines an optional rotation of the mesh. The counter-clockwise rotation angle is consistent with the MT_ModifyMesh.x code.</td>
</tr>
<tr>
<td>Mesh_translate</td>
<td>&lt;X movement&gt; &lt;Y movement&gt;</td>
<td>Defines an optional X and Y translation of the mesh</td>
</tr>
<tr>
<td>Assembly_sizes</td>
<td>&lt;duct FTF 1&gt; … &lt;duct FTF N&gt; &lt;Assembly pitch&gt;</td>
<td>Defines any duct flat-to-flat pitches and the assembly flat-to-flat pitch</td>
</tr>
<tr>
<td>Assembly_regions</td>
<td>&lt;duct name 1&gt; … &lt;duct name N&gt;</td>
<td>Defines the region names of each consecutive duct</td>
</tr>
<tr>
<td>Assembly_meshes</td>
<td>&lt;meshes 1&gt; … &lt;meshes N&gt;</td>
<td>Defines the number of radial meshes in each duct region</td>
</tr>
<tr>
<td>Assembly_border</td>
<td>&lt;integer value&gt; [integer value] …</td>
<td>Sets the number of intervals on each face of a hexagonal (6 inputs) or Cartesian (4 inputs) assembly. A single input can be given to set all faces to the same value. Mesh symmetry is imposed in multi-pin cases.</td>
</tr>
<tr>
<td>Pincell_radii</td>
<td>&lt;alias name&gt; &lt;radius 1&gt;…&lt;radius R&gt;</td>
<td>Defines the radius setup of the pin-cell given by the alias name</td>
</tr>
<tr>
<td>Pincell_regions</td>
<td>&lt;alias name&gt; &lt;name 1&gt;…&lt;name R&gt;</td>
<td>Defines the region name of each radial ring in the pin-cell given by the alias name</td>
</tr>
<tr>
<td>Pincell_meshes</td>
<td>&lt;alias name&gt; &lt;meshes 1&gt;…&lt;meshes R&gt;</td>
<td>Defines the number of radial meshes in each radial ring for the pin-cell given by the alias name</td>
</tr>
<tr>
<td>Pincell_azimuthal</td>
<td>&lt;integer value&gt; [integer value] …</td>
<td>Defines the number of azimuthal meshes per half face of a hexagonal (12 inputs) or Cartesian (8 inputs). A single input can be given to set all half-faces to the same value. Mesh symmetry is imposed in multi-pin cases. Sector ordering is given in Figure 15.</td>
</tr>
<tr>
<td>Grid_map</td>
<td>&lt;Index J&gt; &lt;name 1&gt; …&lt;name X size&gt;</td>
<td>Defines the layout of the pin-cells in the assembly grid</td>
</tr>
</tbody>
</table>
Figure 15. Face (outside) and Sector (inside) Numbering for Cartesian (left) and hexagonal (right) Geometry
GRID_TYPE CARTESIAN  5  5
GRID_PITCH  2.0  2.0
ELEMENT_ORDER  1
ASSEMBLY_SIZES 10.0
PINCELL_RADII  A1  0.1  0.5  0.75
PINCELL_REGIONS  A1  FUEL1  FUEL1  CLAD1  MODERATOR
PINCELL_MESHES  A1  1  2  1  2
PINCELL_RADII  A2  0.1  0.5  0.6  0.7  0.75
PINCELL_REGIONS  A2  FUEL1  FUEL1  INNERCLAD2  POISON2  OUTERCLAD2  MODERATOR
PINCELL_MESHES  A2  1  2  1  1  1  2
PINCELL_REGIONS  A3  MODERATOR
PINCELL_MESHES  A3  3
GRID_MAP  5  O  A1  A1  A1  O
GRID_MAP  4  A1  A2  A2  A2  A1
GRID_MAP  3  A1  A2  A3  A2  A1
GRID_MAP  2  A1  A2  A2  A2  A1
GRID_MAP  1  O  A1  A1  A1  NULL

Figure 16. Example Cartesian Assembly UFmesh File

GRID_TYPE HEXAGONAL  5  5
GRID_PITCH  2.0
ELEMENT_ORDER  2
ASSEMBLY_SIZES  10.20  10.3
ASSEMBLY_REGIONS  DUCT1
ASSEMBLY_MESHES  1
PINCELL_RADII  A1  0.1  0.5  0.75
PINCELL_REGIONS  A1  FUEL1  FUEL1  CLAD1  MODERATOR
PINCELL_MESHES  A1  1  2  1  2
PINCELL_RADII  A2  0.1  0.5  0.6  0.7  0.75
PINCELL_REGIONS  A2  FUEL1  FUEL1  INNERCLAD2  POISON2  OUTERCLAD2  MODERATOR
PINCELL_MESHES  A2  1  2  1  1  1  2
PINCELL_REGIONS  A3  MODERATOR
PINCELL_MESHES  A3  3
GRID_MAP  5  A1  A1  A1  O  O
GRID_MAP  4  A1  A3  A2  A1  NULL
GRID_MAP  3  A1  A2  A1  A2  A1
GRID_MAP  2  O  A1  A2  A3  A1
GRID_MAP  1  O  O  A1  A1  A1
PINCELL_AZIMUTHAL  4

Figure 17. Example Hexagonal Assembly UFmesh File
GRID_TYPE HEXAGONAL  1  1
GRID_PITCH     8.0
!MESH_ROTATE 30.0
ELEMENT_ORDER 1
PINCELL_REGIONS A1 OUTERCORE
PINCELL_MESHES A1 2
GRID_MAP   1 A1
PINCELL_AZIMUTHAL 2

Figure 18. Example Hexagonal Homogeneous Pin-cell UFmesh File

GRID_TYPE HEXAGONAL  5  5
GRID_PITCH     1.0
ELEMENT_ORDER 1
ASSEMBLY_REGIONS DUCT1
ASSEMBLY_MESHES 1
ASSEMBLY_SIZES 4.5 4.6
PINCELL_RADII A1 0.4
PINCELL_REGIONS A1 DRAP DILUENT
PINCELL_MESHES A1 1  1
GRID_MAP   5 A1 A1 A1 0  0
GRID_MAP   4 A1 A1 A1 A1 NULL
GRID_MAP   3 A1 A1 A1 A1 A1
GRID_MAP   2 0 A1 A1 A1 A1
GRID_MAP   1 0 0 A1 A1 A1
PINCELL_AZIMUTHAL 4 4 1 1 1 1 1 1 1 1 1 1
ASSEMBLY_BORDER 16 16 16 16 16 16

Figure 19. Example Hexagonal Sector and Face Meshing Control UFmesh File
GRID_TYPE CARTESIAN 5 5
GRID_PITCH 2.0 2.0
ELEMENT_ORDER 1
ASSEMBLY_SIZES 10.0 11.0 12.0
ASSEMBLY_REGIONS DUCT1 DUCT2
ASSEMBLY_MESHES 1 2
PINCELL_RADII A1 0.1 0.5 0.75
PINCELL_REGIONS A1 FUEL1 FUEL1 CLAD1 MODERATOR
PINCELL_MESHES A1 1 2 1 2
PINCELL_RADII A2 0.1 0.5 0.6 0.7 0.75
PINCELL_REGIONS A2 FUEL1 FUEL1 INNERCLAD2 POISON2 OUTERCLAD2 MODERATOR
PINCELL_MESHES A2 1 2 1 1 1 2
PINCELL_REGIONS A3 MODERATOR
PINCELL_MESHES A3 3
GRID_MAP 5 A2 A1 A1 A2
GRID_MAP 4 A1 A2 A2 A1
GRID_MAP 3 A1 A2 A3 A2 A1
GRID_MAP 2 A1 A2 A2 A2 A1
GRID_MAP 1 A3 A1 A1 A1 A3
PINCELL_AZIMUTHAL 3 4 3 4 3 4 3 4
ASSEMBLY_BORDER 22 18 22 18

Figure 20. Example Cartesian Sector and Face Meshing Control UFmesh File
Figure 21. Example Hexagonal Pin-cell Lattice without a Duct Restriction on the Geometry
Appendix B. Grid: A File Format Specification for Structured Grid Input

This section discusses the GRID file format which is meant for the PROTEUS-NODAL code but can be used for the PROTEUS-SN code as mesh generation. As is the case for much of the rest of PROTEUS, the GRID input specification is a keyword input where all available keywords are defined in Table 27. Unlike the ufmesh format, the keyword input for the grid format does not have to be in any particular order. We provide four example problems noting that there are several verification problems that demonstrate the usage of the grid input format.

Unlike other structured grid based codes, the geometrical input of grid is always a three-dimensional array of size (I,J,K). As a consequence, all of the grid inputs will be three dimensional in the X-Y-Z coordinate system and one uses the Slice_at_* inputs to actually select a lower dimensional portion of the 3D geometry. The definition of the third dimension (K) is done using the “assembly” keyword input which defines a 1D array of region names and region dimensions. The two-dimensional geometry (I,J) is then a simple mapping of each assembly into an I-J matrix position. Based upon experience, this maximizes the readability of conventional assembly homogenization schemes and minimizes the input required to define a 3D domain. It can be a considerable increase in information for 2D and 1D geometries.

We start by looking at the extensive keyword input noting that only a small fraction of the inputs are actually needed for a mesh to be built. The most important keyword is the geometry type declaration as this one defines how all of the rest of the input is interpreted. The options include the three basic Cartesian geometries (X, X-Y, and X-Y-Z), two hexagonal geometries, and two regular triangular geometries. In the case of 1D Cartesian, one constructs a three-dimensional input and uses slice_at_x and slice_at_y to select a particular “assembly.” One can alternatively use slice_at_x and slice_at_z to select a particular radial portion of the assemblies. Figure 22 through Figure 25 show examples of a 2D Cartesian geometry, 3D Cartesian geometry, 3D hexagonal geometry, and 3D regular triangular geometry, respectively. Figure 22 is the only example of the use of Slice_at_z to reduce the 3D input to a 2D one. The picture on the right hand side of Figure 22 shows the region number (random color assignment) of each finite element in the mesh. The geometry orientation of the hexagonal and regular triangular lattice is such that the first hexagon ring always extends down the X axis. The regular triangular meshing scheme shown in Figure 26 shows the X-Y coordinate directions noting that the central hex will always be centered at 0-0 by default. This orientation is -30 degrees from most structured grid hexagonal or regular triangular codes of note, but is extremely easy to work with.

The next keyword input of note is the locate_origin one which can be used to translate the user geometry to any coordinate position in the physical space. This is most useful if during a multi-physics calculation, the coordinate location of an existing neutronics problem needs to be translated to match another physics codes coordinate system. The next four keyword inputs, Ritz_xy, Ritz_z, Ritz_source, and Ritz_surface are inputs to set the spatial approximation in the
PROTEUS-NODAL code and thus beyond the scope of this report as they do not impact the mesh creation.

The element_order, element_type, and element_scheme are the most important control parameters for the mesh generation. The element_order is rather straightforward as it allows the selection of linear, quadratic, etc… basis functions. The element_type input only has three input options of serendipity, lagrangian, and gauss-lobatto. The linear order of these three element types is identical and the main differences start at quadratic order basis functions. For a detailed explanation, consult the element information that is provided with the PROTEUS-SN or a book on finite elements. The element_scheme is by far the most vague of the three and only has the three options of BOXES, PRISMS, and ALTBOX. Figure 26 shows the meshing scheme for a grid point in the Cartesian and hexagonal geometry option. Note that the minimal triangular geometry grid must contain six triangles and thus we show the meshing of six meshes in that grid geometry. As can be seen in this figure, the PRISMS case generally yields more elements than the BOXES approach. In practice it also leads to more vertices in the final mesh. The ALTBOX scheme was added to produce better meshes for geometries with certain symmetry and periodic distributions. For the regular triangular geometry, only the prism meshing scheme is available at this time although the addition of a BOXES approach has been discussed.

The next keyword inputs in Table 27 are hex_pitch, tri_pitch, xgrid, ygrid, and zgrid. All of the input on these lines is interpreted as cm. This input is read independent of the rest of the grid input and is used as necessary. As an example, if XY geometry is specified and a hex_pitch is given, then the hex_pitch will be ignored. Similarly, if a Hex geometry is specified and xgrid input is given, then the xgrid input will be ignored. For all geometries, one should include a zgrid specification although it is not required. This input is the only way to easily control the mesh size without having to add additional region boundaries in the assembly input. The xgrid and ygrid inputs are required for any Cartesian geometry as the input options to specify the I-J grid fundamentally has no dimensions. For hex geometries, one does not need xgrid or ygrid, but must provide a hex_pitch which is the flat to flat distance for a hexagon. Similarly, regular triangular geometries do not need xgrid or ygrid input and one must provide a tri_pitch. The tri_pitch size is equivalent to the hex_pitch and represents the height of two triangular elements. For xgrid, ygrid, and zgrid, the lower and upper coordinates of a desired mesh are given which are considered a “coarse” grid. The last integer specifies any subdivision of each coarse meshes which are considered a “fine” grid. If either the zgrid boundaries given or those resulting from the finer meshes are not present in the assembly geometry they will all be added.

As mentioned, the assembly keyword input in Table 27 is a one dimensional geometry definition and must have four inputs per line. The first input is the assembly ‘name’ which is not preserved in the either the mesh or grid geometry. The second input is the region name which is preserved in the mesh and grid geometry. This region name is used when mapping compositions to regions of the geometry and thus has identical meaning for PROTEUS-Nodal and PROTEUS-SN. The last two inputs on the assembly keyword are the lower z and upper z coordinate over
which the region name is to be applied. This input is overlay type input such that the latest line
overrules all previous assignments of region names to geometry. Figure 23 shows the best
example of its usage where region R_WATER is used to overwrite the region R_UO2 in
assembly P_UO2 between 2.0 and 5.0 cm. Note how all of the assembly definitions in the
example problems have the same overall z dimensions. This is required to define a structured grid
problem and any inconsistencies in total assembly geometry will be flagged as a fatal error.

There are two main ways to define the I-J grid of assembly input for the X-Y plane. The
position input is the first way, the first option of which allows the user to set the assembly
definition in every i-j position in the domain. The first input on the position input is the assembly
name followed by the integer value of the I and J coordinate. For Cartesian geometries only, the
number of i-j positions should exactly match the number of coarse grid specifications given on
the xgrid and ygrid inputs. For hexagonal and regular triangular geometry, the extent of the i-j
position input option will invoke the creation of an appropriately sized I-J grid. In both hexagonal
and regular triangular geometries, the domain is checked to be convex and appropriate failure
messages are given if the domain is not convex.

For convenience, we provide an alternate input for hexagonal and regular triangular
geometries which is the second option of the position input. In this case the first input is the
assembly name while the second is the hexagonal ring. The hex rings are number conventionally
noting that the maximum hexagonal ring will invoke the creation of an I-J grid that is large
enough to fit a full core hex map. The third and fourth input specify the starting and ending hex
position along the ring. The maximum hex position of any ring is computed to be (R-1)*6. Much
like the assembly keyword input, an overlay approach is taken where each consecutive line
overwrites the information given on previous input lines. Setting these values to 0 and 0 will
cause the entire ring to be filled with the stated assembly. The fifth input is optional and only
meaningful for triangular geometries. In this case the triangle numbering that is given in Figure
26 shows how to assign assembly compositions to specific triangles. Note that the coordinate
access directions are given for appropriate orientation. If one uses the optional triangle input, one
would have to include six different lines to assign different compositions to each triangle. If a
triangular grid is desired, we strongly suggest using the gridmap input or I-J position input as
alternatives.

The other I-J geometry input option is to specify a gridmap. The keyword input description
from Table 27 consists of a J index position followed by I total ‘name’ values. These ‘names’ are
intended to be the assembly names defined from the Assembly keyword input but for
convenience, an ALIAS keyword option was added to the grid input such that an alternative name
can be used for the assembly when giving the gridmap input. Figure 24 shows an excellent
example of a gridmap and alias keyword input for hexagonal geometry. Looking at the picture in
this figure, one can see that the gridmap input specifies a 5 ring hexagonal geometry. The
gridmap of course specifies a 9 x 9 array of input which is the minimal required to contain a full 5
ring hexagonal geometry. For each position that does not have an assembly assigned to it, the
keyword “O” is used where “NULL” is a valid alternative. Note that one could have defined a 20 x 20 grid and still only have filled a 9 x 9 portion of the domain. The Cartesian gridmap input is rather straightforward to understand and Figure 23 shows an easy to follow example. The regular triangular geometry is rather difficult to understand using the gridmap input and the example in Figure 25 is basically a duplicate of the hexagonal geometry gridmap input. The resulting picture is rather difficult to understand, but a close inspection shows that each J row of the gridmap input yields a row of triangles in the picture. If one fills the entire map with assembly names instead of using the “O” null settings, one would see a parallelogram of triangles that is 9 triangles wide and 9 rows of triangles high. Conventionally, using the hex geometry position input option, one would find the triangular grid would have twice as many I positions as J positions in order to store the full grid.

The last part of each example input are the boundary condition specifications which consist of periodic_bc, z_periodic_bc, symmetry_bc, and boundary_condition. The periodic_bc and symmetry_bc should both not be included as they infer different treatments to the same geometry. The periodic boundary conditions, if given a valid I-J grid, will impose 90 degree periodicity in Cartesian geometries and 60, 120, or 180 degree periodicity in hexagonal or regular triangular geometries. Remember that the full map is built in all cases and that this input option only enforces a grid assembly specification that has the rotational periodicity. The symmetry_bc keyword has a similar impact in that it imposes 45 degree symmetry in Cartesian or 30, 60, and 90 degree symmetry in hexagonal and regular triangular geometries. The z_periodic_bc is a special case of periodicity that is allowed in the z direction and thus only appropriate for 3D Cartesian, hexagonal, or regular triangular geometries. The boundary_condition keyword input is the actual way in which the boundary conditions are specified. The position name input specifies the surface of the domain and is always considered the boundary conditions applied to the full geometry. The boundary types supported are either reflected or void (vacuum) which are conventional for neutronics. From the four example problems, one can see that the boundary condition setup are all appropriately defined for each given geometry in that X,Y,Z are used in Cartesian and hex000 through hex300 are given for hexagonal and +Z and –Z for the axial boundary conditions. For regular triangular geometries, one uses the same boundary conditions as for the hexagonal geometry as the interfaces are identical. One should note that all boundary conditions for hex should always be given as are all boundary conditions X,Y,Z for Cartesian.

Overall, the preceding grid library input should provide a convenient way to define a conventional structured grid homogenized assembly input geometry. While this geometry was envisioned for use within the PROTEUS-Nodal code, the geometry and meshing are available to the mesh library of the PROTEUS code and thus any component of PROTEUS can use this geometry option as a way to define mesh input. The four examples here should be sufficient to explain how to use the grid input and control the meshing schemes that are available for it.
### Table 27. Keyword Input Options for the GRID File Format

<table>
<thead>
<tr>
<th>Key Word</th>
<th>Format</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>&lt;Name&gt;</td>
<td>Specify the type of grid being constructed &lt;X, XY, XYZ, Hexagonal, Hexagonal-Z, Triangular, Triangular-Z&gt;</td>
</tr>
<tr>
<td>Slice_at_x</td>
<td>&lt;x position&gt;</td>
<td>For 1D domains, you give it a 3D domain and tell it to pick this X plane for evaluation</td>
</tr>
<tr>
<td>Slice_at_y</td>
<td>&lt;y position&gt;</td>
<td>For 1D domains, you give it a 3D domain and tell it to pick this Y plane for evaluation</td>
</tr>
<tr>
<td>Slice_at_z</td>
<td>&lt;z position&gt;</td>
<td>For 2D domains, you give it a 3D domain and tell it to pick this Z plane for evaluation</td>
</tr>
<tr>
<td>Locate_origin</td>
<td>&lt;axis name&gt; &lt;value&gt;</td>
<td>Input option to translate the domain &lt;axis name&gt; = &lt;X&gt; &lt;Y&gt; &lt;Z&gt;</td>
</tr>
<tr>
<td>Ritz_xy</td>
<td>&lt;integer value&gt;</td>
<td>PROTEUS-Nodal x-y spatial approximation</td>
</tr>
<tr>
<td>Ritz_z</td>
<td>&lt;integer value&gt;</td>
<td>PROTEUS-Nodal z spatial approximation</td>
</tr>
<tr>
<td>Ritz_source</td>
<td>&lt;integer value&gt;</td>
<td>PROTEUS-Nodal source spatial approximation</td>
</tr>
<tr>
<td>Ritz_surface</td>
<td>&lt;integer value&gt;</td>
<td>PROTEUS-Nodal leakage spatial approximation</td>
</tr>
<tr>
<td>Element_order</td>
<td>&lt;integer value&gt;</td>
<td>Order of the FE basis to build</td>
</tr>
<tr>
<td>Element_type</td>
<td>&lt;basis type&gt;</td>
<td>Type of FE element to use &lt;Serendipity, lagrangian, Gauss-lobatto&gt;</td>
</tr>
<tr>
<td>Element_scheme</td>
<td>&lt;scheme name&gt;</td>
<td>Type of meshing scheme to use &lt;boxes, prisms, altbox&gt;</td>
</tr>
<tr>
<td>Hex_pitch</td>
<td>&lt;value&gt;</td>
<td>Defines the hexagonal pitch of the structured grid</td>
</tr>
<tr>
<td>Tri_pitch</td>
<td>&lt;value&gt;</td>
<td>Defines the triangular pitch of the structured grid</td>
</tr>
<tr>
<td>Xgrid</td>
<td>&lt;lower x&gt; &lt;upper x&gt; &lt;subintervals&gt;</td>
<td>Used to define coarse X grid dimensions and meshing</td>
</tr>
<tr>
<td>Ygrid</td>
<td>&lt;lower y&gt; &lt;upper y&gt; &lt;subintervals&gt;</td>
<td>Used to define coarse Y grid dimensions and meshing</td>
</tr>
<tr>
<td>Zgrid</td>
<td>&lt;lower z&gt; &lt;upper z&gt; &lt;subintervals&gt;</td>
<td>Used to define coarse Z grid dimensions and meshing</td>
</tr>
<tr>
<td>Assembly</td>
<td>&lt;name&gt; &lt;region name&gt; &lt;lower z&gt; &lt;upper z&gt;</td>
<td>Defines a 1D geometry of regions or the 3D aspects of a homogenized assembly</td>
</tr>
<tr>
<td>Position</td>
<td>&lt;name&gt; &lt;I position&gt; &lt;J position&gt;</td>
<td>Two options are supported noting that [triangle] input is ignored for hexagonal geometries. For each input line, the assembly name to be loaded at the indicated position is given. &lt;Start Hex&gt; and &lt;End Hex&gt; are used</td>
</tr>
<tr>
<td>Key Word</td>
<td>Format</td>
<td>Purpose</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alias</td>
<td>&lt;alias name&gt; &lt;assembly name&gt;</td>
<td>Each assembly name can be given a smaller name to make the gridmap input more compact</td>
</tr>
<tr>
<td>gridmap</td>
<td>&lt;index J&gt; &lt;Alias 1&gt; &lt;Alias 2&gt;… &lt;Alias I&gt;</td>
<td>Used to specify a 2D map of data which indicates how assemblies are to be loaded into the structured grid</td>
</tr>
<tr>
<td>Periodic_bc</td>
<td>&lt;angle to apply periodicity&gt;</td>
<td>Used to automatically fill a structured grid with periodic loading of assemblies. &lt;60, 90, 120, 180&gt; are supported</td>
</tr>
<tr>
<td>Z_periodic_bc</td>
<td>&lt;yes/no&gt;</td>
<td>Used to indicate that the axial direction is periodic</td>
</tr>
<tr>
<td>Symmetry_bc</td>
<td>&lt;angle to impose symmetry&gt;</td>
<td>Used to automatically fill a structured grid with symmetric loading of assemblies &lt;30, 45, 60, 90&gt; are supported</td>
</tr>
<tr>
<td>Boundary_condition</td>
<td>&lt;position name&gt; &lt;boundary type&gt;</td>
<td>Main path to define boundary conditions &lt;position&gt; = &lt;-X,+X,-Y,+Y,-Z,+Z&gt; &lt;position&gt; = &lt;hex000, hex060, hex120,…hex300&gt; &lt;boundary type&gt; = &lt;reflective, void&gt;</td>
</tr>
</tbody>
</table>
Element_Order 2
Element_Scheme BOXES
GEOMETRY XY
SLICE_AT_Z 0.5
RITZ_XY 10
RITZ_Z 10
RITZ_SURFACE 4
ASSEMBLY P_UO2 R_UO2 0.0 1.0
ASSEMBLY P_MOX8 R_MOX8 0.0 1.0
ASSEMBLY P_MOX7 R_MOX7 0.0 1.0
ASSEMBLY P_MOX4 R_MOX4 0.0 1.0
ASSEMBLY P_GTubFisC R_GTub 0.0 1.0
ASSEMBLY P_WATER R_WATER 0.0 1.0
XGRID 0.0 2.0 1
XGRID 2.0 4.0 1
XGRID 4.0 6.0 1
XGRID 6.0 8.0 1
XGRID 8.0 12.0 1
YGRID 0.0 2.0 1
YGRID 2.0 4.0 1
YGRID 4.0 6.0 1
YGRID 6.0 8.0 1
YGRID 8.0 12.0 1
ZGRID 0.0 1.0 1
POSITION P_WATER 1 1
POSITION P_UO2 2 1
POSITION P_WATER 3 1
POSITION P_MOX8 4 1
POSITION P_WATER 5 1
POSITION P_UO2 1 2
POSITION P_MOX7 4 2
POSITION P_WATeR 5 2
POSITION P_WATeR 1 3
POSITION P_UO2 2 3
POSITION P_WATeR 3 3
POSITION P_MOX4 4 3
POSITION P_WATeR 5 3
POSITION P_MOX8 1 4
POSITION P_MOX7 2 4
POSITION P_MOX4 3 4
POSITION P_UO2 4 4
POSITION P_WATeR 5 4
POSITION P_WATeR 1 5
POSITION P_WATeR 2 5
POSITION P_WATeR 3 5
POSITION P_WATeR 4 5
POSITION P_WATeR 5 5
PERIODIC_BC 90
BOUNDARY_CONDITION -X REFLECTIVE
BOUNDARY_CONDITION +X VOID
BOUNDARY_CONDITION -Y REFLECTIVE
BOUNDARY_CONDITION +Y VOID

Figure 22. Example Two-Dimensional Cartesian Grid File
Element_Order 2
Element_Scheme BOXES
GEOMETRY XYZ
RITZ_XY 4
RITZ_2 4
RITZ_SURFACE 1
ASSEMBLY P_UO2 R_UO2 0.0 10.0
ASSEMBLY P_UO2 R_WATER 2.0 5.0
ASSEMBLY P_MOX8 R_MOX8 0.0 10.0
ASSEMBLY P_MOX7 R_MOX7 0.0 10.0
ASSEMBLY P_MOX4 R_MOX4 0.0 10.0
ASSEMBLY P_GTubFisC R_GTub 0.0 10.0
ASSEMBLY P_WATER R_WATER 0.0 10.0
XGRID 0.0 2.0 1
XGRID 2.0 4.0 1
XGRID 4.0 6.0 1
XGRID 6.0 8.0 1
XGRID 8.0 10.0 1
YGRID 0.0 2.0 1
YGRID 2.0 4.0 1
YGRID 4.0 6.0 1
YGRID 6.0 8.0 1
YGRID 8.0 10.0 1
ZGRID 0.0 1.0 1
ZGRID 1.0 2.0 1
ZGRID 2.0 5.0 3
ZGRID 5.0 10.0 5
ALIAS A1 P_UO2
ALIAS A2 P_MOX8
ALIAS A3 P_MOX7
ALIAS A4 P_MOX4
ALIAS A5 P_GTubFisC
ALIAS A6 P_WATER
GRIDMAP 5 A6 A6 A6 A6 A6
GRIDMAP 4 A2 A3 A4 A1 A6
GRIDMAP 3 A6 A1 A6 A4 A6
GRIDMAP 2 A1 A5 A1 A3 A6
GRIDMAP 1 A6 A1 A6 A2 A6
BOUNDARY_CONDITION -X REFLECTIVE
BOUNDARY_CONDITION +X VOID
BOUNDARY_CONDITION -Y REFLECTIVE
BOUNDARY_CONDITION +Y VOID
BOUNDARY_CONDITION -Z REFLECTIVE
BOUNDARY_CONDITION +Z REFLECTIVE

Figure 23. Example Three-Dimensional Cartesian Grid File
Element_Order 1
Element_Scheme prisms
GEOMETRY HEXAGONAL-Z
RITZ_XY 2
RITZ_Z 2
RITZ_SURFACE 0
ASSEMBLY P_UO2 R_UO2 0.0 10.0
ASSEMBLY P_MOX8 R_MOX8 0.0 10.0
ASSEMBLY P_MOX7 R_MOX7 0.0 10.0
ASSEMBLY P_MOX4 R_MOX4 0.0 10.0
ASSEMBLY P_GTubFisC R_FisC 0.0 10.0
ASSEMBLY P_WATER R_WATER 0.0 10.0
HEX_PITCH 3.0
ZGRID 0.0 2.0 1
ZGRID 2.0 10.0 3
ALIAS A1 P_UO2
ALIAS A2 P_MOX8
ALIAS A3 P_MOX7
ALIAS A4 P_MOX4
ALIAS A5 P_GTubFisC
ALIAS A6 P_WATER
GRIDMAP 9   A6 A6 A6 A6 A6 o o o o
GRIDMAP 8   A6 A1 A1 A1 A1 A6 o o o o
GRIDMAP 7   A6 A1 A4 A6 A4 A1 A6 o o
GRIDMAP 6   A6 A1 A6 A2 A3 A6 A1 A6 o
GRIDMAP 5   A6 A1 A4 A3 A5 A2 A4 A1 A6
GRIDMAP 4   o A6 A1 A6 A2 A3 A6 A1 A6
GRIDMAP 3   o o A6 A1 A4 A6 A4 A1 A6
GRIDMAP 2   o o o A6 A1 A1 A1 A6
GRIDMAP 1   o o o o A6 A6 A6 A6 A6
PERIODIC_BC 120
BOUNDARY_CONDITION HEX000 VOID
BOUNDARY_CONDITION HEX060 VOID
BOUNDARY_CONDITION HEX120 VOID
BOUNDARY_CONDITION HEX180 VOID
BOUNDARY_CONDITION HEX240 VOID
BOUNDARY_CONDITION HEX300 VOID
BOUNDARY_CONDITION -Z REFLECTIVE
BOUNDARY_CONDITION +Z REFLECTIVE

Figure 24. Example Three-Dimensional Hexagonal Grid File
<table>
<thead>
<tr>
<th>Element_Order</th>
<th>ALIAS A5 P_GTubFisC</th>
<th>ALIAS A6 P_WATER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element_Scheme</td>
<td>GRIDMAP 9 A6 A6 A6 A6 A6 A6 A6 A6</td>
<td>GRIDMAP 4 A6 A6 A6 A6 A6 A6 A6</td>
</tr>
<tr>
<td>GEOMETRY TRIANGULAR-Z</td>
<td>GRIDMAP 8 A6 A1 A1 A1 A1 A6 A6 A6</td>
<td>GRIDMAP 3 A6 A1 A4 A6 A4 A4 A6</td>
</tr>
<tr>
<td>RITZ_XY 2</td>
<td>GRIDMAP 7 A6 A1 A4 A6 A4 A1 A6 A6</td>
<td>GRIDMAP 2 A6 A1 A1 A1 A1 A6</td>
</tr>
<tr>
<td>RITZ_Z 2</td>
<td>GRIDMAP 6 A6 A1 A6 A1 A6 A1 A6 A6</td>
<td>GRIDMAP 1 A6 A6 A6 A6 A6 A6</td>
</tr>
<tr>
<td>RITZ_SURFACE 0</td>
<td>ASSEMBLY P_UO2 R_UO2 0.0 10.0</td>
<td>ASSEMBLY P_WATER R_WATER 0.0 10.0</td>
</tr>
<tr>
<td>ASSEMBLY P_MOX8 R_MOX8 0.0 10.0</td>
<td>ASSEMBLY P_GTubFisC R_FisC 0.0 10.0</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLY P_MOX7 R_MOX7 0.0 10.0</td>
<td>GRIDMAP 9</td>
<td>ALIAS A5 P_GTubFisC</td>
</tr>
<tr>
<td>ASSEMBLY P_MOX4 R_MOX4 0.0 10.0</td>
<td>ALIAS A6 P_WATER</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLY P_GTubFisC R_FisC 0.0 10.0</td>
<td>GRIDMAP 8</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>HEX_PITCH 3.0</td>
<td>GRIDMAP 7</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ZGRID 0.0 2.0 1</td>
<td>GRIDMAP 6</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ZGRID 2.0 10.0 3</td>
<td>GRIDMAP 5</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ALIAS A1 P_UO2</td>
<td>GRIDMAP 4</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ALIAS A2 P_MOX8</td>
<td>GRIDMAP 3</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ALIAS A3 P_MOX7</td>
<td>GRIDMAP 2</td>
<td>ALIAS A6 P_WATER</td>
</tr>
<tr>
<td>ALIAS A4 P_MOX4</td>
<td>GRIDMAP 1</td>
<td>ALIAS A6 P_WATER</td>
</tr>
</tbody>
</table>

Figure 25. Example Three-Dimensional Regular Triangular Grid File
<table>
<thead>
<tr>
<th></th>
<th>Cartesian</th>
<th>Hexagonal</th>
<th>Regular Triangular</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXES</td>
<td><img src="image" alt="Cartesian Box" /></td>
<td><img src="image" alt="Hexagonal Box" /></td>
<td><img src="image" alt="Regular Triangular Box" /></td>
</tr>
<tr>
<td>PRISMS</td>
<td><img src="image" alt="Cartesian Prism" /></td>
<td><img src="image" alt="Hexagonal Prism" /></td>
<td><img src="image" alt="Regular Triangular Prism" /></td>
</tr>
<tr>
<td>ALTBOX</td>
<td><img src="image" alt="Cartesian Altbox" /></td>
<td><img src="image" alt="Hexagonal Altbox" /></td>
<td><img src="image" alt="Regular Triangular Altbox" /></td>
</tr>
</tbody>
</table>

Figure 26. Grid Finite Element Meshing Schemes (Regular Triangular shows 6 nodes)
Appendix C. NEMESH: A PROTEUS File Format for Finite Element Mesh Input

This section specifies the PROTEUS-SN “nemesh” mesh file format. An example “nemesh” input file is also provided. Ahiadf;lkjasd;flkjjasdf;lkj;lkjdsjfl;lkjasdf

C.1 File Format Description for “nemesh” mesh file

******************************************************************************
*                                                                        *
*       FILE FORMAT DESCRIPTION                                           *
*                                                                        *
*         <<<<  NEMESH  >>>>                                           *
*                                                                        *
*      Last updated: 06/18/2013                                         *
*        nera-software@anl.gov                                        *
*                                                                        *
******************************************************************************

*** CARD TYPE DIRECTORY ***

<table>
<thead>
<tr>
<th>CARD CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00 HEADER LINES</td>
<td></td>
</tr>
<tr>
<td>01 CONTROL FLAGS</td>
<td></td>
</tr>
<tr>
<td>02 BASIC MESH INFO</td>
<td></td>
</tr>
<tr>
<td>03 ELEMENT LIST</td>
<td></td>
</tr>
<tr>
<td>04 ELEMENT CONNECTIVITY</td>
<td></td>
</tr>
<tr>
<td>05 VERTEX POSITION DATA</td>
<td></td>
</tr>
<tr>
<td>06 BOUNDARY SURFACE LIST</td>
<td></td>
</tr>
<tr>
<td>07 BLANK LINE</td>
<td></td>
</tr>
</tbody>
</table>

*** CARD TYPE DESCRIPTIONS ***

======================================
CARD TYPE 00: HEADER LINES
======================================
LINE NUMBERS: 1-9
FORMAT: N/A
DESCRIPTION:
The first nine lines of the input file must be blank or commented. Do not place any other card data in lines 1-9 of the input file.

======================================
CARD TYPE 01: CONTROL FLAGS
=======================================
LINE NUMBER: 10
FORMAT: <Input Style Flag> <Debug Print Flag>
DESCRIPTION:
<Input Style Flag>: 0....Indexed Input (numbered elements and nodes)
1....Not Indexed
<Debug Print Flag>: Integer....0-10 (0=no printing, 10=full debug printing)

======================================
CARD TYPE 02: BASIC MESH INFO
======================================
LINE NUMBER: 11
FORMAT: <NumElements> <NumNodes> <unused integer> <NumBoundarySurfaces>
DESCRIPTION:
<NumElements> : Integer....Number of elements in the mesh
<NumNodes> : Integer....Number of nodes (vertices) in the mesh
<unused integer> : 0........(reserved for future use)
<NumBoundarySurfaces> : Integer....Number of boundary surfaces in the mesh

======================================
CARD TYPE 03: ELEMENT LIST
=======================================
LINE NUMBERS: 12+, with a total of NumElements lines

FORMAT: { [Optional Index] <ElementType> <MaterialBlock> }; repeat for each element specified in <NumElements>

DESCRIPTION:
[Optional Index] : Integer.... Optional index of the element

<ElementType> : Integer from the following list
1....1-D Bar             Linear
2....1-D Bar             Quadratic
5....2-D Triangular      Linear
6....2-D Triangular      Quadratic
10....2-D Quadrilateral  Linear
11....2-D Quadrilateral  Quadratic
15....3-D Tetrahedron    Linear
16....3-D Tetrahedron    Quadratic
20....3-D Tri. Prismatic Linear
21....3-D Tri. Prismatic Quadratic
25....3-D Hexadron       Linear
26....3-D Hexadron       Quadratic
31-39....1-D Bar             Lagrangian (31 = linear, 39 = 9th order)
41-49....2-D Triangular      Lagrangian (41 = linear, 49 = 9th order)
51-59....2-D Quadrilateral   Lagrangian (51 = linear, 59 = 9th order)
61-69....3-D Tetrahedron    Lagrangian (61 = linear, 69 = 9th order)
71-79....3-D Tri. Prismatic Lagrangian (71 = linear, 79 = 9th order)
81-89....3-D Hexadron       Lagrangian (81 = linear, 89 = 9th order)
91-99....2-D Quadrilateral   NonConforming (Lagrangian setup)
101-109....3-D Hexadron       NonConforming (Lagrangian setup)

<MaterialBlock> : Integer.... Material block for this element

======================================
CARD TYPE 04: ELEMENT CONNECTIVITY
=======================================
LINE NUMBERS: Immediately following last CARD TYPE 03, with a total of NumElements lines

FORMAT: [ [Optional Index] <ConnectivityArray> ]; repeat for each element specified in <NumElements>

DESCRIPTION:
[Optional Index] : Integer.... Optional index of the element

<ConnectivityArray> : Array of integers read as CONNECTIVITY(J),J=1,ELEMENTVERTICES
This array lists the indexed vertices which define the element. ELEMENTVERTICES is determined automatically from ElementType. For example ElementType 1 (1-D Bar Linear) has 2 vertices.

======================================
CARD TYPE 05: VERTEX POSITION DATA
=======================================
LINE NUMBERS: Immediately following last CARD TYPE 04, with a total of NumNodes lines
FORMAT: { [Optional Index] <X> [Y] [Z] }; 
repeat for each node specified in <NumNodes>

DESCRIPTION:
[Optional Index] : Integer.... Optional index of the node
<X>        : Real....X-value of this node (always present)
[Y]        : Real....Y-value of this node (present only for 2D and 3D elements)
[Z]        : Real....Z-value of this node (present only for 3D elements)

======================================
CARD TYPE 06: BOUNDARY SURFACE LIST
======================================
LINE NUMBERS: Immediately following last CARD TYPE 05, with a total of NumBoundarySurfaces lines

FORMAT: { [Optional Index] <ElementIndex> <ReferenceSurface> <BoundaryConditionFlag> }; 
repeat for each node specified in <NumNodes>

DESCRIPTION:
[Optional Index]        : Integer....Optional index of the boundary surface
<ElementIndex>          : Integer....Index of the element with the boundary surface
<ReferenceSurface>      : Integer....Reference element surface number corresponding to the boundary surface
<BoundaryConditionFlag> :   1..... Reflected
Other....Void

======================================
CARD TYPE 07: BLANK LINE
======================================
LINE NUMBER: Last line in the file

FORMAT: blank line

DESCRIPTION:
A blank line must be placed at the end of the line in order to avoid "end of file" error upon reading.

*************************************************************
*    END OF FILE FORMAT DESCRIPTION   *
*          <<<< NEMESH >>>            *
*************************************************************

C.2 Example “nemesh” file

! ANL FINITE ELEMENT INPUT - NEMESH FILE DESCRIPTION - 9 HEADER LINES ALWAYS
! This example was taken from bench05 of the benchmarks directory
! Header line 3
! Header line 4
! Header line 5
! Header line 6
! Header line 7
! Header line 8
! Header line 9
0 0                     ! CARD TYPE 1   control info - indexed input with no debug printing
16 45 0 12              ! CARD TYPE 2   mesh info    - 16 elements, 45 nodes, 12 boundary surfs
1 6 1                  ! CARD TYPE 3   element info - first element is type 6 with material 1
2 6 1
3 6 1
4 6 1
5 6 1
6 6 1
7 6 1
8 6 1
9 6 1
10    6    7
11    6    7
12    6    7
13    6    7
14    6    7
15    6    7
16    6    7

1    1    2    3    11    17    10   ! CARD TYPE 4 connectivity: element 1 is connected to
2    3    4    5    13    19    12   ! vertices 1,2,3,11,17,10
3    17    11    3    12    19    18
4    17    18    19    24    13    19
5    5    6    7    15    21    14
6    5    14    21    20    19    13
7    19    20    21    26    25    30
8    28    24    19    25    30    29
9    28    29    30    34    37    33
10    7    8    9    16    21    15
11    9    22    32    27    21    16
12    21    27    32    31    30    26
13    32    36    39    35    30    31
14    39    35    38    37    34
15    37    38    39    41    42    40
16    39    43    45    44    42    41
1    0    0   ! CARD TYPE 5 node info: node 1 at (0.0,0.0)
2    0.10237500000000001    0
3    0.20475    0
4    0.307125    0
5    0.4094999999999999    0
6    0.5400026370274923    0
7    0.5850013185137463    0
8    0.63    0
9    0.63    0
10    0.07239005672397281    0.07239005672397281
11    0.1891643342816859    0.07835443277675214
12    0.2915393342816866    0.07835443277675214
13    0.4016315723251228    0.0798894868656449
14    0.4655512466645283    0.06988148343414935
15    0.5353828396092143    0.07048448799998519
16    0.5758012466645285    0.06988148343414935
17    0.1447801134479456    0.1447801134479455
18    0.2615543910056587    0.1507444895007249
19    0.3783286685633719    0.1567088655535044
20    0.44965580866662141    0.1482359162109015
21    0.5216024933690565    0.1397629668682987
22    0.63    0.16500000000000001
23    0.2171701701719185    0.2171701701719183
24    0.34048680623798923    0.2275060017763959
25    0.4229923351698838    0.2133550923362522
26    0.4988973838518217    0.2066500626238805
27    0.5758012466645285    0.1748814834341492
28    0.2895082268958912    0.2895082268958912
29    0.3786081143361436    0.2797807727048186
30    0.4675650017763959    0.2700011385137462
31    0.5488280008881977    0.2400065925687311
32    0.63    0.21000000000000001
33    0.3356998766983245    0.3356998766983244
34    0.4284128958518366    0.3287327769853338
35    0.5488280008881977    0.3450065925687311
36    0.63    0.31500000000000001
37    0.3818395265007578    0.3818395265007578
38    0.5059197632503787    0.4009197632503797
39    0.63    0.4199999999999998
40    0.4438796448755683    0.4438796448755683
41    0.5679598816251895    0.4629598816251895
42    0.5859197632503788    0.5859197632503788
43    0.63    0.5249999999999998
44    0.5679598816251895    0.5679598816251895
45    0.63    0.63

1    1    1    1   ! CARD TYPE 6 boundary data: the first boundary surface is on ele 1, ref. surf 1, refl bc
2 1 3 1
3 2 1 1
4 4 3 1
5 5 1 1
6 9 3 1
7 10 1 1
8 11 1 1
9 13 1 1
10 15 3 1
11 16 1 1
12 16 2 1

! MAKE CERTAIN THERE IS AN EXTRA BLANK LINE AFTER ALL OF THE INPUT TO PREVENT EOF ERROR