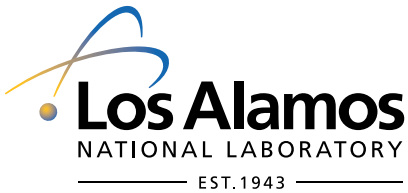
LA-UR-17-22442

Approved for public release; distribution is unlimited.

Title:	The MCNP6 Book On Unstructured Mesh Geometry: User's Guide For MCNP 6.2
Author(s):	Roger L. Martz
Intended For:	General Reference / MCNP® Website
Issued:	March 2017



This page intentionally left blank.

The MCNP6 Book On Unstructured Mesh Geometry: User's Guide for MCNP 6.2

Roger L. Martz

March 2017

This page intentionally left blank.

This page intentionally left blank.

Contents

List of Figures

This page intentionally left blank.

List of Tables

This page intentionally left blank.

IN MEMORIAM

David L. Crane, Ph.D.

1963 - 2016

In January 2016 David lost his battle with cancer. We were deeply honored to have him as a colleague and friend. His finite element contributions to the unstructured mesh library in MCNP6 and technical discussions and guidance have been invaluable. He is sorely missed by all who knew him.

This page intentionally left blank.

Chapter 1

Overview

Los Alamos National Laboratory's (LANL) Monte Carlo N-Particle® (MCNP®) transport code has a more general geometry capability than has been available in most combinatorial geometry codes [1]. In addition to the capability of combining several predefined geometric bodies, as in a combinatorial scheme, MCNP6 [2] gives the user the added flexibility of defining geometric regions from all the first and second degree surfaces of analytical geometry and elliptical tori and then combining them with Boolean operators. This decades-old constructive solid geometry (CSG) capability has been well-tested and verified. However, it has long been recognized that as the model complexity increases, this process of describing the geometry is difficult, tedious, time-consuming, and error prone[3, 4, 5]. Consequently, innovators have taken on the task of developing a better way to construct geometries, not only in MCNP6, but other particle transport codes as well.

MCNP6 addresses this issue and the issue of multi-physics integration by permitting the embedding of an unstructured mesh (UM) representation of a geometry in its legacy CSG to create a hybrid geometry. The UM is contained within a MCNP6 universe where it must not be clipped by the fill cell into which it is placed and no other universe or cell may be contained within it. MCNP6 also allows multiple instances of an UM universe and multiple UM universes.

The UM capability was originally designed to work with an unstructured mesh created with the Abaqus/CAE [6] tool and the ASCII input file that it generates. An overview of this input file is given in Chapter 9. Many other CAE tools have the ability to generate a mesh from a solid model that can be easily converted to the Abaqus format. It is the user's responsibility to verify that these third-party tools are generating the Abaqus format that meets the MCNP6 specification; see Chapter 9 for what MCNP6 expects. In addition, the information in the Abaqus input file can be converted to the MCNP6-friendly `MCNPUM` file type.

The original intent at the beginning of this work was for this UM capability to be implemented as a modular mesh-tracking library written in Fortran 90/95. Actual code methods and implementation details are not discussed in this work, but are the subjects of other documents. The reader will, from time to time, run across in the MCNP6 documentation the term REGL which stands for Revised Extended Grid Library; this is the UM library and the "regl" tag is heavily used in the actual coding and documentation.

This page intentionally left blank.

Chapter 2

Terminology

One of the problems of merging two capabilities that have long, independent development paths is dealing with the distinct and sometimes contradictory terminology that has evolved with each. For example, the term "cell" is often used to generically denote the smallest building block in a geometry. However, a MCNP6 cell is quite different from an UM cell that will be referred to as a "finite element". Therefore, great care is exercised in giving definitions as can be seen with the following.

elements or finite elements: The smallest building blocks into which the mesh geometry is broken. These are unstructured polyhedrons with 4, 5, and 6 sides or faces, Figure 2.1. First-order elements have nodes only at the vertices. When a face has 4 nodes, all 4 nodes are not guaranteed to lie in the same plane. This face has a degree of curvature and is known as bilinear. Thus, first-order elements may have either planar or bilinear faces. First-order elements with bilinear faces have trilinear volumes.

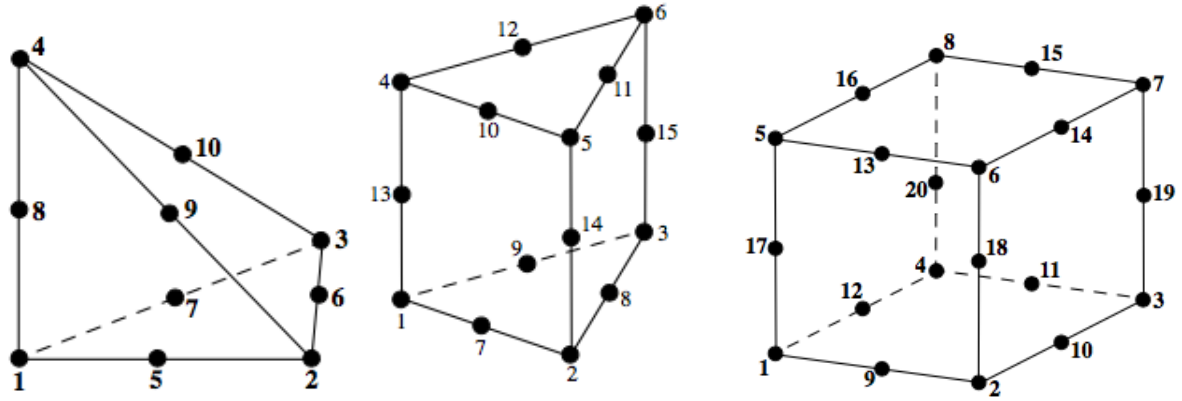
Second-order elements have nodes at the vertices and at the midpoints between the vertices. When 4 or more nodes define a face, they are not guaranteed to lie in the same plane. With 6 or 8 nodes defining a face, the degree of curvature can be greater than with 4 nodes and the faces are known as biquadratic. Thus, second-order elements may have either planar, bilinear, or quadratic faces. Second-order elements with quadratic faces have triquadratic volumes.

mesh: The collection of elements comprising the entire model. The mesh is a representation of the geometry described by the solid model in the CAE tool.

elsets: Elsets is short for element sets. An elset is a collection of elements and has associated with it a specific tag, label, or name.

part: This is the smallest geometric object created in the Abaqus/CAE tool. In a CAE tool such as Cubit [7], this would be a block. While the part is the smallest object which can be meshed in Abaqus, it is possible to further sub-divide a part into sections. Each section can be assigned a different material and will become a pseudo-cell (see below).

instance: An instance is a copy of a part used in constructing an assembly. Thus, a simple part may be used multiple times, giving rise to multiple instances of that part.



(a) Tetrahedron (b) Pentahedron (c) Hexahedron

Figure 2.1: Finite element type.

assembly: An assembly is the largest geometric object created in the Abaqus/CAE tool. It may consist of an instance of one part to many instances of many parts. It can be viewed as a composite object. From this assembly, MCNP6 will create a global mesh model.

pseudo-cell: In the UM library this refers to an elset with a distinct material and statistical set definition. Internally to the code, all instances are broken up into pseudo-cells; particle tracking takes place on the pseudo-cells. In MCNP6 this is a special cell definition, defined with a null or zero surface, that is used to associate normal MCNP6 cell features with the UM elset (e.g., cells for F4 tallies).

background cell: A cell that serves as the background medium for the UM. This is an MCNP6 cell into which the mesh has been place. If one were to ignore the mesh representation in the mesh universe, this would be the sole cell that describes the mesh universe.

mesh universe: This is the MCNP6 universe composed of the UM and the background cell. This universe may not contain any other lower universes or cells. The UM must not be clipped by the boundaries of the fill cell that define this universe. This clipping requirement is not enforced by the code at this point, but is the user's responsibility to ensure that it doesn't occur. If clipping does occur, the user will experience lost particles is these regions of phase space.

Chapter 3

Constructing A Mesh Geometry

The first step in creating an unstructured mesh model for use in MCNP6 is to create a part or series of parts with the CAE tool. Each part can consist of a single segment of one homogeneous material, or if Abaqus/CAE is used, multiple segments of different homogeneous materials. Once each part is created, material, statistic, and/or source elsets must be created for it (details given below). After each part is meshed independently in Abaqus/CAE, they are combined to form an assembly, Figure 3-1. The final step is to define material names.

Other CAD/CAE tools may promote a different workflow, but ultimately must meet the requirements present in this chapter.

(a) parts (b) assembly

Figure 3.1: Constructing an assembly from parts.

3.1 Naming elsets and materials

Each elset in the Abaqus inp file must be tagged with a name. The UM input parser requires the elset name to be in a specific format as shown next.

```
????AAAA????_ZZZ
```

where `AAAA` is one of the keywords:

```
material, statistic, tally, source
```

`ZZZ` is the set number following an underscore, '_', or an hyphen, "-", and must be at the end of the character string. `ZZZ` can be from 1 to 12 digits in length.

`????` are any other character or groups of characters, but should NOT be a duplication of any of the keywords. In addition, none of these keywords should appear in the part name.

For any `AAAA`, the `ZZZ` number must be unique within the part if separate elsets are desired in the part. The `ZZZ` number must be unique within the assembly for the material elsets and material names in order for the `EEOUT` file to be fully functional with auxiliary programs such as GMV [8]. It is this material number that is assigned internally to the elements by the UM library and is output in the `EEOUT` file (see Chapter 10) for each element. For best results, make `ZZZ` one of the same numbers that appear on the MCNP6 material cards.

As a convenience, it is possible to construct one elset that has multiple functions by specifying more than one keyword in the elset name. The suggested format for this name is given as

```
???AAAA???%BBBB%CCCC???%ZZZ
```

where `AAAA`, `BBBB`, and `CCCC` are the keywords defined above; there should at least be one keyword present. The elset number, `ZZZ`, must be an appropriate number for each function or keyword. MCNP6 will use this info to internally construct the appropriate elsets for its use. If more than one statistic elset is needed in the part, this format should not be used. % indicates that either an underscore, '_', or an hyphen, "-", be present.

In Abaqus/CAE material names are independently created and are placed near the end of the Abaqus inp file. It is highly recommended that the material names adhere to the following format

```
????????_ZZZ
```

where `ZZZ` is a material number that corresponds to those numbers used in the material elsets and `????????` are any other character or groups of characters, except the keywords mentioned above. In other words, make `ZZZ` a valid MCNP6 material number.

Material names appear in the pseudo-cell cross reference table that is written to the MCNP6 output file after the REGL processes the mesh description and creates the global tracking model. This table is intended to help users understand how the pseudo-cells should be specified. When searching for material names to insert into this table, the code tries to match the material number for the pseudo-cell to the material number in the material name. If that fails, the code assumes that the material names have been entered sequentially from 1 to the maximum number of material numbers and uses the pseudo-cell material number to select one of these. If both of these rules fail to produce a defined name, a message is inserted into the table to the effect that the material name does not exist.

Abaqus/CAE permits assignment of material properties with the material name. The physical density or number density may be entered here using the MCNP6 convention. This information is added to the .inp file and is used when *um_pre_op* generates a MCNP6 input deck; see Chapter 14.

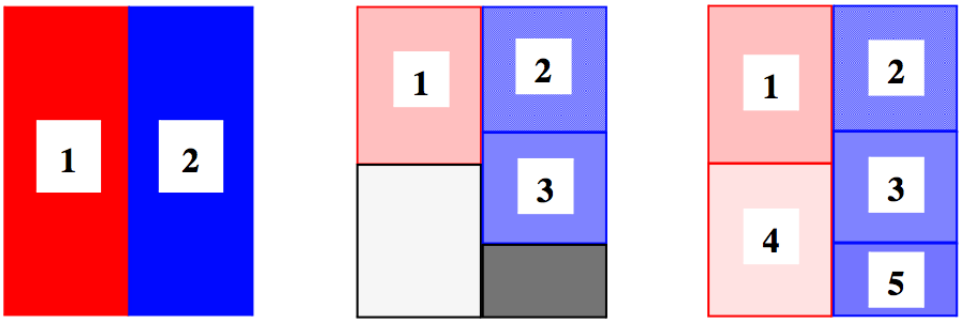
Of the four AAAA keywords, `material` is required and `statistic`, `tally`, and `source` are optional. The keywords `statistic` or `tally` are highly recommended and are interchangeable; use one or the other, but not both to describe the elset. Initially, the intent was to collect individual elements into a group for the purpose of volume tallies (i.e., F4, F6, or F7); hence, MCNP6's full statistical treatment for tallies (e.g., tally fluctuation chart, empirical history score PDF) could be extended to the elsets. Likewise, this group can be used for other features such as the `IMP` variance reduction game. Basically, this group exhibits cell-like features; hence, coining of the term pseudo-cell. It should be recognized that MCNP6 requires its cells to be

associated with only one material and this must be upheld through the pseudo-cells. Therefore, the UM library checks the material and statistic elset requests to ensure this requirement.

The `source` keyword should only be used to describe a volume source region in the unstructured mesh. MCNP6 will sample the source starting position (x, y, z) uniformly over the elements associated with the volume source; multiple volume sources are permitted, but see Section 5.3 on how to select among various volume sources. That is, a source element is selected from the source elset(s) with a probability proportional to the fractional volume of the source element in the total source volume. The source coordinates (x, y, z) are uniformly selected, by a rejection technique, over the selected element. No source biasing of position within a source elset (or pseudo-cell) is permitted with this capability. All other, non-positional fixed source (`SDEF`) options should work in conjunction with this capability, but extensive testing has not be performed. Volume sources may be defined but will not be used unless requested on the `SDEF` card.

3.2 Pseudo-Cell Creation

With detailed models and partitioned parts, it is conceivable that the user may overlook some elements when collecting them for the material and statistic elsets. Therefore, the UM library checks every element to ensure that there are material and statistic numbers assigned. If an element isn't assigned a material number, the only thing the UM library can do is generate an error message and stop. The UM library collects all elements of a part that haven't been assigned a statistical set number and lumps them into a catch-all set. This may or may not produce the desired effect wanted by the user. The user is highly encouraged to define all elsets in the CAE tool. After all elements in a part are checked for material and statistic set numbers, the UM library uses the material and statistical set information to define the pseudo-cells from which a cross-reference table is printed in the output. The practitioner can use this table to double check the pseudo-cell definitions in the MCNP6 input.



(a) 1 part; 2 materials

(b) 3 defined; 2 undefined statistical regions

(c) 5 pseudo-cells

Figure 3.2: Pseudo-cell example.

Consider the example shown in Figure 3.2. The user creates a part, Figure 3.2a, with two materials. That is, there are two material elsets: red (1) and blue (2). The user then creates a statistical elset (1) in the upper portion of the red material and two statistical elsets (2) and (3) in the blue material, Figure 3.2b. Notice that the lower portions of the part (white / black

regions in Figure 3.2b) are not assigned to any statistical elsets. Since it has been verified by the UM library that each of the statistical elsets (1), (2), and (3) consist of only one material, they meet the requirements for a pseudo-cell; hence, Figure 3.2c shows them as pseudo-cells (1), (2), and (3). Pseudo-cells (4) and (5) in Figure 3.2c arise from the catch-all statistical elsets in the red and blue materials, respectively. In this example, the user explicitly defined three pseudo-cells with material and statistic elset numbers and the UM library implicitly defined two more pseudo-cells from the material and catch-all elsets.

From the UM library's perspective, pseudo-cells are numbered consecutively starting at 1 in the order the parts are instanced into the problem. If part #2 is instanced ahead of part #1 in the Abaqus input file, pseudo-cell #1 is associated with the first instance of part #2. If part #2 is instanced a second time later in the input, it will generate a different set of pseudo-cell numbers.

3.3 Mesh Universe

A simplified MCNP6 hybrid geometry arrangement with an unstructured mesh embedded in the CSG geometry (i.e., mesh universe) is shown in Figure 3.3. The mesh universe is everything contained within the fill cell where the fill cell's outer boundary is the heavy black rectangle. Note, "fill cell" means the traditional MCNP6 cell card that contains the "fill" parameter and a collection of defined surfaces that crops the universe which it contains. These surfaces must not crop the mesh!

A background cell is needed to make the mesh universe infinite in extent and is the region outside of the blue unstructured mesh region in Figure 3.3; it is cropped by the surface that defines the fill cell. Specifying the background cell in the MCNP6 input is a 2-step process. First, a pseudo-cell must be specified in the cell block and the background keyword must appear on the embed data card; more detail on these input cards is given in Chapter 5. The material specified for the background cell is also the material used in all gaps within the UM.

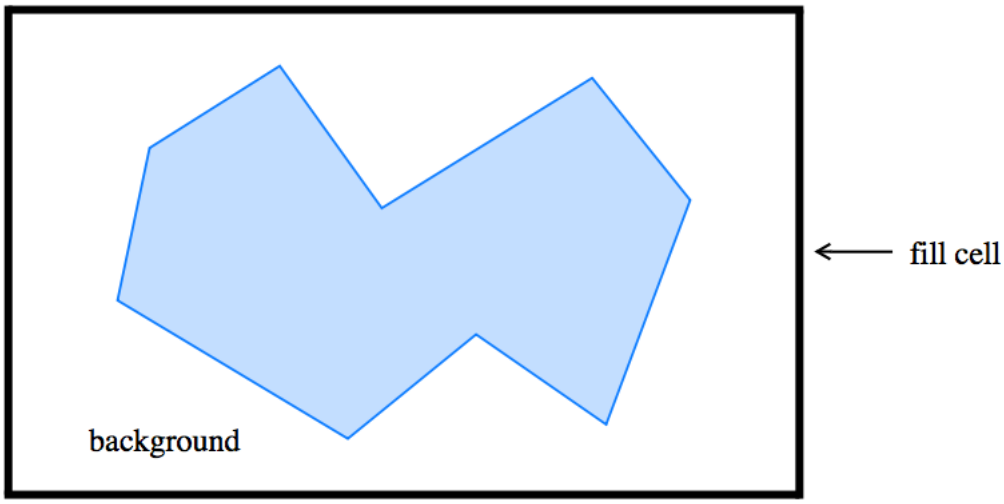


Figure 3.3: Example mesh universe with unstructured mesh.

3.4 Overlaps

One of the initial requirements for the unstructured mesh implementation in MCNP6 was to permit multiple, non-contiguous, meshed parts instead of requiring one contiguous mesh. This naturally leads to the possibility of overlapping parts, particularly when two parts attempt to share a curved surface. If it is crucial to the model that the integrity of any curved surface be maintained, the user should then consider, using Abaqus/CAE terminology, merging the two separate parts into a single part, try second-order elements, or try refining the mesh. Significant overlapping regions are never a good idea. Users should never rely on any of the following models to correctly produce the same results as a model where the boundary between two regions is definitely defined so that there is no overlap.

The program can accommodate a small amount of overlap in one of several ways. For the initial implementation there was no correction for tracking through overlapping elements. A particle tracks in an element until it finds a definite transition point in phase space (i.e., another element, gap, or background cell). Of the three overlap models currently in place (see the `OVERLAP` keyword on the `EMBED` card and Figure 3.4 below), the initial implementation is known as the `EXIT` model, meaning that in an overlap situation the exit point of the overlap is used and results are accumulated accordingly.

The second overlap model, `ENTRY`, is the one that uses the entry point of the overlap in an overlap situation and the results are accumulated accordingly. If the ENTRY point is behind the particle's current position, the current position is used; the particle never moves backwards. The third and last overlap model is called `AVERAGE` and results in averaging the entry and exit points in an attempt to find the midpoint of the overlap in the direction the particle is tracking; the particle's path length in the overlap is then divided between the two parts instead of being assigned to one or the other.

Although the code defaults to the `EXIT` model, ultimately the choice of which model to use is left to the user. If both parts are important and the particle flux through this region is fairly isotropic, the `AVERAGE` model is probably the best choice. If the flux is somewhat more directional and one part is deemed more significant than the other, a better choice might be `ENTRY` or `EXIT`; the user must decide. The user also has the ability to select the model to use by the instance/part (i.e., pseudo-cell) with the decision based upon the current instance/part in which the particle resides. For example, if the particle is currently in a part that specifies the `EXIT` model and the part into which it will travel specifies the `ENTRY` model, the `EXIT` model is used.

Note that extensive testing has been performed with the `EXIT` model but not the other two.

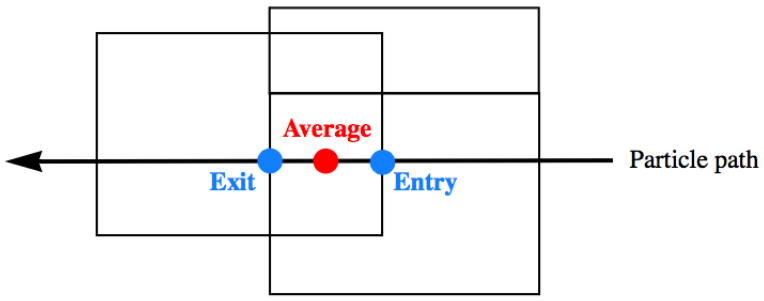


Figure 3.4: Illustration of the three critical points for the overlap models.

Chapter 4

Output: Elemental Edits

To obtain results at the element level, a path length estimate of the flux is accumulated as particles track from one element face to another, Figure 4.1.

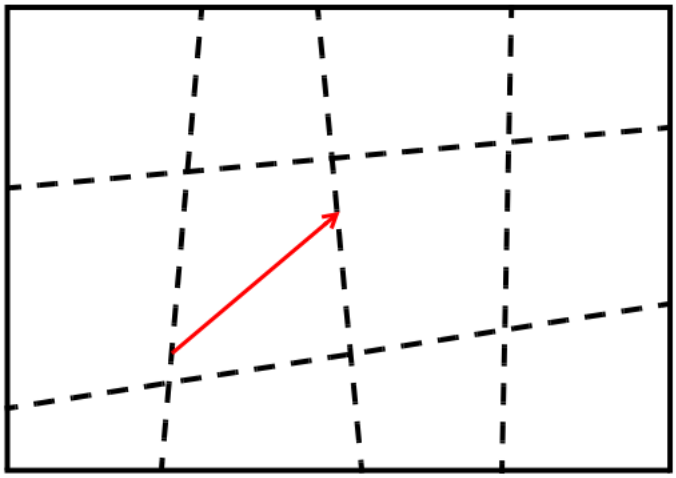


Figure 4.1: Illustration of element-to-element tracking on a 12-element part.

To differentiate the mesh results from the traditional MCNP6 tally treatment, those results accumulated on the unstructured mesh are referred to as elemental edits. There is no current intention to duplicate all of the tally features with the edits. The elemental edits, along with a generic description of the unstructured mesh model, are output in a special file (see embee card) known as the `EEOUT` (Elemental Edit OUTput) file. See Chapter 10 for a description of this file. See Chapter 13 for a discussion of the *um_post_op* utility program for manipulating this file.

At this time, relative errors are optional for the results on any element. Specifying errors can result in large `EEOUT` files. If the traditional MCNP6 statistical analysis (e.g., tally fluctuation chart, empirical history score pdf) is desired for the results, set up a tally for an appropriate pseudo-cell.

As stated in Reference 1 (p 2-114), relative errors have a range of zero to unity (inclusive) when the results are of the same sign.

This page intentionally left blank.

Chapter 5

Input Cards

5.1 Cell Cards

The pseudo-cells are defined in the MCNP6 cell block when a null surface appears in the surface specification. The cell card format for these cells have the following properties that differentiate them from the typical MCNP6 cells.

- have a single null-surface entry for the surface description (i.e., 0)
- are assigned to a universe (e.g., `u=10`)
- the universe number matches the `N` specified on the `embedN` card with which it is associated
- can not be filled by another universe or lattice (i.e., no `fill=` or `lat=` entry)

5.2 Data Cards

There are eight new data section cards that can be used with the unstructured mesh; one is required and the other seven are optional. Also, there is a new option for the `POS` parameter on the `SDEF` card.

5.2.1 EMBED

`EMBED:<p> keyword = value ...`

This is the embedded geometry specification. It is valid in a continuation run.

`n` embedded mesh universe number; must match a universe number specified on the pseudo-cell cards.

Input on this card follows the keyword value format: key = value(s).

Required Keys:

BACKGROUND	pseudo-cell number from the cell block that serves as the background cell
MATCELL	m_1 c_1 m_2 c_2 m_3 c_3 ... m_i c_i integer pairs, one for each pseudo-cell in the embedded mesh; m_i values are embedded mesh pseudo-cell numbers; numbers must be sequential starting at 1 c_i values are the MCNP6 pseudo-cell numbers from the cell block
MESHGEO	mesh type; acceptable values: abaqus, mcnpum
MGEOIN	name of input file containing the mesh description
MEEOUT	name of the EEOUT results file to write

Optional Keys:

FILETYPE	type of file for MEEOUT to write; acceptable entries: ascii or binary; default: ascii
GMVFILE	file name for GMV output file; geometry only description for GMV program[8]; LANL use
LENGTH	a multiplicative conversion factor to centimeters for all mesh dimensions in the input and output files; default: 1
MCNPUMFILE	file name for the MCNPUM output file
MEEIN	name of the EEOUT results file to read; required for a continuation run. Must not be the same as MEEOUT.
OVERLAP	model to treat overlapping parts. First entry should be one of the following: EXIT (default when OVERLAP is not provided), ENTRY, AVERAGE. Treatments for individual pseudo-cells can be specified by following the initial entry with a second parameter and a list of valid pseudo-cell numbers (from the MATCELL entry). All 3 parameters may be used if the format is correct. See the example in Section 5.4.

5.2.2 EMBEE

EMBEE:<p> keyword = value ...

Embedded elemental edits control card.

n	elemental edit number ending in 4, 6, or 7; follows the tally convention for meaning; if this card is not present, a total flux edit is created for each particle on the mode card.
<p>	particle designator; current valid entries: n, p, h, or any of the valid charged particles.

Input on this card follows the keyword value format: key=value(s)

Required Keys:

`EMBED`	embedded mesh universe number; must correspond to a valid embed card or mesh universe number

Optional Keys:

`COMMENT`	edit comment to appear in the `EEOUT` file; limited to 128 characters. Same functionality as `FC` card for tallies except that this is for the elemental edits.
`ENERGY`	a multiplicative conversion factor from MeV/g for all energy related output; default: 1
`ERRORS`	request statistical uncertainties; NO (default) / YES.
`TIME`	a multiplicative conversion factor from shakes for all time related output; default: 1

Additional parameters added to support flux multipliers on the edits.

`ATOM` flag to multiply by atom density; NO (default) / YES.

`FACTOR` multiplicative constant; default: 1.0; equivalent in concept to `|C|` on the `FM` card.

`LIST` reaction list where this is the sum and/or product of ENDF or special reaction numbers. Limited to 1 reaction list as with `FMESH` tallies. Parentheses can be used but are ignored by the code.

`MAT` material number identified on an `Mn` card. Can be a dummy material or 0 (default). If the value is 0, use the cell material.

`MTYPE` Multiplier type. Acceptable character input values follow:

- `flux` normal volume flux calculations. Same interpretation as `FMESH` tally type = flux. (default)
- `isotopic` isotopic calculation. UM equivalent to the `FMESH` isotopic mesh tallies that require an `+FM` card [3].
- `population` population calculation. Same as an `F4` tally with an `FM` card where `k = -2` in the multiplier set.
- `reaction` reaction calculation that requires the `LIST` parameter. This `mtype` with the `LIST` parameters is equivalent to an `FMESH` tally with a single multiplier set specified and its accompanying `FM` card.
- `source` accumulate source point locations. Same interpretation as `FMESH` tally type = source.
- `tracks` tracks calculation. Same as an `F4` tally with an `FM` card where `k = -1` in the multiplier set.

5.2.3 EMBEB

`EMBEBn` B_1 B_2 ... B_k

Embedded elemental edit energy bin boundaries.

`n` elemental edit number from `embee` card; 0 is not valid.

B_i energy of the i'th bin; monotonically increasing upper energy bin boundaries; values in units of MeV; default: one energy bin with boundary set to the maximum energy limit for the particle type.

5.2.4 EMBEM

`EMBEMn` M_1 M_2 ... M_k

Embedded elemental edit energy bin multipliers.

`n` elemental edit number from `embee` card; 0 is not valid.

M_i multiplier for the i'th energy bin; default: 1.

5.2.5 EMBTB

`EMBTBn` B_1 B_2 ... B_k
Embedded elemental edit time bin boundaries.

`n` elemental edit number from `embee` card; 0 is not valid.

B_i time of the i'th bin; monotonically increasing upper time bin boundaries; values in units of shakes (1 shake = 10^{-8} s); default: one time bin with boundary set to the maximum time limit for the particle type.

5.2.6 EMBTM

`EMBTMn` M_1 M_2 ... M_k
Embedded elemental edit time bin multipliers.

`n` elemental edit number from `embee` card; 0 is not valid.

M_i multiplier for the i'th time bin; default: 1.

5.2.7 EMBDE / EMBDF

In order to avoid confusion and maintain the separability of edits from tallies, the following response function cards are available to implement response functions on the unstructured mesh edits. These are similar to the standard `de/df` cards; there are no built in functions associated with these cards at this time.

`EMBDEn` B_1 B_2 ... B_k
Embedded elemental edit dose energy bin boundaries.

`n` elemental edit number from `embee` card; 0 is not valid.

B_i energy of the i'th bin; monotonically increasing upper energy bin boundaries; values in units of MeV; default: one energy bin with boundary set to the maximum energy limit for the particle type.

`EMBDFn` M_1 M_2 ... M_k
Embedded elemental edit dose function multipliers.

`n` elemental edit number from `embee` card; 0 is not valid.

M_i multiplier for the i'th time bin; default: 1.

5.2.8 SDEF VOLUMER

SDEF general fixed source specification – addendum.

POS x, y, z vector/reference point for position sampling; default {0,0,0}. Use value *volumer* for unstructured mesh volume source(s) so that x, y, z may be sampled from the volume source description. This value may also be used with dependent distributions. See Section 5.3 for more discussion on volume sources and how they may be selected. Note that the last character 'r' stands for sampling by rejection.

5.3 Volume Sources

See the source keyword discussion in Section 3.1 for more information about describing volume sources in the unstructured mesh. This section will describe how the user can select among multiple volume sources (pseudo-cells) defined in the unstructured mesh.

First, if volume sources have been defined in the mesh and you do not wish to sample from them, don't use the VOLUMER value anywhere in describing the source on the SDEF card.

Second, if you want to sample uniformly over all volume source regions defined in a model, simply set the POS parameter to VOLUMER.

```
Example: sdef pos=volumer
```

Next, if the volume sources appear in different pseudo-cells and you desire to sample non-uniformly among the pseudo-cells, use a dependent distribution where POS is a function of CEL. Only uniform sampling within a cell is possible.

```
Example:
sdef pos=fcel=d1 cel=d2
ds1 L volumer volumer
si2 L 101 103
sp2 0.4 0.6
```

In this example, MCNP6 will first select proportionally from cells 101 (40%) and 103 (60%). With the cell selected, the code will select uniformly over that cell proportional to each element's volume to find an element from which it will select a position uniformly over that element.

Finally, it is possible to combine volume sources with point sources (and other legacy source descriptions) with a dependent distribution of distributions.

```
Example: 2 volume sources and a point source
sdef pos=fcel=d3 cel=d2
c
si2 L 101 102 103
sp2 0.4 0.2 0.4
c
ds3 S 5 4 6
c
si4 L .1 .2 .3
```

```
sp4 1
c
si5 L volumer
sp5 1
c
si6 L volumer
sp6 1
```

As before, the cell is selected first, then the position from the appropriate distribution. In this example, the point source is selected 20% of the time.

5.4 Initial Run Example

```
C Cell Cards
10  1  -2.03  0    u=2   $ pseudo-cell
11  1  -2.03  0    u=2   $ pseudo-cell
12  1  -2.03  0    u=2   $ pseudo-cell
13  1  -2.03  0    u=2   $ pseudo-cell
14  1  -2.03  0    u=2   $ pseudo-cell
15  1  -2.03  0    u=2   $ pseudo-cell
21  0         0    u=2   $ background cell
30  0       -99 fill=2   $ fill cell
40  0        99
c Surface Cards
99 sph 0.0.3.10.
c Data Cards
m1 1001 -0.02 8016 -0.60 1400 -0.38
c
embed2 meshgeo= abaqus
       meeout= sample01.eeout
       gmvfile= sample01.gmv
       filetype= binary
       background= 21
       matcell= 1 10  2 11  3 12  4 13  5 14  6 15
       overlap=average exit 1 entry 5 6
c
embee4:n embed=2
embtb4 1 2 3 4 5 1e+39
embeb4 0.1 1.0 1e+10
```

5.5 Continue Run Example

MCNP6 continue runs with the unstructured mesh feature require an input `EEOUT` file in addition to the *runtpe* file.

```
CONTINUE
```

```
C
embed2 meshgeo= abaqus
       meein= sample01.eeout
       meeout= sample01.cont.eeout
       background= 21
       matcell=110 211 312 413 514 615
```

Chapter 6

Parallel Input Execution

MCNP6 has been able to transport particles on the UM in parallel (threads, MPI, or both) for some time. Until September 2011, MCNP6 input processing was not parallelized. However, unstructured mesh input can take a long time to process if there are multiple parts and at least one of the parts has more than roughly 30,000 to 50,000 elements. Therefore, sections of the unstructured mesh input processing have been parallelized with MPI in order to speed up the overall process. In addition, some processing loops have been threaded with OpenMP.

Some of the mesh data is organized at the part level while other data is organized at the instance level. To minimize the input processing time, the number of MPI processes specified on the command line should be one (1) more than the maximum number of parts or instances in the mesh input file. This way, one process will be responsible for handling a single part or instance. For example, if there are 2 parts with 4 instances of the first part and 1 instance of the second part, then the number of MPI processes to request is 6 in order to achieve the quickest input processing time. If more MPI processes are requested than required by the rule mentioned above, the extra MPI processes remain idle until the particle transport is started. If fewer MPI processes are requested than required by the rule mentioned above, the MPI processes split the work amongst themselves; there is no load balancing.

The *um_ convert* utility (see Chapter 15) is a highly parallelized program that can convert the `Abaqus` inp file to the `MCNPUM` file type (see Chapter 11). This file type is highly recommended when a complex geometry will be used more than once.

This page intentionally left blank.

Chapter 7

MCNP6 Plotter

Limited plotting of the unstructured mesh is possible with the MCNP6 plotter. It is only possible to produce shaded plots of the mesh pseudo-cells by material, atom density, or mass density so the user may see that the unstructured mesh is positioned correctly relative to the CSG. No cell outlines or unstructured mesh lines are possible. Labels may appear but will not be correct. See Figures 7.1 to 7.5 for several examples. Overlaps may make regions appear distorted, Figure 7.4. Gaps may give rise to extended regions of the background material, Figure 7.5.

Caution should be exercised with large mesh files. While the plotter should be able to plot large mesh geometries, it may take a long time to build the model if the sequential version of the code is used; parallel plotting is not supported, but the parallel version of the code will be beneficial in terms of processing the input.

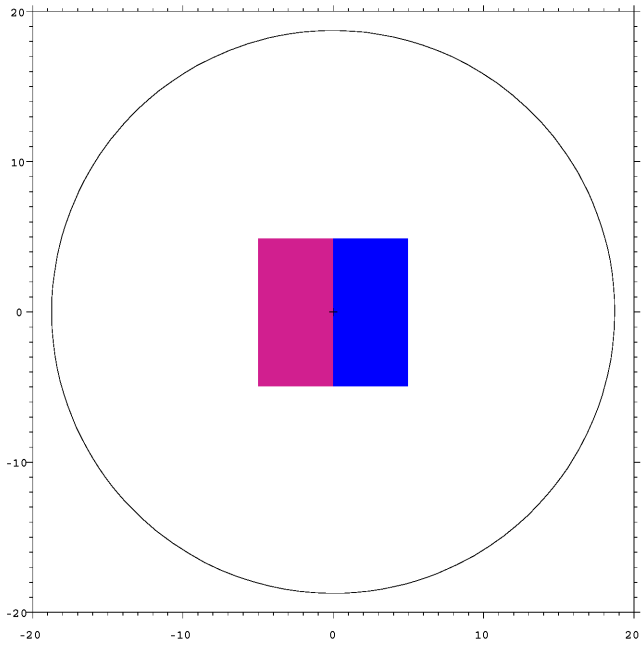


Figure 7.1: Pseudo-cells shaded by material in the mesh universe.

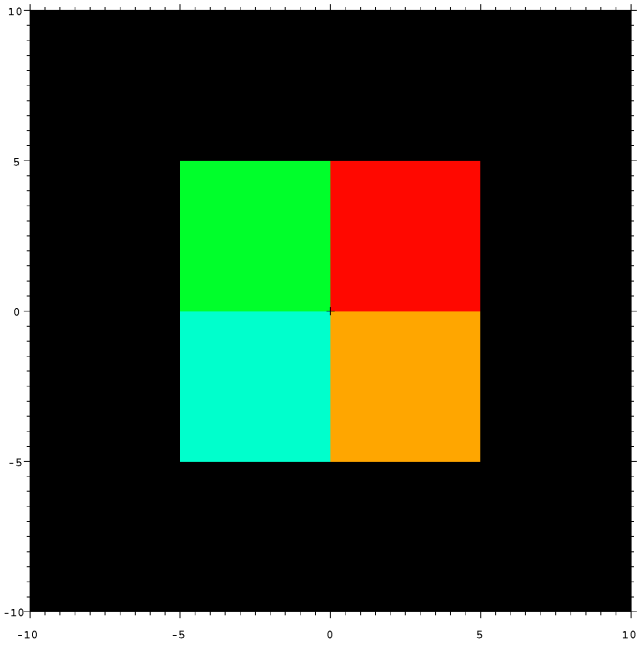


Figure 7.2: Pseudo-cells shaded by material density.

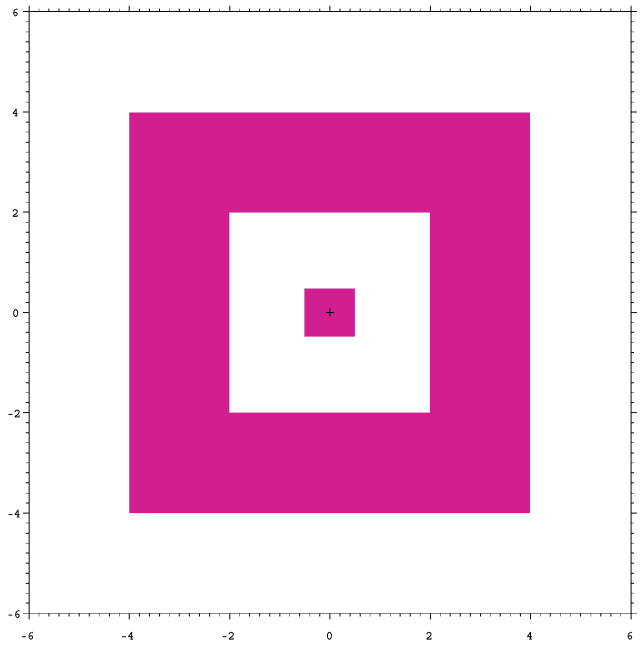


Figure 7.3: Model demonstrating correct plotting of a gap.

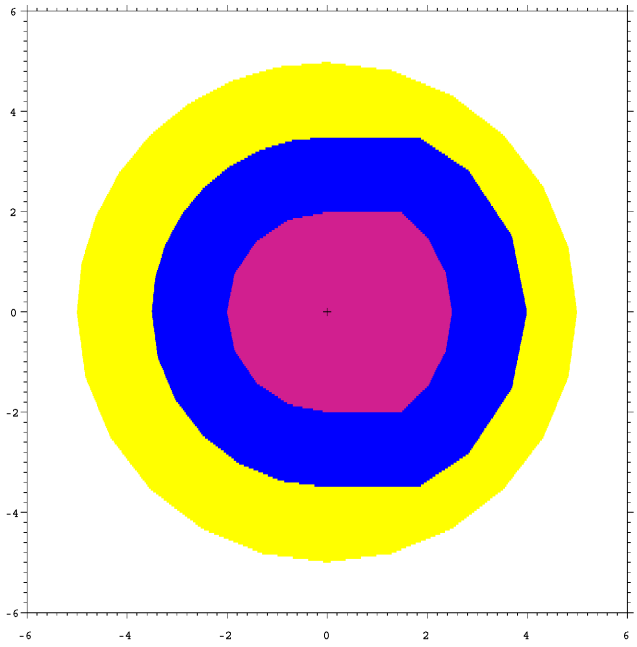


Figure 7.4: Model demonstrating overlaps.

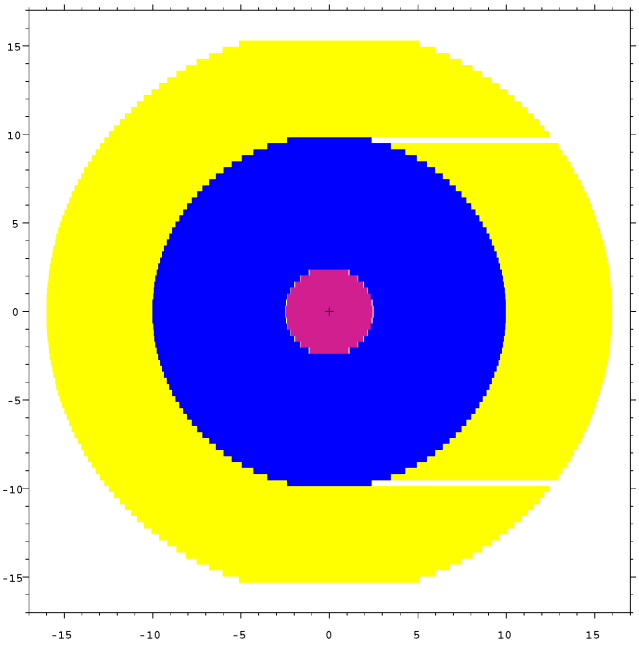


Figure 7.5: Model demonstrating gaps.

This page intentionally left blank.

Chapter 8

Limitations and Restrictions

At this time, the unstructured mesh capability is not fully integrated with all of the pre-existing MCNP6 features. Also, there are a number of proposed mesh-specific features for implementation in the future. This chapter highlights the limitations and restrictions known at this time. Several limitations and restrictions have been removed from this list since this manual was first issued.

- limited to neutrons, photons, electrons with the default physics option, protons, and charged particles heavier than protons. Should not be used with magnetic fields.
- unstructured mesh can not be placed inside a lattice
- an universe can not be placed within a mesh universe
- CSG surfaces must not clip or intersect the unstructured mesh
- the MCNP6 plotter will plot limited aspects of the unstructured mesh for the purpose of seeing its position in the hybrid geometry
- mesh surfaces can not be used for surface sources; normal surface source reads and writes have under gone limited testing with the UM and are not guaranteed to work with it
- reflecting and periodic boundary conditions are not guaranteed to work with the unstructured mesh itself but should work with CSG cells/surfaces that have these conditions
- source particles may not be started in mesh gaps
- surface tallies are not permitted in the unstructured mesh, but can still be used with CSG surfaces
- only pentahedra and hexahedra may appear together in a part; otherwise a part must contain only a single mesh type
- overlapping parts must not be severe; any single element may not be wholly contained within another element
- even running parallel with MPI, problem setup may take considerable time if any one part has many ($>$ 30,000 to 50,000) elements; consider converting to the `MCNPUM` file type.

It is possible that other items have been overlooked and should be added to this section. For example it is unknown whether a `PTRAC` file will contain all surface related information. Not all combinations of parameters associated with the SDEF card have been tested in conjunction with the unstructured mesh volumes sources.

Chapter 9

The Abaqus Input File

9.1 Introduction

This chapter provides a brief description of the Abaqus/CAE input file (inp) and how it relates to the Revised Extended Grid Library (REGL) as implemented in MCNP6. The "format" that is discussed in this section is based on what Abaqus/CAE is currently generating and merits two cautionary points: 1) Depending upon the purpose of the model created in CAE and who is generating it (mechanical vs nuclear engineer, for instance), there may be extraneous information in the input file that could cause problems with the MCNP6 input file parser. What is shown in this chapter is the basic information that MCNP6 needs. If the user is not careful, Abaqus may include information in the inp file that causes problems for MCNP6. For best results only include the data types discussed here. 2) Other CAD/CAE tools can write what they claim to be an Abaqus compatible file, but in reality may include a wrinkle in the file formatting that MCNP6 can't handle. It is the user's responsibility to ensure that the Abaqus/CAE input file meets the requirements outlined here.

9.2 Abaqus inp File

As it relates to the unstructured mesh (UM) library, this chapter describes the ASCII inp file that Abaqus/CAE generates. Greater detail on the format can be found in the Abaqus documentation. In general, comments begin with a double asterisk, "**". Keyword lines begin with a single asterisk, "*" and are followed by the keywords. Data lines have no special characters preceding them, but generally contain integer and/or real numbers. A sample inp file follows this discussion and it includes all three of these line types.

The sample inp file that follows is color-coded for ease of reading. The keywords of interest to the unstructured mesh parser are shown in blue. Several special tags, also of interest to the parser, are shown in red and are discussed below. The model present in this sample file is simple and consists of one part that has been instantiated (replicated) four times; this is discussed in more detail in the assembly section below.

Each of the keywords of interest to the unstructured mesh parser are discussed in various detail next in the order that they usually appear in the inp file. Some keywords occur in pairs, meaning that there is a keyword that starts a block of data and another keyword that ends a block of data. Other keywords are singular in that they start a block of data and an unrelated

keyword or comment ends the block. In the following, keywords are shown in mixed case; the UM input parser converts characters to lower case.

9.2.1 Part

The "*Part" keyword signifies the beginning of the information for a particular part. Each part is given a name that begins after the "name=" characters on the keyword line. The UM library parser retrieves everything after the equals sign up to and including 256 characters in the name.

This name is used by the UM library in locating the correct part when it is instantiated in the assembly. The part name is also used when the UM library outputs information about the mesh model. Do not use any of the elset keywords in the name of the part. All characters are converted to lower case.

9.2.2 Node

The "*Node" keyword appears in the "*Part" block and signifies the beginning of the node data specific to the part. Each line following this keyword contains four numbers: one integer and three real numbers. The integer is nothing more than the node number starting at 1 and going sequentially to the maximum number of nodes in the part. The three real numbers are the x-, y-, z-locations of the given node.

9.2.3 Element

The "*Element" keyword appears in the "*Part" block and marks the beginning of the element connectivity data. Appearing on this keyword line is a description of the type of elements appearing in this part. The element type codes appearing on this line that the UM library can handle are presented in Table 9.1. In the example that follows, the type code is presented in red-lettered characters on the "*Element" keyword line.

Table 9.1: Element Type Codes

Element Type	Type Code
First-order tetrahedra	C3D4
First-order pentahedra	C3D6
First-order hexahedra	C3D8
Second-order tetrahedra	C3D10
Second-order pentahedra	C3D15
Second-order hexahedra	C3D20

Each line following this keyword contains a variable number of integers depending upon the number of nodes that define the element. In the type code given in the table above, the number of nodes for a particular element type appears as the number following the letter "D". The first integer on the data line is the element number; the remainder are the node numbers that define the element. The exception to this is second order hexahedra where two lines are required for each element. For these, the first line contains the element number plus 15 node numbers; the second line contains the remaining 5 node numbers and is generally indented.

When multiple element types appear in a part, Abaqus places multiple "*Element" keyword sets in the "*Part" block. Currently, the UM library can only handle mixed element parts containing pentahedra and hexahedra. If tetrahedra are needed in the model, they must appear in parts that don't contain the other element types.

Other Abaqus type codes may generate similar connectivity data. It is the user's responsibility to ensure use of codes from Table 9.1.

9.2.4 Element Set

In Abaqus parlance, element sets are referred to as elsets and the "*Elset" keyword signifies the beginning of the elset data. The elset mechanism permits the grouping of elements in order to assign various properties. At least two different elsets (see Section 3.1) should be defined or named for the mesh input to be useful to the UM library; these names are easy to find in the example that follows -- look for the red-lettered characters after the "*Elset" keyword that is in a blue font.

The first elset is the material elset and is required. All of the elements in a part must be assigned a material number. The name or tag for this elset must contain the word "`material`" and the material number. At this time, the material number must be the last part of the tag and it must be separated from the rest of the tag by an underscore character or hyphen. In addition to the material elset tag presented in the example at the end of this discussion, the following tag is also acceptable:

```
Set-my_material_uranium_02
```

Note that any number of characters can appear between the word "`material`" and the material number. But, the total length of the line containing the keyword and the tag is limited to 256.

The second elset is the *statistic* or *tally* elset. This elset is optional, but highly recommended. The name or tag for this elset must contain the word "`statistic`" or "`tally`" and the statistic set number. The same rules and conventions apply to this elset tag as for material elsets. All elements in a statistic elset must have the same material number; there is no mixing of materials in the statistic set. The UM library will enforce this.

For each of these keyword types, the data lines following them may be one of two forms. The first of which is just an integer list of element numbers, on the order of 16 integers per line. The second form is in compact notation where the word "generate" appears on the keyword line and the data line consists of 3 integers. The first integer is the starting element number. The second integer is the ending element number. The third integer is the stride from the starting to the ending element numbers. For example, to specify all of the odd element numbers from 1 to 27, use the following:

```
1, 27, 2
```

The UM library uses these two elsets (material and statistic) to define the pseudo-cells that the user must map back to the calling code (in this case MCNP6) cells. Every unique material-statistic elset combination is a new pseudo-cell. The library outputs a "Pseudo-Cell Cross-Reference" table that shows how the pseudo-cell number matches with the calling code (MCNP6) cell number, the instance number, the part number, the material number, and the material name.

9.2.5 End Part

The "*End Part" keyword marks the end of a part's input. Another part description may follow, in which case there will be another "*Part" keyword to signify its beginning, or the assembly description may follow.

9.2.6 Assembly

The "*Assembly" keyword appears after all of the parts are defined. A look at the sample file presented at the end of this chapter shows that an assembly name appears after this keyword much like what appeared for the part. The UM library does not care what the assembly name is; it only uses the keyword to know that all of the part input is complete.

There is also an "*End Assembly" keyword that signifies the end of a particular assembly. Between these two keywords is the important information that the UM library needs in order to construct the mesh model from the parts.

9.2.7 Instance

Appearing in the Assembly block are the "*Instance" keywords. The number that appear here correspond to the number of parts that were instantiated into the Assembly; there is one for each instance. There are two parameters appearing on the "*Instance" keyword line: "name" and "part". The "name" parameter is just the name of the instance and, unless changed by the user in the CAE tool, is just the part name appended with an instance number. The "part" parameter is of interest to the UM library since this is the same name as one of those used with the "*Part" keyword. The UM library uses this "part" parameter name to match with the "*Part" keyword name in order to locate the right one to use.

The "*End Instance" keyword marks the end of the information block for a particular instance. From the example at the end of this discussion there are four instances of the same part. The last instance in this example has no additional lines between the "*Instance" and "*End Instance" keyword lines while the other three have one or two data lines present that describe the translation or rotation of the part as it was instantiated into the assembly.

The first data line appearing between the key words is the translation information. The three real values given here are nothing more than the values of the translation applied in the x-, y-, and z-directions, respectively.

If the part is rotated as it is instantiated into the assembly, two lines appear between the instance keyword lines. The first line is the translation information as discussed previously. If there is a pure rotation the values for the three real numbers on this line are all zero. If there is both a translation and rotation, the translation is applied before the rotation.

There are seven real numbers that appear on the rotation line. The first six real numbers define an axis of rotation. The first three numbers are the x-, y-, and z-locations of the first point that defines the axis. The second three numbers are the x-, y-, and z-locations of the second point that defines the axis. The last or seventh number is the angle of rotation in degrees about the axis.

In the example given at the end of this discussion, the first and third instances have just a translation while the second instance has a rotation but no translation. The fourth instance is neither translated or rotated.

9.2.8 Material

The next to last keyword of interest to the UM library appearing in the inp file is "*Material". This keyword has one parameter which is the material name. The UM library parser retrieves everything after the equals sign up to and including 256 characters in the name. Please see the first section (3.1) of the user's guide for the recommendations in naming materials.

9.2.9 Density

The last keyword of interest to the UM library appearing in the inp file is "*Density". On the next line following this keyword is the actual, user-specified density. If the number is positive, the library treats it as a number density; if it is negative, then the library interprets it as the negative of a physical density. This is the same convention as MCNP6. This keyword and associated value is needed by the *um_pre_op* program, Chapter 14.

9.2.10 Example Abaqus .inp File

```
*Heading
** Job name: job_block_demo_01 Model name: Model-1
** Generated by: Abaqus/CAE 6.10-1
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=Part-block_01
*Node
      1,           4.,           4.,           4.
      2,           4.,           2.,           4.
      3,           4.,           0.,           4.
      4,           4.,           4.,           2.
      5,           4.,           2.,           2.
      6,           4.,           0.,           2.
      7,           4.,           4.,           0.
      8,           4.,           2.,           0.
      9,           4.,           0.,           0.
     10,           2.,           4.,           4.
     11,           2.,           2.,           4.
     12,           2.,           0.,           4.
     13,           2.,           4.,           2.
     14,           2.,           2.,           2.
     15,           2.,           0.,           2.
     16,           2.,           4.,           0.
     17,           2.,           2.,           0.
     18,           2.,           0.,           0.
     19,           0.,           4.,           4.
     20,           0.,           2.,           4.
     21,           0.,           0.,           4.
     22,           0.,           4.,           2.
     23,           0.,           2.,           2.
     24,           0.,           0.,           2.
     25,           0.,           4.,           0.
     26,           0.,           2.,           0.
     27,           0.,           0.,           0.
*Element, type=C3D8R
1, 10, 11, 14, 13,  1,  2,  5,  4
2, 11, 12, 15, 14,  2,  3,  6,  5
3, 13, 14, 17, 16,  4,  5,  8,  7
4, 14, 15, 18, 17,  5,  6,  9,  8
5, 19, 20, 23, 22, 10, 11, 14, 13
6, 20, 21, 24, 23, 11, 12, 15, 14
7, 22, 23, 26, 25, 13, 14, 17, 16
8, 23, 24, 27, 26, 14, 15, 18, 17
```

```
*Nset, nset=Set-material_01, generate
  1,  27,   1
*Elset, elset=Set-material_01, generate
  1,  8,  1
*Nset, nset=Set-statistic_01, generate
  1,  27,   1
*Elset, elset=Set-statistic_01, generate
  1,  8,  1
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=Part-block_01-1, part=Part-block_01
          4.,          0.,          0.
*End Instance
**
*Instance, name=Part-block_01-2, part=Part-block_01
          0.,          0.,          0.
          0.,          0.,          0.,          1.,          0.,          90.
*End Instance
**
*Instance, name=Part-block_01-3, part=Part-block_01
          4.,          0.,         -4.
*End Instance
**
*Instance, name=Part-block_01-4, part=Part-block_01
*End Instance
**
*End Assembly
**
** MATERIALS
**
*Material, name=Material-part1_001
*Density 0.047984,
```

This page intentionally left blank.

Chapter 10

The EEOUT File

10.1 Introduction

This chapter provides a brief description of version 6 of MCNP6's elemental edit output file (`EEOUT`) generated by the Revised Extended Grid Library (REGL).

10.2 EEOUT File

This chapter describes the elemental edit output file from MCNP6, otherwise known as the `EEOUT` file. This file contains a variety of information besides the edit results that have been calculated on a given mesh. Mainly, the information in this file consists of the results, known as edits, and a generic description of the mesh. What is meant by generic is that the mesh description in the file bears little resemblance to that created by the tool which generated the mesh. Therefore, many specific formats may eventually be read by the mesh library, but only one output format will be supported. In that regard, the format for the `EEOUT` file has been developed to accommodate what is thought to be all of the relevant data not only for post-processing but also for problem restart. Some of the data present in the file may be in a form that is only relevant to the REGL.

The following description is for version 6 of the `EEOUT` file and is similar to previous versions. An example `EEOUT` file follows this discussion and is a composite of two different problems. This composite file was done to make it easier to illustrate some data sets and to keep the example short. Some lines in this example are color coded for easier identification.

Note that this implementation of the unstructured mesh library is with Fortran and that both ASCII and binary versions of the `EEOUT` file are possible. Fortran inserts beginning and ending record markers around each binary file record; this should be taken into consideration when using a non-Fortran programming language to construct a routine that reads this file. However, if your distribution comes with the source code, consider using REGL to read the `EEOUT` file. See the UM utilities for examples of how to work with REGL.

10.3 Self-Describing File

The `EEOUT` file was designed to be a self-describing file with the goal of allowing easy access to and identification of the file's data for those developers who choose not to link with the mesh library and use its routines. The meta data and keyword-value pairs (KWV-pairs), both

discussed below, permit the developer a number of options in terms of parsing through the file to extract relevant information.

The data in the file is grouped into data-sets with at least two data-set segments and at most three data-set segments per data-set. The three data-set segments are

- identification
- title line
- data

Except for the first line of the file, each data-set adheres to this convention. All data-sets must contain the data-set identification which is nothing more than the meta data that describes the segments following it. There is no justification for the meta data appearing in the file by itself, so either one or both of the other data-set segments follow it.

The `EEOUT` file also uses KWV-pairs. These appear anywhere there is character data. That is, these pairs may appear in either the title line segment or the data segment. The keyword-value pair is a convenient way to group a short description with either a numeric or alphanumeric value. Each pair consists of one or more keywords to the left of a colon (:) and a value to the right. When multiple KWV-pairs appear on a line, they are separated by a semicolon (;).

10.3.1 Identification Segment

This single meta data line always consists of six 8-byte integers, 1) through 6). Their significance is described next and accommodates some flexibility in use.

1. Number of characters in the title line (A value of 0 indicates no title line segment)
2. Number of records in the data-set after the title record (A value of 0 indicates no data segment)
3. Data type that appears in the data segment. No mixed types are permitted.

   0 - no data lines follow (redundant when 2) = 0)
   1 - character data
   2 - integer data
   3 - real data
4. Size in bytes of each 3) data item. If 2) = 0, then 4)'s value is meaningless.
5. Number of items in each data record. If 2) = 0, then 5)'s value is meaningless.
6. Parse length of each record. This is the number of entries formatted for each ASCII data line. If 2) = 0, then 6)'s value is meaningless.

10.3.2 Title Line Segment

The title line data segment is optional. However, it must be present if there is no data segment. In this sense, the data is contained within the title line as one or more KWV-pairs. This line is always interpreted as character data so that item 1) in the meta data line is a positive integer. Generally, this title line describes the data that follows it.

10.3.3 Data Segment

The data segment is optional. However, it must be present if there is no title line segment. This data may be character, integer, or real as indicated by item 3) in the meta data. Most of the data segments in the `EEOUT` file are either integer or real. In some instances where there is character data in this segment it may be something as simple as a list of material names or it may be KWV-pairs.

10.4 The EEOUT File Description

The following sections discuss the various data-sets that appear in the `EEOUT` file in the order that they appear. As mentioned above, an example file follows, Section 10.5. In the example file all identification segments (meta data) appear in red and all title line segments appear in blue.

10.4.1 First Line

The first line of the `EEOUT` file is a description line that contains exactly 12 characters. If the file is the ASCII version, the 12 characters, ignoring the double quotes, are "MCNP EDITS A", where the "A" stands for ASCII. If the file is the binary version, the 12 characters, ignoring the double quotes, are "MCNP EDITS B", where the "B" stands for binary. Note that there is no meta data line preceding this line.

In the binary version of this file there will be Fortran inserted record markers before and after these 12 characters. If the developer is using a programming language other than Fortran to read the file, the length of the markers can be deduced from the total length of this line. With this information, subsequent records in the file can be read and the markers ignored to obtain the record information.

10.4.2 First Data Set

The first data set in the file does not contain a title line segment, but contains two KWV-pairs in two records. From the first pair, the value provides the mesh source. Since the Abaqus/CAE inp file is currently the only mesh input file that the library reads, the value is "ABAQUS". From the second KWV-pair, the value provides the version number of the `EEOUT` file.

10.4.3 Calling Code Labels

The second data-set consists of KWV-pairs containing descriptive information from the code that calls the mesh library. In the case of MCNP6, there are 8 labels that it passes to form the KWV-pairs in the output. Note that the calling code has inserted a special character, "|", to signify the end of meaningful characters on a line. The first one has keywords "Prob ID" and is the problem description supplied by the user in the MCNP6 run. The second and third KWV-pairs have the keywords "Calling Code" and "Code Version" which in this case confirms that the code using the library is MCNP6 and its associated build version. The fourth KWV-pair provides the Date & Time that the `EEOUT` file was generated. The fifth through eighth KWV-pairs supply four files associated with the MCNP6 calculation that generated the `EEOUT` file. These four files are

- the MCNP6 inp file
- the MCNP6 outp file
- the MCNP6 runtpe file
- the Abaqus inp file that contains the mesh description

Other associated files may by added to this data-set in the future.

NOTE: When the third entry on MCNP6's `prdmp` card is set to -1, this data-set is not present.

10.4.4 Integer Parameters

The third data-set contains 12 KWV-pairs where the value part of the pair is an integer. The second through tenth pairs are parameters associated with the mesh geometry and their names are self-explanatory. They are the numbers of nodes, materials, instances, first-order tetrahedra, first-order pentahedra, first-order hexahedra, second-order tetrahedra, second-order pentahedra, and second-order hexahedra.

The first KWV-pair is the number of particles in the calculation.

The eleventh KWV-pair is the number of histories from the Monte Carlo calculation upon which the edit results are based. This is the number that is used in normalizing the edits.

The twelfth KWV-pair provides the number of edits or embee cards that were specified in the input.

10.4.5 Real Parameters

The fourth data-set contains 2 KWV-value pairs where the value parts of the pairs are real numbers. In the first pair, the value is the length conversion for all of the spatial coordinates from the input mesh file and represents the multiplier needed to convert from the units of the original mesh model to centimeters (in this case the units required by MCNP6). This value has been applied to all coordinates appearing in the `EEOUT` file and consequently is reflected in all of the results. In the second pair, the value is the normalization factor that has been applied to all results in the file. This factor is used to un-normalize the results for continue runs.

10.4.6 Particle List

The fifth data-set contains a list of the particle numbers from the calling code. In the example given, there are two particles and the numbers in the data set are 1 and 2. Since this was MCNP6 writing the file, these number correspond to neutrons and photons. If the number would have been 2 and 3, the particles would be photons and electrons.

10.4.7 Particle Edit List

The sixth data-set is a mapping of the particles to the edits and is needed internally by the code. Inside the code this information is stored in a 2-D array where the first index is for the edit number (where the maximum number of entries corresponds to the total number of edits) and the second index is for the particle. The value stored in any array slot is the internal edit number to which the particle contributes. For the example here, it can be seen that neutrons contribute

to both edits while gammas contribute only to the second edit. A value of 0 terminates the particle's list if there are fewer particles in the edit than the maximum number of particles in the problem.

10.4.8 Edit Description

The seventh data-set begins with the title EDIT DESCRIPTION and contains 6 integers: the number of different particles, the number of elemental edits -- this is for both the second and third entries (one of these will be removed at a later date), the maximum number of problem energy bins, the maximum number of problem time bins, and the maximum number of response bins.

10.4.9 Edit Data Groups

At this point in the file there begins a variable number of data-sets which describe details of the elemental edits. What follows for each particle in the problem are 5 data-sets beginning with EDIT DATA and ending with RESPONSE BINS. Except for the unit conversion factors, most of the information presented in these next data-sets also appear in the title lines of the edit data-sets that appear later in the file.

The EDIT DATA data-set set contains 8 integer values described in the following table:

Integer	Description
1	internal edit number
2	user edit number; negative if errors requested
3	special combined energy deposition indicator; 9 if a combined edit, 0 otherwise
4	particle number in REGL
5	particle number from MCNP6
6	number of energy bins
7	number of time bins
8	number of response bins

The CONVERSION FACTORS data-set provides two real numbers in one record: the energy unit conversion factor followed by the time unit conversion factor.

The next three data-sets each contain two real records. The ENERGY BINS data-set supplies the upper energy cut points for the energy bins followed by the energy multipliers for these energy bins. The TIME BINS data-set provides similar information for the time bins. The RESPONSE BINS data-set provides similar information for the response bins. There should always be one energy, one time, and one response bin whether requested by the user or not. It is up to the calling code to enforce this.

10.4.10 Materials

This data-set contains the alphanumeric names of the materials to associate with the material numbers assigned to each element. The names are ordered alphanumerically.

10.4.11 Cumulative Instance Element Totals

Parts are instantiated into the global mesh model in the order directed by the mesh input file. As the parts are added, the number of elements in that part are totaled and stored sequentially in the cumulative element totals array. The first element of this array contains the number of elements in the first instance. For the number of elements in the remaining instances (2 through max number of instances), subtract the value in the preceding array location from the instance's array location value. The values appearing in this data-set are just the cumulative values stored in this array. This information is primarily of interest internally to the mesh library and may be eliminated at a later date from the `EEOUT` file.

10.4.12 Instance Element Names

This data-set contains the alphanumeric names of the pseudo-cells. There is one record in the data segment for each pseudo-cell and, generally, these names are allowed to be 256 characters long. The order of the names in this data-set is the same order with which they are added to the global mesh model as directed by the mesh input file.

The user should note that the pseudo-cell names are slightly altered from what appears in the Abaqus inp file. Abaqus can segment a part. Each of these segments in REGL becomes a pseudo-cell. Because of a recent infrastructure change in REGL, it was necessary to separate the pseudo-cells from the instances and promote the pseudo-cell as the entity that builds the assembly. When the pseudo-cells are separated from the instances, a name is assigned to the pseudo-cell based on the original instance name. The instance name is appended with the letter P and a number starting at 1. The number is incremented for each additional pseudo-cell removed from the instance.

In a future version of this file, the file name of this section may be changed.

10.4.13 Instance Element Type Totals

The elements in the global mesh model are ordered and numbered by element type. This standard order is first-order tetrahedra, first-order pentahedra, first-order hexahedra, second-order tetrahedra, second-order pentahedra, and second-order hexahedra. Element numbers proceed sequentially from 1 to the maximum number of elements in the model. Any first order tetrahedra has an element number that is less than the first first-order pentahedra that appears in the model. Similar statements can be made regarding the element numbers concerning the other element types. For example, the first instance added to the model may contain a mixture of first-order pentahedra and hexahedra. The second instance added may contain only first-order tetrahedra. Even though the instance containing the tetrahedra was added later, its element numbers will always be less than the instance containing the pentahedra and hexahedra.

This data-set contains one record of 12 integers for each pseudo-cell in the model and the records appear in the order in which the pseudo-cells were added to the global mesh model as directed by the mesh input file. These 12 integers are grouped into pairs with each pair providing the first global element number and the last global element number for each element type. The order of the pairs is in standard order. In the example provided in this document, the first pseudo-cell contains only second-order hexahedra and its first element has global element number 204 and its last global element number is 331.

In a future version of this file, the file name of this section may be changed.

10.4.14 Nodes Group

The next three data-sets contain node location data. The first set, "NODES X (cm)", lists all of the x-locations for nodes 1 through max number of nodes. The second set, "NODES Y (cm)", lists all of the y-locations for nodes 1 through max number of nodes. The third set, "NODES Z (cm)", lists all of the z-locations for nodes 1 through max number of nodes. As indicated in the title line, these values are in centimeters, the required unit for the calling code (in this case, MCNP6).

10.4.15 Element Type

This data-set contains integers that describe the element type for each of the global elements starting at 1 and proceeding to the maximum number of elements in the mesh model. First-order tetrahedra, pentahedra, and hexahedra are given the values 4, 5, and 6, respectively. These number are just the number of faces in each element type. Second-order tetrahedra, pentahedra, and hexahedra are given the values 14, 15, and 16, respectively. These number are just the number of faces in each element type plus 10.

10.4.16 Element Materials

This data-set contains integers that represent the material number assigned to each element. Each element in the global mesh model is associated with a material through its material number. The elements appear sequentially from 1 to the maximum number of elements in the global mesh model.

10.4.17 Connectivity Data Group

There are a variable number of connectivity data-sets appearing in the `EEOUT` file, depending upon the element types present in the model. If all six types appear, there will be six data-sets appearing in standard order. In the example provided in this document, there is only one data-set in this group and it is for the first-order hexahedra.

The title line in this data-set contains the text ELEMENT ORDERED. This means that nodes appear by element. All of the nodes for the first element appear before the nodes for the second element, etc. This is a change from earlier versions of this file where the information was NODE ORDERED where all of the first nodes of all elements appeared before all of the second nodes of all of the elements, etc.

10.4.18 Nearest Neighbor Data Group

There are a variable number of nearest neighbor data-sets appearing in the `EEOUT` file, depending upon the element types present in the model. If all six types appear, there will be six data-sets appearing in the standard order. In the example provided in this document, there is only one data-set in this group and it is for the first-order hexahedra.

This data is ordered in the same fashion as the connectivity data. All of the neighbors for the first element appear before all of the neighbors of the second element, etc. In addition, the ordering of the neighbors is by face number. Therefore, a 0 appearing in the third neighbor position means there is no element appearing as a neighbor on that face.

10.4.19 Edit Sets Group: Data Output and Data Sets

Depending upon the edit requests from the calling code, a variable number of edit set results appear after the nearest neighbor data. Starting with the first particle and continuing through the total number of particle types tracked on the mesh, all of the regular edits are output by particle type. The title line segment that appears in all of these data-sets contain KWV-pairs which provide details describing the edit set.

Each particle edit list combination comprises its own edit group. The start of this group of edits is signified with a data-set consisting of just the meta data segment and a title line segment with three KWV-pairs. The keyword for the first KWV-pair is DATA OUTPUT PARTICLE and its value is the particle number. The keyword for the second KWV-pair is EDIT LIST and its value is just the edit list number for the particle. The edit list is a list used by the mesh library. The keyword for the third KWV-pair is TYPE and its value is a set of alphanumeric characters that are an amalgamation of the edit type (e.g., FLUX) and the edit number (e.g., 14) specified in the calling code.

The next data-set is the DATA OUTPUT COMMENT data-set and consists of the meta data segment and a title line with one KWV-pair. The keyword is always DATA OUTPUT COMMENT. If no comment was specified by the user for the edit, the value field is left blank; otherwise, it contains the comment that was provided in the input.

After the DATA OUTPUT COMMENT data-set, the remainder of the data-sets forming the edit list appear. These data-sets are full data-sets with title line and data segments. The title line segment has six KWV-value pairs containing the time and energy bin numbers, bounds, and multipliers. Note that in order to avoid a KWV-pair with a non-existent value, extra keywords were added to the first KWV-pair; these extra keywords are DATA SETS and flag the data-set as the one with the numerical results. The keywords TMULT and EMULT are shorthand for time bin multiplier and energy bin multiplier, respectively.

If either the time or energy domains are broken into bins, the mesh library will automatically sum the bins to produce a total result. When this appears in the file the bin number is replaced with the string TOTAL and the corresponding bin values and multipliers are replaced with the string N/A, indicating that this information is not applicable because it was not input by the user. If both the time and energy domains are broken into bins, the mesh library will automatically sum the bins to provide total time results for each energy bin and total energy results for each time bin in addition to total time and total energy results.

After all of the regular edit set information is written to the `EEOUT` file, any edit sets for composite edits appear. The only thing that differs with this edit group is the particle descriptor in the DATA OUTPUT title line. For the regular edits the value of the first KWV-pair is a particle number. For the composite edits the value is a string where the particle numbers have been blended to produce an unique identifier (e.g., 1_2).

NOTE: Users familiar with earlier versions of MCNP6 and the `EEOUT` file should recognize that the components of the composite edit are no longer handled separately. This was done to save memory for really large problems.

10.4.20 Centroids Group

After the edit set data-sets there appear three data-sets for the element centroids. These three data-sets are presented in a similar fashion as the node information. X-centroids for all elements appear first in their own data-set followed by data-sets for the y-centroids and z-centroids,

respectively.

10.4.21 Densities & Volumes

The next to last data-set is the material density values for each element for elements number one to the maximum number of elements in the global mesh model. The units for these values are grams per cubic centimeter as indicated in the corresponding title line.

The last data-set contains the volumes for each element for elements number one to the maximum number of elements in the global mesh model. The units for these values are cubic centimeters.

10.5 Example EEOUT File

```
MCNP EDITS A
             0             2             1            16             1             1
 EEOUT  : ABAQUS
 VERSION:      6
            20             8             1           256             1             1
CALLING CODE LABELS
Prob ID         : simple cube, each element is a statistical set, 8 total|
Calling Code    : MCNP6|
Code Version    : 6.1.88|
Date & Time     :  02/08/11  09:48:4|
Inp File        : inp1007a|
Outp File       : inp1007ao|
Runtpe File     : inp1007ar|
Geom Inp File   : um1007.inp|
             0            12             1             1            42             1
NUMBER OF PARTICLES:                    2
NUMBER OF NODES    :                   27
NUMBER OF MATERIALS:                    6
NUMBER OF INSTANCES:                    6
NUMBER OF 1st TETS :                   30
NUMBER OF 1st PENTS:                    8
NUMBER OF 1st HEXS :                  128
NUMBER OF 2nd TETS :                   29
NUMBER OF 2nd PENTS:                    8
NUMBER OF 2nd HEXS :                  128
NUMBER OF HISTORIES:                 1000
NUMBER OF EDITS    :                    2
             0             2             1             1            43             1
LENGTH CONVERSION   :  1.0000000000000E+00
NORMALIZATION FACTOR:  1.0000000000000E-03
            14             1             2             4             2             2
 PARTICLE LIST
           1           2
            19             1             2             4             4             4
 PARTICLE EDIT LIST
           1           2           2           0
            18             1             2             4             6             6
 EDIT DESCRIPTION
         2       2       2       2       2       1
            10             1             2             4             8             8
 EDIT DATA
       1     -14       0       1       1       1       1       1
            19             1             3             8             2             2
 CONVERSION FACTORS
  1.00000E+00  1.00000E+00
```

```
            12              2              3              8              1              5
 ENERGY BINS
  1.00000E+36
  1.00000E+00
            10              2              3              8              1              5
 TIME BINS
  1.00000E+33
  1.00000E+00
            14              2              3              8              1              5
 RESPONSE BINS
  1.00000E+36
  1.00000E+00
            10              1              2              4              8              8
 EDIT DATA
       2     -36       9       2       2       2       2       1
            19              1              3              8              2              2
 CONVERSION FACTORS
  1.00000E+00  1.00000E+00
            12              2              3              8              2              5
 ENERGY BINS
  2.00000E+00  1.00000E+10
  1.00000E+00  1.00000E+00
            10              2              3              8              2              5
 TIME BINS
  1.00000E+00  1.00000E+39
  1.00000E+00  1.00000E+00
            14              2              3              8              1              5
 RESPONSE BINS
  1.00000E+36
  1.00000E+00
            10              6              1            256              1              1
 MATERIALS
Material_end_lin_hex_01
Material_end_quad_hex_06
Material_mid_lin_pent_04
Material_mid_lin_tet_02
Material_mid_quad_pent_05
Material_mid_quad_tet_03
            36              1              2              4              6              5
 INSTANCE CUMMULATIVE ELEMENT TOTALS
         128        136        166        174        203
         331
            23              6              1            256              1              1
 INSTANCE ELEMENT NAMES
Part-end_quad_hex-1P1
Part-mid_lin_pent-1P1
Part-mid_lin_tet-1P1
```

```
Part-mid_quad_pent-1P1
Part-mid_quad_tet-1P1
Part-end_lin_hex-1P1
            29             6             2             4            12            12
 INSTANCE ELEMENT TYPE TOTALS
         0         0         0         0         0         0         0         0         0         0       204       331
         0         0        31        38         0         0         0         0         0         0         0         0
         1        30         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0       196       203         0         0
         0         0         0         0         0         0       167       195         0         0         0         0
         0         0         0         0        39       166         0         0         0         0         0         0
            13             1             3             8            27             5
 NODES X (cm)
 -5.00000E+00 -5.00000E+00 -5.00000E+00 -5.00000E+00 -5.00000E+00
 -5.00000E+00 -5.00000E+00 -5.00000E+00 -5.00000E+00  0.00000E+00
  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00
  0.00000E+00  0.00000E+00  0.00000E+00  5.00000E+00  5.00000E+00
  5.00000E+00  5.00000E+00  5.00000E+00  5.00000E+00  5.00000E+00
  5.00000E+00  5.00000E+00
            13             1             3             8            27             5
 NODES Y (cm)
 -5.00000E+00  0.00000E+00  5.00000E+00 -5.00000E+00  0.00000E+00
  5.00000E+00 -5.00000E+00  0.00000E+00  5.00000E+00 -5.00000E+00
  0.00000E+00  5.00000E+00 -5.00000E+00  0.00000E+00  5.00000E+00
 -5.00000E+00  0.00000E+00  5.00000E+00 -5.00000E+00  0.00000E+00
  5.00000E+00 -5.00000E+00  0.00000E+00  5.00000E+00 -5.00000E+00
  0.00000E+00  5.00000E+00
            13             1             3             8            27             5
 NODES Z (cm)
  1.00000E+01  1.00000E+01  1.00000E+01  5.00000E+00  5.00000E+00
  5.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00  1.00000E+01
  1.00000E+01  1.00000E+01  5.00000E+00  5.00000E+00  5.00000E+00
  0.00000E+00  0.00000E+00  0.00000E+00  1.00000E+01  1.00000E+01
  1.00000E+01  5.00000E+00  5.00000E+00  5.00000E+00  0.00000E+00
  0.00000E+00  0.00000E+00
            13             1             2             4             8            20
 ELEMENT TYPE
  6  6  6  6  6  6  6  6
            17             1             2             4             8            10
 ELEMENT MATERIAL
       1       1       1       1       1       1       1       1
            49             1             2             4            64             8
 CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED
          10          11          13          14          19          20          22          23
          11          12          14          15          20          21          23          24
          14          15          17          18          23          24          26          27
          13          14          16          17          22          23          25          26
```

```
          1          2          4          5          10         11         13         14
          2          3          5          6          11         12         14         15
          5          6          8          9          14         15         17         18
          4          5          7          8          13         14         16         17
            37             1             2             4            48             6
NEAREST NEIGHBOR DATA 1ST ORDER HEXS
          55         0          0          40         47         0
          56         0          0          41         48         39
          57         0          0          42         49         40
          58         0          0          43         50         41
          59         0          0          44         51         42
          60         0          0          45         52         43
          61         0          0          46         53         44
          62         0          0          0          54         45
            58             0             0             0             0             0
DATA OUTPUT PARTICLE : 1 ; EDIT LIST : 1 ; TYPE : FLUX_14
            22             0             0             0             0             0
DATA OUTPUT COMMENT :
           145             1             3             8             9             5
DATA SETS RESULT TIME BIN : 1 ; TIME VALUE : 1.000E+33 ; TMULT : 1.00000E+00 ; ENERGY BIN : 1 ; ENERGY VALUE : 1.000E+36 ; EMULT : 1.00000E+00
0.00000E+00   4.43650E-02   4.52883E-02   4.73776E-02   4.99024E-02
4.24386E-02   4.63551E-02   4.10545E-02   4.90753E-02
            60             0             0             0             0             0
DATA OUTPUT PARTICLE : 1 ; EDIT LIST : 2 ; TYPE : ENERGY_36
            22             0             0             0             0             0
DATA OUTPUT COMMENT :
          145             1             3             8             9             5
DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 1 ; ENERGY VALUE : 2.000E+00 ; EMULT : 1.00000E+00
0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
           145             1             3             8             9             5
DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 2 ; ENERGY VALUE : 1.000E+10 ; EMULT : 1.00000E+00
0.00000E+00   2.39444E-02   2.43435E-02   2.58262E-02   2.70384E-02
2.33694E-02   2.53041E-02   2.24486E-02   2.64320E-02
           137             1             3             8             9             5
DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : TOTAL ; ENERGY VALUE : N/A ; EMULT : N/A
 2.39444E-02   2.43435E-02   2.58262E-02   2.70384E-02  2.33694E-02
 2.53041E-02   2.24486E-02   2.64320E-02
            60             0             0             0             0             0
DATA OUTPUT PARTICLE : 2 ; EDIT LIST : 1 ; TYPE : ENERGY_36
            22             0             0             0             0             0
DATA OUTPUT COMMENT :
           145             1             3             8             9             5
DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 1 ; ENERGY VALUE : 2.000E+00 ; EMULT : 1.00000E+00
0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
           145             1             3             8             9             5
```

```
 DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 2 ; ENERGY VALUE : 1.000E+10 ; EMULT : 1.00000E+00
 0.00000E+00   1.23164E-03   1.33096E-03   1.34315E-03   1.38873E-03
 1.19235E-03   1.34903E-03   1.20800E-03   1.41040E-03
           137             1             3             8             9             5
 DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : TOTAL ; ENERGY VALUE : N/A ; EMULT : N/A
  1.23164E-03   1.33096E-03   1.34315E-03   1.38873E-03  1.19235E-03
  1.34903E-03   1.20800E-03   1.41040E-03
            69             0             0             0             0             0
 DATA OUTPUT PARTICLE : 1_2 ; EDIT LIST : 1 ; TYPE : ENERGY_36
            22             0             0             0             0             0
 DATA OUTPUT COMMENT :
          131             1             3             8             9             5
 DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 1 ; ENERGY VALUE : 2.000E+00 ; EMULT : 1.00000E+00
 1.00000E-01   0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
 0.00000E+00   0.00000E+00   0.00000E+00   0.00000E+00
           131             1             3             8             9             5
 DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : 2 ; ENERGY VALUE : 1.000E+10 ; EMULT : 1.00000E+00
 0.00000E+00   2.51760E-02   2.56744E-02   2.71694E-02   2.84271E-02
 2.45618E-02   2.66531E-02   2.36566E-02   2.78424E-02
           137             1             3             8             9             5
 DATA SETS RESULTS TIME BIN : 1 ; TIME VALUE : 1.000E+39 ; TMULT : 1.00000E+00 ; ENERGY BIN : TOTAL ; ENERGY VALUE : N/A ; EMULT : N/A
  2.51760E-02   2.56744E-02   2.71694E-02   2.84271E-02  2.45618E-02
  2.66531E-02   2.36566E-02   2.78424E-02
            17             1             3             8             8             5
 CENTROIDS X (cm)
 -2.50000E+00 -2.50000E+00 -2.50000E+00 -2.50000E+00  2.50000E+00
  2.50000E+00  2.50000E+00  2.50000E+00
            17             1             3             8             8             5
 CENTROIDS Y (cm)
 -2.50000E+00  2.50000E+00 -2.50000E+00  2.50000E+00 -2.50000E+00
  2.50000E+00 -2.50000E+00  2.50000E+00
            17             1             3             8             8             5
 CENTROIDS Z (cm)
  7.50000E+00  7.50000E+00  2.50000E+00  2.50000E+00  7.50000E+00
  7.50000E+00  2.50000E+00  2.50000E+00
            18             1             3             8             8             5
 DENSITY (gm/cm^3)
  1.87401E+01  1.87401E+01  1.87401E+01  1.87401E+01  1.87401E+01
  1.87401E+01  1.87401E+01  1.87401E+01
            15             1             3             8             8             5
 VOLUMES (cm^3)
  1.25000E+02  1.25000E+02  1.25000E+02  1.25000E+02  1.25000E+02
  1.25000E+02  1.25000E+02  1.25000E+02
```

Chapter 11

Other Files

11.1 GMV File

Often times it is beneficial to have an independent and easy to use program for mesh geometry visualization. The General Mesh Viewer, GMV, program [8] is such a program. For this reason, it is possible to generate a GMV input file (see embed card, parameter `gmvfile`). Note that if during model creation in the CAE tool, the material elsets don't have unique numbers, it will be difficult to differentiate parts in GMV. That is, if each part has one material and the number assigned to that material is the same one in all of the parts, then all elements in GMV will have the same color. Also, GMV limits material names to 8 characters.

This GMV file capability is primarily for LANL use.

11.2 The MCNPUM File

The Abaqus input file contains some basic information about the unstructured mesh, but does not contain everything that MCNP6 needs. Once MCNP6 reads this file, it uses the Abaqus data to generate other information that it needs in its tracking routines such as nearest neighbor lists. Even with the parallel input processing, discussed elsewhere in this document, significant computer time can be required to regenerate this data and create other internal data structures for every MCNP6 calculation that uses the Abaqus unstructured mesh file.

The `MCNPUM` file type [9] was created to contain all of the unstructured mesh data structures that MCNP6 needs, thus eliminating the need to "input process" the Abaqus input file every time the code is run, including continue runs. MCNP6 can generate this file (primarily after processing the Abaqus input file) by simply including the `MCNPUMFILE` option on the embed card. MCNP6 can use the `MCNPUM` file when the `MCNPUM` keyword is supplied to the MESHGEO parameter on the EMBED card. See the input cards Chapter 5.

The *um_ convert* utility (see Chapter 15) is a highly parallelized program that can convert the Abaqus inp file to the `MCNPUM` file type. This file type is highly recommended when a complex geometry will be used more than once.

This page intentionally left blank.

Chapter 12

Verification & Validation

Confidence that MCNP6's UM feature is functioning as intended and calculating the correct answer is highly critical. This chapter documents some of the work that has been completed to meet this goal and is part of the MCNP6 code team's software quality assurance process (SQA). In addition the team has performed code reviews on this software. There is also a substantial amount of internal testing not only by the code developers but by friendly testers.

12.1 Regression Test Problems

What follows in this section is a brief, descriptive list of the UM regression test problems. These problems are run at least every night with the code team's continuous build and test system (CBTS) with various versions of the code; that is, versions of the code that have been built with different compilers and combinations of compiler options. Each test generates a number of output files that can vary by problem and are compared against templates. In non-UM problems, files that are always generated and compared are the outp and mctal files. In addition, for these non-UM problems, `WWOUT`, `PTRAC`, and `MESHTAL` files may be generated and compared. For UM problems, `EEOUT` and GMV files are generated and compared except where noted in the following list; sometimes the `MCNPUM` file is substituted for the GMV file when the problem needs to test some `MCNPUM` aspect of the code. All `EEOUT` files are ASCII unless otherwise noted.

For those users who have access to the MCNP6 source code, these problems are available as problems 1001 to 1063 in the REGRESSION test directory and can serve as examples. A complete understanding of everything mentioned in the following list may only be possible with a familiarity of the MCNP6 source code. In the following list tet is an abbreviation for tetrahedra, pent is an abbreviation for pentahedra, and hex is an abbreviation for hexahedra; plural forms follow.

1. Element type test: 6 rectangular parts. Surfaces in contact, but no overlap or gaps. 6 element types. Fixed source ray-tracing. Generates and tests overlay `FMESH` file. 3 particles on the mode card, but only 1 `EMBEE` for neutrons.

2. Continuation run for problem 1.

3. Pathological gap test: 4 rectangular parts with 7 pseudo-cells. Surfaces in contact, but no overlap. 540 1st order hexs. Fixed source ray-tracing. Generates and tests overlay `FMESH` file.

4. Very hard pathological overlap test: 2 rectangular parts with 4 pseudo-cells. Surface overlap, but no gaps. 820 1st order hexs. Fixed source ray-tracing.

5. Tet criticality test: Godiva with 32 1st order tets. 1 part.

6. Mixed part overlap test: Nickel Osaka benchmark. 3 parts as either sphere or spherical shells with minor overlap. 160 1st order hexs; 440 second-order pents. Mode n only.

7. Simple cube, fixed source test: 1 part with eight 1st order hexs, one per octant. Each hex is a pseudo-cell (i.e., 1 instance with multiple pseudo-cells). Plutonium. Mode n p. Source near center. Multiple time and energy bin edits with errors.

8. Simple cube, criticality test. Same as problem 7 except `KCODE` and writes a binary `EEOUT` file.

9. Continuation run for problem 8. Reads binary `EEOUT` file and writes an ASCII final version.

10. Pathological tet vertex tracking test: 1 rectangular part with 1 pseudo-cell. 569 1st order tets. Mode n p. Point source near a node. History #31931734, seed #1315, generator #1 must track through tet vertex (node) and into an element that isn't on the nearest neighbor list. Other histories also exhibit this phenomenon.

11. Greek helmet, mixed element test. 4 parts. Part #1 – mixed 1st order pents & hexs; Part#2 – mixed second-order pents & hexs; Part #3 – 1st order tets; Part #4 – second-order tets. Tests index setup in input processing as well as the usual tracking routines. Void. Ray-tracing from spherical surface.

12. Volume source test with first order tets, pents, and hexs. Four source parts with air. Hex part is $20 \times 10 \times 10$. Pent part is $10 \times 10 \times 10$. Tet part is $10 \times 10 \times 10$. Shield part is $20 \times 20 \times 90$ with very low density concrete. Writes `FMESH` file.

13. Mesh next to lattice test. Mesh is surrounded by lattice structure. Lattice contains fuel pins (fuel / clad (Fe instead of Zr) / water; `KCODE` w/ 3000 histories/batch, 10 total batches; Mesh block is $6 \times 6 \times 4$ w/ 4 1st order hex elements, 1 per quadrant; center cavity in lattice (where lattice is not defined) is $12 \times 12 \times 20$.

14. Proton. $20 \times 20 \times 30$ tungsten block. 20 GeV mono-energetic, isotropic source. 1500 1st order hexs. 10000 histories. Mesh tally comparison.

15. Proton. Ray-tracing. Overlap. 2 parts overlap from $x = 0$ to $x = 1$. $10 \times 10 \times 10$. 820 1st order hexs. 10000 histories.

16. Proton. Ray-tracing. Big gap. 2 part, concentric boxes. Outer part $8 \times 8 \times 8$. Inner part $1 \times 1 \times 1$. 598 1st order hexs. 100,000 histories.

17. Neutron. Concrete slab with 3200 hexs. `SDEF` erg = 2 MeV. Use & generated cell-based weight windows.

18. Same as #17 except use & generate mesh-based weight windows.

19. Volume cell sources. 1 part hex model, beam, w/ 3 pseudo-cells. Source regions at each end. 2 `VOLUMER` sources not a dependent function of `CEL` parameter. Causes `FATAL` error.

20. Same as #19, but 2 `VOLUMER` sources are dependent function of `CEL` parameter.

21. Same as #20, but added a point source that is in the mesh.

22. Geometry same as #7. Ray tracing source on sphere directed inward. Void. Source with 3 energies to sample from 3 different `EMBDE` / `EMBDF` card sets – enables quick check of data sets in `EEOUT` file.

23. Point detectors. Tets and hexs in contact - - concentric cubes. Copper & graphite blocks. Neutron. Point source.

24. Point detectors. Hex only. Concentric blocks – graphite inner, copper outer. Gap between blocks (causes differences between UM & CSG). Neutron. Point source.

25. Point detectors. Hex only. Concentric blocks with overlap, gaps, and contact. (causes differences between UM & CSG). Neutron. Point source.

26. DXTRAN. 4 parts – hexs and tets. Square cross section duct of air. Cooper cap piece. Copper top hat. Graphite around duct. Neutron problem. Point source.

27. Background material (fix) test for neutrons and photons; UM is box shell w/ 1st order hexs (296), inside & outside is background material; fill cell is a sphere of radius 7 and this contains the background material outside of the mesh; source is on the spherical, fill cell surface and is directed inward.

28. Electron ray-tracing using the same geometry as problem #1: $8 \times 8 \times 6$, 6 parts, 6 different element types.

29. Electron. Mimic regression problem #59. Electron beam on copper target. 8000 1st order hexs. 1 part.

30. Electron. Slab problem with second-order tets. Turn off knock-on electrons and MCS (no deflection). 15 instances. 36000 total tets. 2400 tets per part.

31. Modified (void) Big 10 geometry for ray tracing with second-order tets (3501). Tests tricky intersection at edge/face where 2 attempts at the intersection routine are required.

32. Photon & electron. `F8` and `*F8` tally test w/ 1st order hex shells (air, carbon, aluminum). 3816 elements in 3 parts.

33. Guard against NaN's in `EEOUT` because of improper normalization by voided elements that have 0 density. Same as problem 1007 except half off the elements are voided. Reduced number of edits on `EMBEE`'s. Slightly different tallies.

34. Electron problem; volume source; 11 parts; 13482 1st order tets - total; problem was in infinite loop in strag_landau because of d=0 being passed; offending history was 4192; 1 line fix in electron_history to set dls to huge float.

35. Neutron problem to test tracking on hemi-head w/ mica washer; 13144 1st order hexs; symmetric point source locations (+/-Y); minute gaps in negative Y half that demonstrate the tolerance of 10^{-20} for distance-squared calculations.

36. Photon problem to test fix for source sampling of position from a distribution that exceeds the extents of the underlying cell – hence cell rejection. Problem setup generated by Attila4MC; 3 parts; 9014 1st order tets.

37. Multiple mesh test problem using the same 8-hex geometry of 1007. Simple cube model used 3 times in 3 translated mesh universes. 8 1st order hex elements; 1 per octant. Each element is a statistical set, 8 total. Fixed source, ray tracing from inward directed surface source. Tally and edit results should be the element volume × source fraction. Tests 4 particles & 3 tracking loops: neutral particle low energy (neutrons & photons), low mass charged (electrons), heavy mass charged (protons).

38. Proton energy deposition using the 8-hex geometry from 1007.

39. Kobayashi benchmark-like. Point detector ray-tracing – traces parallel to tet faces (initially causing element selection confusion); 3 parts / 3 pseudo-cells; 228 1st order tets; volume source.

40. Ueki benchmark-like. Point detector. Neutron. 1 pseudo-cell, but 9 parts in file. Checks revised transm routine.

41. Multiple mesh test problem; peg w/ triangular cross section (448 1st order hexs, uranium fuel) inside square donut (1206 1st order hexs, concrete); point source in peg.

42. Bank fill test when fissile material is in overlap region; 1st order hexs; Pu-box – 216 hexs; poly shell – 1304 hexs; neutron fixed source.

43. Same as 1001 except generates `MCNPUM` file um1044.inp. `GMV` file is really `MCNPUM` file to compare against the template.

44. Same as 1001 except uses `MCNPUM` file generated by 1043.

45. Continue run of 1044 using the `MCNPUM` file.

46. Same as 1012. Testing volume source with `MCNPUM` file. Testing `MCNPUM` as `GMV`.

47. Same as 1020. Testing volume sources with `MCNPUM` file. Testing `MCNPUM` as `GMV`.

48. Same as 1037. Testing multiple UM mesh files with 2 being Abaqus and 1 being `MCNPUM`. Testing `MCNPUM` as `GMV`.

49. Flux multiplier test problem: eigenvalue problem with neutrons & photons; tests neutron flux, fission power, photon tracks, photon population by comparing to F4 tallies; tests neutron source by comparing to `FMESH` tally.

50. Flux multiplier test problem: fixed source problem neutrons only; comparing isotopic UM edits with `FMESH` tallies like those presented in LA-UR-10-06217 example.

51. Flux multiplier test problem: fixed source; tests proton heating reaction, tracks, population by comparing to `F4` tallies; tests proton source by comparing to `FMESH` tally.

52. Flux multiplier test problem: fixed source; tests electron flux multiplier factor only by comparing to `F4` tallies.

53. Continue run of 1047. Testing volume source continue run with `MCNPUM` file.

54. Energy deposition verification for protons; use 1007 & 1008 geometry; zinc target

55. Energy deposition verification for electrons; use 1007 & 1008 geometry; zinc target

56. Testing extraction of pseudo-cells from a part. Part #1 is 8-element hex cube where each element is a pseudo-cell. Part #2 is 8-element cube where there is 1 pseudo-cell in the part. Part #2 is used in 1st and 3rd instance. Part #1 is used in the 2nd instance.

57. Kobayashi benchmark. 17430 linear tets. Elsets with non-contiguous elements.

58. Godiva-like criticality problem; 896 1st order hexs; U-235; tests new top level tracking routine where the source particle is on the surface of an element headed out – 5 surfaces have intersection values of -1 and 1 surface has value of 0, resulting in nextEl being set to -2.

59. Flux multiplier. Same as 1050. Tests that code can set up the necessary material / cross section info from input on `EMBEE` cards and not the fm cards.

60. Fixed source neutron. Berp ball-like. Tests volume source selection in overlap region of fissile and non-fissile material.

61. Fixed source photon source. 1008 geometry. Testing photonuclear flux multiplier on edits.

62. UM hex equivalent of advanced variance reduction class problem – concrete duct w/ penetration; mode n p; photon surface tally; multiple volume source; ensure sources don't start in void cell; many parts.

63. Similar to 1035 – modified hemi-head problem; parts w/ multiple pseudo-cells w/ element numbers out of order; tests global element number from skdtree search using instFind trees; mode n.

12.2 Publications & Reports

This section is a list of Verification & Validation (V&V) publications and reports on the UM; some are available on the mcnp.lanl.gov website.

12.2.1 Peer Reviewed Publications

- Roger L. Martz, "MCNP6 Unstructured Mesh Initial Validation and Performance Results," Nuclear Technology, Vol 180 (Dec 2011).

- Roger L. Martz and Kevin M. Marshall, "A Notable Comparison of Computation Geometries in MCNP6 Calculations," Nuclear Technology, Vol 184 (Nov 2013).

- Joel A. Kulesza and Roger L. Martz, "Evaluation of the Kobayashi Analytical Benchmark Using MCNP6's Unstructured Mesh Capabilities," Nuclear Technology, Vol 195, (July 2016).

- Joel A. Kulesza and Roger L. Martz, "Evaluation of Pulsed Sphere Time-of-Flight and Neutron Attenuation Experimental Benchmarks Using MCNP6's Unstructured Mesh Capabilities," Nuclear Technology, Vol 195, (July 2016).

- Joel A. Kulesza and Roger L. Martz, "Evaluation of the Pool Critical Assembly Benchmark with Explicitly Modeled Geometry using MCNP6's Unstructured Mesh Capabilities," to be published in Proceedings of Sixteenth International Symposium on Reactor Dosimetry (ISRD16), Santa Fe, New Mexico, May 7–12, 2017.

12.2.2 Los Alamos National Laboratory Reports

- Roger L. Martz and David L. Crane, "Accurate Volume Calculations for Unstructured Mesh Elements When Used to Model Primitive Objects," Los Alamos National Laboratory report LA-UR-10-02144 (2010) for American Nuclear Society RPSD 2010 Meeting, Las Vegas, NV, April 18-23, 2010, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2010).

- Karen C. Kelley, Roger L. Martz, and David L. Crane, "Riding Bare-Back on Unstructured Mesh for 21st Century Criticality Calculations," Los Alamos National Laboratory report LA-UR-09-7320 (2010) for American Nuclear Society PHYSOR 2010 Meeting, Pittsburgh, PA, May 9-14, 2010, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2010).

- Timothy P. Burke, et al., "Reactor Physics Verification of the MCNP6 Unstructured Mesh Capability," Los Alamos National Laboratory report LA-UR-12-24277 (2012) for American Nuclear Society International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, ID, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013).

- Roger L. Martz, "Flux Multiplier Capability for MCNP6 Unstructured Mesh Feature", Los Alamos National Laboratory report LA-UR-16-22004 (2016).

- Roger L. Martz, "Verification of the Multi-Mesh Capability for MCNP6s Unstructured Mesh Feature", Los Alamos National Laboratory report LA-UR-16-23111 (2016).

- Roger L. Martz, "The MCNPUM Capability for MCNP6s Unstructured Mesh Feature", Los Alamos National Laboratory report LA-UR-16-23286 (2016).

- Roger L. Martz, "Verification of Charged Particle Energy Deposition Edits for MCNP6s Unstructured Mesh Feature", Los Alamos National Laboratory report LA-UR-16-23771 (2016).

Chapter 13

The UM_POST_OP Utility Program

13.1 Introduction

The *um_post_op* (unstructured mesh post operations) program is a utility program that performs various manipulations on MCNP6's elemental edit output file, `EEOUT`. This program is written in Fortran and uses various routines and data structures from the Revised Extended Grid Library (REGL) in order to maintain consistency with MCNP6. Like MCNP6, *um_post_op* is designed to run from the command line. Current supported features include adding and merging multiple `EEOUT` files into one, converting binary files to ASCII, generating Visualization TooKit *(VTK)* visualization files, creating instance-based pseudo-tallies, writing a single edit to a file, and generating error histograms for those edits with errors. Some of these features support the processing of multiple files with one command.

13.2 Valid Command Line Options

To be reminded of *um_post_op's* functionality and to see the command line options, enter the following at the command line prompt:

```
um_post_op --help
```

Note, your path must include the path to the program. A message similar to the following should appear:

```
** UTILITY PROGRAM FOR UNSTRUCTURED MESH EEOUT FILE **
Functions:



1) add many eeout files into one
2) merge many eeout files into one
3) convert binary files into ascii files
4) generate vtk files for VisIt visualization
5) generate pseudo-tallies by pseudo-cell
6) write a single edit to an ascii file
7) generate a histogram of edit errors
```

```
Command Line Arguments:
-h,   --help        summary of features & arguments
-a,   --add         add multiple files (no weighting)
-m,   --merge       merge multiple files
-o,   --output      single output file name
-p,   --pos         value range for wse and wsep
-bc,  --binconvert  convert binary file to ascii
-eh,  --errorhist   generate a histogram of edit errors
-ex,  --extension   multiple output file extension
-ta,  --tally       pseudo-tallies from file
-vtk, --vtkfile     generate ascii visualization file
-wse, --writesedit  write a single edit to file
```

13.3 Mutually Exclusive Options

This utility program has seven mutually exclusive options: merging (-m) many files into one ASCII file, adding (-a) many files together into one ASCII file, converting (-bc) any number of binary files into ASCII files, generating VTK files (-vtk) for visualization, generating pseudo-tallies (-ta) for instances, writing a single edit (-wse) to an ASCII file, and generating a histogram of edit errors (-eh) for those edits that have errors. Only one of these options may be requested at a time.

13.4 The -o and -ex Options

The output file name (-o, --output) and extension name (-ex, --extension) options are intended to be mutually exclusive. The user should receive error messages if both of these arguments appear on the same command line. However, one or the other must be used. The output file name is intended for use when there is one `EEOUT` file to manipulate or many files that are to be merged into one. The extension name is pre-appended with a period, '.', and then appears as the suffix to the input file name(s) when new files must be created after processing many input files (e.g., converting many files from ASCII to binary). The first argument following these arguments is interpreted as either the output file name or the extension name.

13.5 Merging Files

The original intent for this utility program was to establish a means of merging many `EEOUT` files into one file. These many files are expected to be from independent runs of a problem so that results are weighted by the number of histories in the file. This differs from adding files where there is no history weighting.

When the *um_post_op* utility is given a list of files to merge into one, it reads the header information (that includes number of nodes, materials, instances, tetrahedra, pentahedra, hexahedra) and checks the consistency of this header information for each subsequent file against the first file. For all files other than the first one, a message about that consistency is output to

the terminal window. Without consistency among the files, the utility program can not make a meaningful and successful merge.

If there is only one file specified for merging, the program will print out an error message and stop. Since one file is created from many, the output file name argument is required.

Example command line:

```
um_post_op -m -o my_merge_file eeout1 eeout2 ... eeoutN
```

Note that the first argument after the -o argument is interpreted as the output file name.

At this time, the output file that is generated is ASCII, even if all of the input files are binary. The input files may be any mixture of ASCII or binary.

13.6 Adding Files

This capability provides a means of adding (or collecting) many `EEOUT` files into one file. These many files are expected to be from different calculational runs on the same mesh geometry; results are NOT weighted by the number of histories in the file. Rather, already normalized results are simply added together. This differs from merging files where there is history weighting. For example, this capability is useful if there are different runs because independent sources were used in different calculations and there is a need for the results to be combined.

Cautions and restrictions discussed under the merging files section apply here and are not repeated.

Example command line:

```
um_post_op -a -o my_add_file eeout1 eeout2 ... eeoutN
```

Note that the first argument after the -o argument is interpreted as the output file name.

13.7 Converting Files

This capability allows the conversion of `EEOUT` files from binary format to ASCII. In performing this operation there is a loss of precision since all double precision reals are written with only six significant digits. Currently, there is no capability to convert from ASCII to binary.

On the command line, one or many files may be specified for conversion. When many files are requested for conversion, there is no consistency check performed as there is when merging files since that is a meaningless action for this option.

When the conversion request asks for only one file, the -o argument may be used. Example command line:

```
um_post_op -bc -o eeout.ascii eeout.binary
```

It is also legitimate to use the -ex argument. Example command line:

```
um_post_op -bc -ex ascii eeout.binary
```

The resulting output file is named: eeout.binary.ascii

When more than one file is to be converted, the -ex argument must be used. Example command line:

```
um_post_op -bc -ex asc eeout1 eeout2 ... eeoutN
```

The resulting files appear with the names

```
eeout1.asc eeout2.asc ... eeoutN.asc
```

13.8 Visualization Files

This capability should generate files in the VTK format for visualization from `EEOUT` files. The geometry data and the edit information is taken from the `EEOUT` file and reformatted to be consistent with version 4.2 of the *VTK* standard and written to an ASCII file. Details on the *VTK* file format and requirements can be found in the *VTK* documentation, available on the worldwide web and in text books.

On the command line, one or many files may be specified for conversion to the VTK format. When many files are requested for conversion, there is no consistency check performed as there is when merging is requested since that is a meaningless action for this option.

When the generation request asks for only one file, the `-o` argument may be used.

Example command line:

```
um_post_op -vtk -o eeout.vtk eeout1
```

It is also legitimate to use the -ex argument. Example command line:

```
um_post_op -vtk -ex vtk eeout1
```

The resulting output file is named: eeout1.vtk

When more than one file is to be generated, the -ex argument must be used. Example command line:

```
um_post_op -vtk -ex vtk eeout1 eeout2 ... eeoutN
```

The resulting files appear with the names

```
eeout1.vtk eeout2.vtk ... eeoutN.vtk
```

Note that while it is possible to specify any file extension or output file name for the *VTK* file, some visualization programs will not recognize it as such unless there is a *VTK* extension.

Note that this capability has not received extensive testing and may not be supported in the future.

13.9 Generating Pseudo-Tallies

This capability will generate a pseudo-tally for each pseudo-cell from the corresponding edit and write the results to an output file (see example at the end of this chapter). If no output file is

specified, the output is written to a file named "fort.1001". These tallies are volume weighted according to the following equation:

$$tally_i = \frac{\sum_{n=1}^{N} vol_n \cdot edit_n}{\sum_{n=1}^{N} vol_n}$$

where

$tally_{\mathrm{i}}$	tally for pseudo-cell i form corresponding edit
vol_{n}	volume of element n
$edit_{\mathrm{n}}$	edit result of element n
N	total number of elements in i

These results are termed pseudo-tallies since they are equivalent to an MCNP tally averaged over a cell (*i.e.,* F4, F6, F7), but do not have an associated statistical uncertainty, tally fluctuation chart, etc. Note that these pseudo-tallies are over pseudo-cells.

On the command line one or many files may be specified for pseudo-tally creation. When many files are requested for pseudo-tally creation, there is no consistency check performed as there is when merging files since that is a meaningless action for this option.

When the conversion request asks for only one file, the -o argument may be used.

Example command line:

```
um_post_op -ta -o eeout.tally eeout.binary
```

It is also legitimate to use the -ex argument.

13.10 Writing A Single Edit To A File

This capability allows the user to write the edit results from a single edit in the `EEOUT` file (see example at the end of this chapter) to an ASCII file that is reformatted with detailed information. For each element in the problem (`EEOUT` file) the information that is available with each edit result is element number, element type number, material number, density, volume, and centroid location. The utility of this file is left to the imagination of the user. Results are ordered by increasing element number.

This request requires that an edit number be specified with the *um_post_op* command line argument, -wse or --writesedit; this number should be the argument immediately following this keyword argument. The correct edit number can be found in the output from the pseudo-tally option (see example at the end of this chapter for edit numbers in blue font), described previously. Since an edit my contain multiple energy, time, and particle bins, using the internal edit number requires less input on the *um_post_op* command line.

Example command line:

```
um_post_op -wse 1 -o eeout.wse eeout1
```

It is also legitimate to use the -ex argument.

It is possible to filter the output for this capability using the -p or --pos arguments. If the value following this argument is 1 or +1, only values greater than zero are included in the edit. Conversely, if the value following the argument is -1, only values less than or equal to zero are included. If a real value is specified instead of the integers just described, its value is the decision point with the sign of the value indicating whether the filter provides values greater than (+) or less than or equal to (−).

Example command line requesting to see all results less than or equal to 0.005.

```
um_post_op -wse 1 -p -5.e-3 -o eeout.wse eeout1
```

13.11 Writing A Single Edit To A File By Position

This capability is similar to that discussed in the previous section, except that the output is ordered by increasing position (i.e., x, y, z location). The appropriate arguments to use on the command line are: -wsep or --writeseditpos. Value filtering, as described in the previous section, works the same way with this capability.

13.12 Generating A Histogram Of Edit Errors

This capability allows the user to write error histograms to an output file for all of the edits in the `EEOUT` file for which errors were requested (see example at the end of this chapter). If no output file is specified, the output is written to a file named "fort.1001". The number of histogram bins can be specified directly after the -eh command line option. The default value is 10 if none is specified. The error bins are defined such that the smallest error is assigned to the first bin and the largest error is assigned to last bin. Bins are evenly spaced between the first and the last bins. Relative error values are in the range of 0 to 1, inclusive. See Chapter 4 for more details on errors and the `EEOUT` file.

The essential header information from the `EEOUT` file is written at the beginning of the error histogram file. Following this information, there is a section for each edit for which errors were requested. There is a description of each edit. In each section following the edit description, there are results by pseudo-cell and results over all mesh in the model. For each group of results there is the minimum and maximum errors on the edit in addition to a table with the error histogram. For each row in the histogram table there is the upper limit for the error bin, the absolute number of elements that fall into this bin, the relative percentage these elements represent of the total, and the cumulative percentage of the current row and all preceding rows.

Example command line specifying a table with 20 bins:

```
um_post_op -eh 20 -o my_error_histogram eeout1
```

It is also legitimate to use the -ex argument.

13.13 Miscellaneous

The REGL routine that reads valid `EEOUT` files has the ability to detect whether the file it is reading is ASCII or binary. If it can't make a determination that the file is a valid `EEOUT` file,

an error message appears in the terminal window. Therefore, when a list of files is specified on the command line, for either merging, adding, or generating VTK files, they may be a mixture of ASCII or binary.

13.14 Example Pseudo-Tally File

What follows is not a complete example. Only enough details are provided to illustrate the main points.

```
 Pseudo-tallies for eeout file via um_post_op
 Eeout file: eeout1007

 Created on    :  4- 3-2012 @  9: 0:37

 Prob ID       : simple cube, each element is a statistical set, 8 total
 Calling Code  : MCNP6
 Inp File      : inp1007
 Outp File     : inp1007o
 Runtpe File   : inp1007r
 Geom Inp File : um1007.inp

 NUMBER OF NODES    :                   27
 NUMBER OF MATERIALS:                    1
 NUMBER OF INSTANCES:                    1
 NUMBER OF 1st TETS :                    0
 NUMBER OF 1st PENTS:                    0
 NUMBER OF 1st HEXS :                    8
 NUMBER OF 2nd TETS :                    0
 NUMBER OF 2nd PENTS:                    0
 NUMBER OF 2nd HEXS :                    0
 NUMBER OF COMPOSITS:                    1
 NUMBER OF HISTORIES:                 1000
 NUMBER OF REG EDITS:                   19
 NUMBER OF COM EDITS:                    9

 ------------------------------------------------------------------------
 EDIT:    1 :: TALLY for  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_1_FLUX_14

  Energy Bin Boundary:  1.00000E+36  Energy Bin Multiplier:  1.00000E+00
  Time Bin Boundary  :  1.00000E+33  Time Bin Multiplier  :  1.00000E+00

 Instance  Name                                Volume        Result
 --------  ----                                ------        ------
        1  simple_cube-1                    1.00000E+03   4.77743E-02

 ------------------------------------------------------------------------
EDIT:    2 :: TALLY for  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_1_ENERGY_36

  Energy Bin Boundary:  2.00000E+00  Energy Bin Multiplier:  1.00000E+00
  Time Bin Boundary  :  1.00000E+00  Time Bin Multiplier  :  1.00000E+00

 Instance  Name                                Volume        Result
 --------  ----                                ------        ------
        1  simple_cube-1                    1.00000E+03   8.12612E-03

 ------------------------------------------------------------------------
EDIT:    3 :: TALLY for  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_2_ENERGY_36

  Energy Bin Boundary:  1.00000E+10  Energy Bin Multiplier:  1.00000E+00
  Time Bin Boundary  :  1.00000E+00  Time Bin Multiplier  :  1.00000E+00

```

```
 Instance  Name                                    Volume        Result
 --------  ----                                    ------        ------
        1  simple_cube-1                       1.00000E+03   7.54778E-03

-------------------------------------------------------------------------------
 EDIT:    4 :: TALLY for  EDIT__PARTICLE_1__TIME_BIN_2_ENERGY_BIN_1_ENERGY_36

  Energy Bin Boundary:  2.00000E+00  Energy Bin Multiplier:  1.00000E+00
  Time Bin Boundary  :  1.00000E+39  Time Bin Multiplier  :  1.00000E+00

 Instance  Name                                    Volume        Result
 --------  ----                                    ------        ------
        1  simple_cube-1                       1.00000E+03   7.84947E-03

 -------------------------------------------------------------------------------
EDIT:    5 :: TALLY for  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_2_ENERGY_36

  Energy Bin Boundary:  1.00000E+10  Energy Bin Multiplier:  1.00000E+00
  Time Bin Boundary  :  1.00000E+39  Time Bin Multiplier  :  1.00000E+00

 Instance  Name                                    Volume        Result
 --------  ----                                    ------        ------
        1  simple_cube-1                       1.00000E+03   2.26368E-03
```

13.15 Example Single Edit File

```
Write single edit for eeout file via um_post_op
Eeout file: eeout1007

Created on    :  4- 3-2012 @ 12:11:25

Prob ID       : simple cube, each element is a statistical set, 8 total
Calling Code  : MCNP6
Inp File      : inp1007
Outp File     : inp1007o
Runtpe File   : inp1007r
Geom Inp File : um1007.inp

NUMBER OF NODES    :                   27
NUMBER OF MATERIALS:                    1
NUMBER OF INSTANCES:                    1
NUMBER OF 1st TETS :                    0
NUMBER OF 1st PENTS:                    0
NUMBER OF 1st HEXS :                    8
NUMBER OF 2nd TETS :                    0
NUMBER OF 2nd PENTS:                    0
NUMBER OF 2nd HEXS :                    0
NUMBER OF COMPOSITS:                    1
NUMBER OF HISTORIES:                 1000
NUMBER OF REG EDITS:                   19
NUMBER OF COM EDITS:                    9

---------------------------------------------------------------
EDIT:    1 ::  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_1_FLUX_14

 Energy Bin Boundary:  1.00000E+36  Energy Bin Multiplier:  1.00000E+00
 Time Bin Boundary  :  1.00000E+33  Time Bin Multiplier  :  1.00000E+00

-------------------------------------------------------------------------------------------------------------
Element   Type  Material     Density       Volume                    Centroid                      Result
                                                                X    Y            Z
--------  ----  --------   -----------   -----------  ------------  ------------  ------------   -----------
       1    6          1   1.87401E+01   1.25000E+02  -2.50000E+00  -2.50000E+00   7.50000E+00   4.50075E-02
       2    6          1   1.87401E+01   1.25000E+02  -2.50000E+00   2.50000E+00   7.50000E+00   4.71156E-02
       3    6          1   1.87401E+01   1.25000E+02  -2.50000E+00  -2.50000E+00   2.50000E+00   4.99385E-02
       4    6          1   1.87401E+01   1.25000E+02  -2.50000E+00   2.50000E+00   2.50000E+00   4.99248E-02
       5    6          1   1.87401E+01   1.25000E+02   2.50000E+00  -2.50000E+00   7.50000E+00   4.59879E-02
       6    6          1   1.87401E+01   1.25000E+02   2.50000E+00   2.50000E+00   7.50000E+00   5.14196E-02
       7    6          1   1.87401E+01   1.25000E+02   2.50000E+00  -2.50000E+00   2.50000E+00   4.33516E-02
       8    6          1   1.87401E+01   1.25000E+02   2.50000E+00   2.50000E+00   2.50000E+00   4.94486E-02
```

13.16 Example Error Histogram File

```
Write error histograms for eeout file via um_post_op
Eeout file: block01_6part_6type.eeout

Created on    :  3-11-2014 @ 13: 8:21

Prob ID       : block01  8x8x6  6 parts, 6 element types
Calling Code  : MCNP6_DEVEL
Code Version  : 6-1-02
Date & Time   :  03/11/14 12.43.38
Inp File      : block01mgv1
Outp File     : outy
Runtpe File   : runtpn
Geom Inp File : job_block_6part_6type_01.inp

NUMBER OF NODES    :                1258
NUMBER OF MATERIALS:                   6
NUMBER OF INSTANCES:                   6
NUMBER OF 1st TETS :                  30
NUMBER OF 1st PENTS:                   8
NUMBER OF 1st HEXS :                 128
NUMBER OF 2nd TETS :                  29
NUMBER OF 2nd PENTS:                   8
NUMBER OF 2nd HEXS :                 128
NUMBER OF COMPOSITS:                   0
NUMBER OF HISTORIES:             1000000
NUMBER OF REG EDITS:                   2
NUMBER OF COM EDITS:                   0

------------------------------------------------------------
EDIT:  EDIT__PARTICLE_1__TIME_BIN_1_ENERGY_BIN_1_FLUX_4

    Energy Bin Boundary:  1.00000E+10  Energy Bin Multiplier:  1.00000E+00
    Time Bin Boundary  :  1.00000E+39  Time Bin Multiplier  :  1.00000E+00

    ----------------------------------------------------
    Results for Instance #     1  :: part-end_quad_hex-1

    Minmum Error       :  1.64393E-02
    Maximum Error      :  1.70379E-02
    Bin Width          :  2.99308E-05

    ------   --------   --------   --------    ----------
    Bin       Upper     Absolute   Relative    Cumulative
    Number    Bound      Number       (%)        (%)
    ------   --------   --------   --------    ----------
       1   1.6469E-02          1     0.7812        0.7812
       2   1.6499E-02          1     0.7812        1.5625
       3   1.6529E-02          3     2.3438        3.9062
       4   1.6559E-02          5     3.9062        7.8125
       5   1.6589E-02          0     0.0000        7.8125
       6   1.6619E-02          7     5.4688       13.2812
       7   1.6649E-02          6     4.6875       17.9688
       8   1.6679E-02         14    10.9375       28.9062
       9   1.6709E-02          5     3.9062       32.8125
      10   1.6739E-02          6     4.6875       37.5000
      11   1.6769E-02         13    10.1562       47.6562
      12   1.6798E-02         14    10.9375       58.5938
      13   1.6828E-02         12     9.3750       67.9688
      14   1.6858E-02         11     8.5938       76.5625
      15   1.6888E-02          5     3.9062       80.4688
      16   1.6918E-02         10     7.8125       88.2812
```

```
      17   1.6948E-02          4     3.1250       91.4062
      18   1.6978E-02          7     5.4688       96.8750
      19   1.7008E-02          3     2.3438       99.2188
      20   1.7038E-02          1     0.7812      100.0000

(Results for instances 2 through 6 were removed to make this example shorter.)

    ---------------------
    Results Over All Mesh

    Minmum Error       :  9.33224E-03
    Maximum Error      :  1.95299E-02
    Bin Width          :  5.09881E-04

    ------   --------   --------   --------    ----------
    Bin       Upper     Absolute   Relative    Cumulative
    Number    Bound      Number      (%)         (%)
    ------   --------   --------   --------    ----------
       1   9.8421E-03          4     1.2085        1.2085
       2   1.0352E-02          8     2.4169        3.6254
       3   1.0862E-02          0     0.0000        3.6254
       4   1.1372E-02          0     0.0000        3.6254
       5   1.1882E-02          4     1.2085        4.8338
       6   1.2392E-02          1     0.3021        5.1360
       7   1.2901E-02          0     0.0000        5.1360
       8   1.3411E-02          3     0.9063        6.0423
       9   1.3921E-02          3     0.9063        6.9486
      10   1.4431E-02          9     2.7190        9.6677
      11   1.4941E-02          9     2.7190       12.3867
      12   1.5451E-02          4     1.2085       13.5952
      13   1.5961E-02          0     0.0000       13.5952
      14   1.6471E-02         15     4.5317       18.1269
      15   1.6980E-02        241    72.8097       90.9366
      16   1.7490E-02         18     5.4381       96.3746
      17   1.8000E-02          5     1.5106       97.8852
      18   1.8510E-02          6     1.8127       99.6979
      19   1.9020E-02          0     0.0000       99.6979
      20   1.9530E-02          1     0.3021      100.0000
```

This page intentionally left blank.

Chapter 14

The UM_PRE_OP Utility Program

14.1 Introduction

The *um_pre_op* (unstructured mesh pre operations) program is a utility program that performs various manipulations on input designed to aid in problem setup with the unstructured mesh (UM). This program is written in Fortran and uses various routines and data structures from the Revised Extended Grid Library (REGL) in order to maintain consistency with MCNP6. Like MCNP6, *um_pre_op* is designed to run from the command line. Current supported features include creating a skeleton MCNP6 input deck (`-m`) from the Abaqus/CAE .inp file, converting a simple lattice-voxel geometry (`-lc`) to an Abaqus .inp file, volume checking (`-vc`) the finite element volumes, and element checking (`-ec`) the .inp file for twisted and/or deformed elements. As with *um_post_op*, there is limited error handling.

14.2 Valid Command Line Options

To be reminded of *um_pre_op's* functionality and to see the command line options, enter the following at the command line prompt:

```
um_pre_op --help
```

Note, your path must include the path to the program. A message similar to the following should appear:

```
** PRE-PROCESSOR PROGRAM FOR UM CAPABILITY **



Functions:
1) Create MCNP input file from Abaqus .inp file
2) Convert MCNP simple lattice to Abaqus .inp file
3) Volume check the Abaqus .inp file and pseudo-cells
4) Element check the Abaqus .inp file
```

```
Command Line Arguments:
-b,   --back        background material for input file
-h,   --help        summary of features & arguments
-m,   --mcnp        generate MCNP skeleton input file --(1)
-o,   --output      output file name



-cf,  --controlfile  file with lattice conversion controls
-dc,  --datacards    data cards file to include
-ex,  --extension    output file extension
-ff,  --fillfile     file with lattice fill description
-lc,  --latconvert   convert simple lattice to Abaqus -- (2)
-vc,  --volcheck     volume check the .inp file       -- (3)
-ec,  --elementcheck element check the .inp file      -- (4)
-len, --length       scale factor for mesh dimensions
```

14.3 Mutually Exclusive Options

Currently, this utility program has four mutually exclusive options: generating (-m) a skeleton MCNP6 input file, converting (`-lc`) a simple lattice-voxel geometry to an Abaqus .inp file, volume checking (`-vc`) the finite element volumes, and element checking (`-vc`) for twisted and/or deformed elements.

14.4 The -o and -ex Options

The output file name (`-o, --output`) and extension name (`-ex, --extension`) options are intended to be mutually exclusive. The user should receive error messages if both of these arguments appear on the same command line. However, one or the other must be used except where indicated in the following feature discussions. If the `-o` argument is present then the output is placed in a file with the name (or argument) that immediately follows on the command line. If the `-ex` argument is present, then the output is placed in a file with a name built from the input file name followed by a period, '.', and the argument immediately following on the command line.

14.5 The -b Option

The -b option is currently only used with the -m option to specify a background cell material number. See the discussion below for more information.

14.6 The -len Option

The -len option is currently only used with the -lc option. This is a scale factor to apply to dimensions from the lattice mesh file.

14.7 Generating an MCNP6 Input File

A skeleton MCNP6 input file can be created from the Abaqus .inp file using the `-m` option. The name of the input file to be created is set with either the `-o` or `-ex` options. The intent of this option is to make it easier for users to get up and running with the unstructured mesh capability and not necessarily to generate a fully functional input file. The degree to which a fully functional input deck can be generated depends upon the completeness and correctness of the data card file provided with the `-dc` option.

The *um_pre_op* program can read the Abaqus .inp file and generate a global mesh model just as if MCNP6 was performing this function. The information in the global mesh model is then used to create the appropriate pseudo-cell cards, background cell, and minimal CSG world to hold the mesh universe plus the `embed` control card for the data section. If more than a minimal CSG structure is required outside the mesh universe, the user must create this by hand.

If the `-b` option is not specified on the command line to supply a valid material number from the Abaqus .inp file, *um_pre_op* will make the background cell void. If an invalid material number (i.e., a number for a material that is not defined in the .inp file) is specified with the `-b` option, *um_pre_op* will default to making the background cell void. At this time the `-b` option only works with the `-m` option.

When using the `-m` option it is possible to read a data cards file, `-dc` argument, for inclusion in the new MCNP6 input file. The *um_pre_op* program scans the data cards file for existing cards. For each particle on an existing and active mode card, a default flux edit (`embee` card) is specified and written to the new input file. If active `IMP` cards are present in the data cards file, they are written to the new input file, otherwise *um_pre_op* creates default `IMP` cards for each particle present on the mode card. If an active `SDEF` card is present in the data cards file, it is written to the new input file, otherwise a skeleton `SDEF` card is written provided volume source elsets are present in the .inp file. All other cards in the data cards file, regardless if they are active cards or comments, are written to the new input file.

Note: At this point the material numbers for the material definitions in the data cards file should be consistent with those used in the Abaqus .inp file. This may be the biggest source of error for some users.

Example command line with data cards argument and the `-b` argument to use material 7 from the .inp file as the background material for the mesh universe:

```
um_pre_op --mcnp -o newinput abaqus.inp -dc dc_cards -b 7
```

14.8 Converting a Simple Lattice Geometry

Simple lattice geometries in MCNP6 that use the `fill` parameter along with the `lat` parameter on a cell card can be converted to an Abaqus .inp file for use with the -m option described previously or for viewing as an orphan mesh geometry in Abaqus. This lattice geometry is described as simple in that each voxel should have a homogenous structure since each voxel is converted to a first order hexahedra with a homogeneous material assignment.

For this feature, two input files are required and the .inp file must be specified using the `-o` option; the `-ex` option is invalid here. In addition, a file named `lat2abq.summary` is created that contains details about the conversion process. The first of the two input files must contain only

the fill information as it appears with the `fill` parameter on the MCNP6 lattice cell card. A short example is given in Figure 14.1. This is known as the fill file and is specified to *um_pre_op* with the `-ff` option. Any attempt to put other information is this file will undoubtedly cause *um_pre_op* to terminate in an unfriendly manner.

Figure 14.1: Example fill file.

```
1 19R
2 7r  3 11R
2 2 4 2r
2 2 4 2r
3 4 3R 3 4 3R
```

The second of the two required input files is the control file and is specified to the *um_pre_op* program with the `-cf` option. An example is provided in Figure 14.2. As can be seen from the description of this file that follows, there are a number of parameters that can be adjusted for this feature, making it tedious to implement and use as command line options.

The first line in the control file is the title line. The line is required, must be the first line in the file, and can contain 256 characters of information. This line is inserted in the Abaqus .inp file on the line after the `*Heading` parameter at the beginning of the file. This is the line that is used for the MCNP6 inp file title line if the *um_pre_op* -m option is invoked.

Any line after the first line with either a #, %, or $ in the first column is treated as a comment line by *um_pre_op* and ignored.

All of the other parameters for this feature are implemented with a set of keywords where the keyword appears at the beginning of the line before any values. The keywords do not need to start in the first column; they can be either upper case, lower case, or a mixture of both. Most keywords have default values. Those that do not have defaults are required keywords and should contain meaningful data.

The `deltas` keyword is required. Three values are needed that specify the length of the voxels in centimeters along the x, y, and z directions. These values will be used to size the hexahedra. All hexahedra will have these dimensions.

The `fill` keyword is required. Three sets of values for the x, y, and z directions are needed in the same format that MCNP6 requires for this keyword on the lattice cell card. Each set consists of two lattice locations separated by a colon. The value to the left of the colon is the smallest index for that direction (for *um_pre_op* this value should be 0) while the value to the right of the colon is the largest index for that direction. The values specified for the `fill` keyword should be the full extents of the problem described in the fill file. A subset of this geometry can be specified with the `extents` parameter described below.

The `universe` keyword is required. There may be as many universes specified on separate lines in the control file as needed to fully describe the problem. For the sake of *um_pre_op* and converting a lattice description to an equivalent unstructured mesh equivalent, the concept of a universe is more restrictive than what MCNP6 allows in general. As stated above, each voxel in the lattice must be homogeneous so that one material can be assigned to it. Therefore, the universe numbers double as material numbers. If the universe and material numbers don't coincide in the existing description, it is up to the user to ensure that they do coincide (are

Figure 14.2: Example control file

```
Jacksonville 1000 x 1000 x 31 model; 1 meter resolution
Deltas 100 100 100
fill 0:999 0:999 0:30
Origin center
#
universe 1 -1.25000E-03 air
universe 2 -0.05 ext_building
universe 3 -0.01 int_building
universe 4 -1.2 ground
universe 5 -0.01 int_garage
universe 6 -0.087058 ext_garage
universe 7 -0.00125 air
#
exclude 1
extents 0 999 0 999 0 0
hints 200 200 50
threshhold 1
```

identical). If the user wishes to convert a more complex voxelized lattice to unstructured mesh, the complex voxels must be homogenized.

Three values are required for each `universe` keyword. The first is the universe number. There should be one for every universe number that is used in the fill file. The universe numbers will be used as the material numbers when describing the material elsets in the Abaqus .inp file. There is no default value for the universe number; so valid input is required. The second value for the `universe` keyword is the material density (either number or physical). This value will be written to the pseudo-cell cards if *um_pre_op* is used with the `-m` option on this file. The third value for this keyword, is the universe / material name that can contain as many as 128 alphanumeric characters. This name is used in creating material and part names. More information on the parts created in this process can be found in the discussion for the `hints` keyword.

The following keywords are optional.

The `exclude` keyword is optional. It contains a single integer instructing *um_pre_op* to exclude the specified universe number from any of the parts. In the Figure 14.2 example, `universe 1` is part of the simple lattice, but because it is air that we don't want MCNP6 to track through as a mesh, we exclude it. This can save computation time, but will not let the program accumulate results on a mesh in these locations. When excluding any universe, it is probably a good idea to set the background material for the mesh universe to this material; see the `-b` option in conjunction with the `-m` option..

The `extents` keyword is optional and is used to select a contiguous extent of the lattice specified from the `fill` keyword. Default values are 0, but any values specified are taken to apply in the order lower x-index, upper x-index, lower y-index, upper y-index, lower z-index, and upper z-index.

The `hints` keyword is optional, but highly recommended since values associated with this

keyword set the overall size of segments and parts. Three values, one for each direction, are permitted with the default for each being 9999999. The values are not physical units, but rather the number of columns (X), rows (Y), or planes (Z). Since MCNP6 input processing for parts in the unstructured mesh can be time intensive if the parts have more than ~50,000 elements, it is best to segment any geometry, whether it comes from a lattice or not, into smaller pieces. The values associated with this keyword provide guidance to *um_pre_op* in order to create these segments. *um_pre_op* will construct segments that are close to the size specified. Each segment has a set of *i-j-k* indices that describes its location from the lower left hand corner in the overall geometry. Once the segments are defined, the program can create parts from the segments. All elements with the same material are lumped together into a part whose name is derived from the *i-j-k* indices, the material number, and the material name. For example, a part composed of the material "ground" with an associated material number of 4 and possessing *i-j-k* indices of 2, 3, 1 would be given the name: `part_2_3_1_4_ground`.

The `origin` keyword is optional and is used to adjust the location of the mesh origin. If this keyword is not included, the origin defaults to 0 0 0, otherwise it is shifted to the value specified. An X Y Z location can be specified, or as a convenience, the characters `CENTER` may be input. With `CENTER` specified, the program calculates the problem's center based upon the overall extents specified with the `fill` and `deltas` keywords. Any triples of values causes the origin to shift to that location.

The `threshold` keyword is optional. It contains a single value instructing *um_pre_op* to make parts when the number of elements in the part exceed the value specified. The default value is 1. It is always a good idea to create parts with more than 1 element.

The information in the `lat2abq.summary` file is fairly self-explanatory. The information in this file can help the user set or adjust values for the hints keyword, among other things. It was decided that it was more appropriate to write this information to a file rather than to the terminal screen.

Example command line to convert a simple lattice geometry:

```
um_pre_op -lc -o geomlattice.inp -ff fillfile -cf control
```

14.9 Volume Checking

This option enables the user to check the finite element volumes (against a value) and obtain volumes and masses for the pseudo-cells. Results are printed to a file specified with either the `-o` or `-ex` options. See the results from the example file at the end of this chapter.

Any value appearing on the command line after the `-vc` argument is treated as the test value. If this value is positive, *um_pre_op* will print out all elements and their corresponding volumes that are greater than or equal to the specified value. If this value is negative, all elements and their corresponding volumes that are less than or equal to the specified value are printed. If no value follows the `-vc` argument, the test is for volumes less than or equal to zero.

Once the volume checks are performed on all of the finite elements, *um_pre_op* calculates the volumes and masses for all of the pseudo-cells. Masses are based on the densities that are present in the Abaqus .inp file. This information appears in the output file after the element listing from the finite element volume check. After this, a list of the instance names appears followed by a list of the material names and associated densities.

Example command line to find all finite elements with a volume less than or equal to 15:

```
um_pre_op -vc -15 -o vc.out simple_cube_warped.inp
```

14.10 Element Checking

This option enables the user to check the .inp file for deformed and/or twisted elements by calculating the determinant of the Jacobian at the appropriate Gauss points and at all node locations that define the finite element. Normal elements (i.e., not deformed or twisted) will have a positive Jacobian indicating that each point (finite volume) in the master space is mapped to an appropriate point (finite volume) in the global space (where some of the tracking algorithms operate). However, very small positive values indicate distortion in the mapping. With appropriate positive Jacobians, the volumes and masses will be correct (as modeled) and there should be no problem with the particle transport.

If a failed element is found (negative Jacobian) during the execution of this option, the global element number and appropriate location information are written to the terminal screen. This same information as well as the results for the Jacobian evaluation at each Gauss and node point are written to the file specified with either the `-o` or `-ex` options. Note that the information is organized by instance. See the results from the example file at the end of this chapter. It is the user's responsibility to fix the problem mesh with the appropriate meshing tool.

Example command line:

```
um_pre_op -ec -o warped.out simple_cube_warped.inp
```

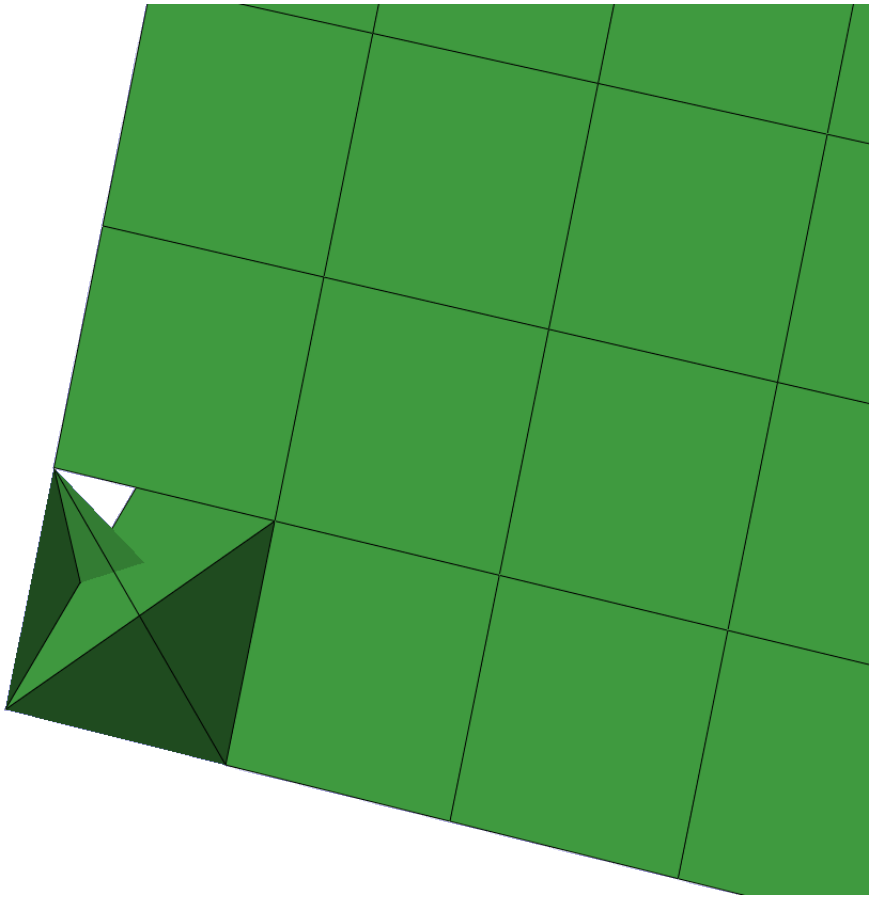


Figure 14.3: Example twisted first-order tetrahedra

14.11 Example Volume Check File

```
 simple warped cube
-
-  Data from file    : simple_cube_warped.inp
-  Created on        :  1-17-2014 @ 14: 0:58
-
    ---------------------------------------
    - Volume Check For Value  1.50000E+01 -
    ---------------------------------------

    Element       Volume
    -------    ------------
          1     7.81250E+00

    Elements with volumes <=  1.50000E+01 :        1

    ----------------------------------
    - Pseudo-Cell Volumes and Masses -
    ----------------------------------

    Cell    Instance    Part    Material    Denisty      Volume Mass
    ----    --------    ----    --------    -------   -----------   -----------
       1           1       1           1   -8.95000   9.99219E+03   8.94301E+04

     ------------------------------------------------------------------------------
    Instance    Name
    --------    ------------------------------------------------------------
          1    simple_cube-1

    ------------------------------------------------------------------------------
    Material    Denisty    Name
    --------    -------    ------------------------------------------------------
           1   -8.95000    material-copper_01
           2   -2.25000    material-graphite_02
```

14.12 Example Element Check File

```
 ---------------------------------
 - Checking Elements By Instance -
 ---------------------------------

  Number  Name
------  ------------------------------------------------------------------------------------
     1  part-cube-1

         Element:          2   failed.    Centroid:    1.50000E+00   5.00000E-01   1.50000E+00
                                            Nodes:        X    Y             Z
                                                1    2.00000E+00   1.00000E+00   1.00000E+00
                                                2    2.00000E+00   0.00000E+00   1.00000E+00
                                                3    2.00000E+00   0.00000E+00   2.00000E+00
                                                4    2.00000E+00   1.00000E+00   2.00000E+00
                                                5    1.00000E+00   1.00000E+00   1.00000E+00
                                                6    1.00000E+00   0.00000E+00   1.00000E+00
                                                7    1.00000E+00   1.00000E+00   2.00000E+00
                                                8    1.00000E+00   0.00000E+00   2.00000E+00

                                                    Determinate Values At Gauss Points
                                                            Gauss Points           Jacobian
                                                1    -0.57735  -0.57735  -0.57735   1.14E-01
                                                2     0.57735  -0.57735  -0.57735   1.14E-01
                                                3     0.57735   0.57735  -0.57735   8.33E-02
                                                4    -0.57735   0.57735  -0.57735   8.33E-02
                                                5    -0.57735  -0.57735   0.57735   8.33E-02
                                                6     0.57735  -0.57735   0.57735   8.33E-02
                                                7     0.57735   0.57735   0.57735  -3.05E-02
                                                8    -0.57735   0.57735   0.57735  -3.05E-02

                                                   Determinate Values At Master Space Nodes
                                                            Gauss Points           Jacobian
                                                1    -1.00000  -1.00000  -1.00000   1.25E-01
                                                2     1.00000  -1.00000  -1.00000   1.25E-01
                                                3     1.00000   1.00000  -1.00000   1.25E-01
                                                4    -1.00000   1.00000  -1.00000   1.25E-01
                                                5    -1.00000  -1.00000   1.00000   1.25E-01
                                                6     1.00000  -1.00000   1.00000   1.25E-01
                                                7     1.00000   1.00000   1.00000  -1.25E-01
                                                8    -1.00000   1.00000   1.00000  -1.25E-01

 Total number of failed elements:      1
```

Chapter 15

The UM_CONVERT Utility Program

15.1 Introduction

The *um_ convert* (unstructured mesh convertor) program is a command-line utility program that takes the information in the Abaqus .inp file and processes it with the UM input processing routines from REGL to produce the internal data structures that MCNP6 needs. The data from these internal data structures are written to a new file type, `MCNPUM`, that MCNP6 can quickly read before launching into calculations. With the `MCNPUM` file type the UM input processing start up penalty need not happen every time the UM geometry is required. This can save substantial time for large mesh geometries that are used repeatedly. Details on the structure of this file and its contents are best learned from looking at the source code.

15.2 Valid Command Line Options

To be reminded of *um_ convert*'s functionality and to see the command line options, enter the following at the command line prompt:

```
um_convert_op --help
```

Note, your path must include the path to the program. A message similar to the following should appear in the command window:

```
** UNSTRUCTURED MESH CONVERSION PROGRAM **
Functions:
1) Convert ABAQUS inp file to mcnpum file
Command Line Arguments:
-h,   --help         summary of features & arguments
-b,   --binary       create mcnpum in binary format
-a,   --abaqus       ABAQUS input file                -- (1)
-o,   --output       um_convert output file name
-t,   --threads      number of threads
-um,  --mcnpum       mcnpum output file name
```

15.3 The -b Option

This argument (`-b, --binary`) requests that the `MCNPUM` file be created as a binary file instead of ASCII. ASCII is default and results if this option is not specified.

15.4 The -a Option

This argument (`-a, --abaqus`) followed by the file name of the Abaqus .inp file communicates this information to the utility program. This information is required.

15.5 The -o Option

This argument (`-o, --output`) followed by a file name tells the utility program where to write messages and information from the file conversion process. The information that MCNP6 would normally print to its `outp` file when building the unstructured mesh model is written to this file. This argument is optional. If no name is specified, the information is written to the `um_convert.out` file.

15.6 The -t Option

This argument (`-t, --threads`) followed by a number sets the number of OpenMP threads for use in the conversion process. The user should be careful and not oversubscribe threads by requesting too large of a number. (See Section 15.8). This is an optional argument. The default value is 1.

15.7 The -um Option

This argument (`-um, --mcnpum`) followed by a file name tells the utility program what to call the `MCNPUM` file that it generates. If no name is specified, the information is written to the `um_convert.mcnpum` file.

15.8 Program Execution

The *um_convert* utility is a highly parallelized program that can be compiled to use MPI processes, OpenMP threads, and vectorized loops. As a note to those wishing to build the code on their systems from the source, the following is the appropriate command line (using the traditional MCNP6 make system) that will build the code with MPI processes, OpenMP threads, and vectorized loops once the mainline MCNP6 code has been built:

```
make depends build CONFI=''intel openmpi omp'' FC_OPT=''-O3'' GNUJ=4
```

Normal execution of *um_convert* from the command line will result in messages similar to the following appearing in the command window:

```
UM_CONVERT input processing begins.       11- 9-2015 @  9:46:31
  Max threads available:     16
  Global Tracking Model Complete
  Element Neighbors Found
  Part Cell Surfaces Complete
  SKD-Trees Build Complete
  Element Connectivity Complete
  um_convert execution time       19.6 sec
UM_CONVERT input processing ends.         11- 9-2015 @  9:46:50
```

Note that the program provides the user with the maximum number of available threads. The product of the number of MPI processes and the number of threads, specified with the -t Option, should not exceed the number of cpu cores present or performance will be degraded.

A combination of MPI processes and OpenMP threads should produce the shortest execution times on most systems. If the user doesn't have MPI available (e.g., a desktop Windows machine), executing with the maximum number of available threads should still produce acceptable execution times. The utility will process one part / instance at a time, using all of the requested threads as it needs them.

If the user is running on a Linux cluster where MPI has been installed, a combination of MPI processes and OpenMP threads is recommended. As always, performance is contingent on the number of parts / instances in the Abaqus inp file. If there are more cpu cores available than parts / instances, then specifying one MPI process for each part / instance with several threads per process is recommended. If fewer MPI processes are specified than parts / instances, then *um_ convert* will give each process a number of parts / instances to work on in a sequential fashion much like MCNP6 does with its parallel processing of parts / instances. In this later scenario where there are more parts / instances than cpu cores, it may be beneficial to reduce the number of MPI processes so that each process has two threads. This should help when one or a few parts have substantially more elements than the other parts.

Unlike MCNP6 where the master MPI process basically functions as a controller during the calculational phase, all MPI processes in the *um_ convert* utility have a chunk of the parts / instances with which to work.

As a reminder when using MPI processes and OpenMP threads together on certain Linux clusters, the `mpi_paffinity_alone` and `bynode` switches (or their equivalent) may be necessary when using mpirun to ensure that threads are assigned to the correct hardware. Check you MPI install.

15.9 Performance

In order to provide some insight on choosing the number of processes and threads when there are more parts / instances in the model than there are available processes so that there is one part / instance per process, a short study was conducted with a rather large model. The model consists of 400 parts, where there is one instance of each part, or a total of ~1.48 million first-order tetrahedra. The size of the parts ranged from a few hundred elements to ~ 44 thousand elements. This variability in part size does not make an ideal input processing situation from the point of load balancing. Invariably, with the way the parts / instances are assigned to the processes, some processes will finish well before others and remain idle. This scenario is a likely

one for users.

These tests were performed on one of LANL's high-performance Chaos linux clusters during November 2015. Each node has 16 Intel Xeon E5-2670 cpu's operating at 2.6 GHz. Each cpu has ~200 MB of cache and ~2 GB of RAM. During these tests, eight nodes were available for a total of 128 cpus. Because this model was so large and the memory per cpu limited, 128 MPI processes couldn't be requested for the MCNP6 tests and neither 64 nor 128 MPI processes could be requested for the *um_convert* tests. Each test was performed a number of times and averages appear in Tables 15.1 (MCNP6) and 15.2 (*um_convert*).

Table 15.1: MCNP6 Input Processing Performance

# MPI / # Threads	Time (sec)
64 / 2	204
32 / 4	159
16 / 8	133
8 / 16	160

Table 15.2: um_convert Performance

# MPI / # Threads	Time (sec)	# MPI / # Threads	Time (sec)	# MPI / # Threads	Time (sec)
32 / 1	221	16 / 1	305	8 / 1	429
32 / 2	167	16 / 2	204	8 / 2	304
32 / 4	137	16 / 4	153	8 / 4	221
		16 / 8	125	8 / 8	173
				8 / 16	143

Acknowledgements

Over the years, the following personnel have been a part of the MCNP6 unstructured mesh development team, either full-time or part-time, and have contributed in some form to its development.

- Roger L. Martz; development and documentation lead, chief architect of REGL
- David L. Crane; finite element methods, Abaqus interfacing
- Karen C. Kelley; verification and validation, requirements
- Steven S. McCready; Engineering Campaign 7 Program funding lead, verification and validation, requirements
- Lawrence J. Cox; user interface; software quality assurance
- Tim Goorley; ASC Program funding lead, verification and validation
- Casey Anderson; VisIt (visualization) post-processing; verification and validation
- Chelsea D'Angelo; verification and validation
- Jeffrey Bull; technical reviews
- David Dixon; technical reviews
- Joel Kulesza; verification and validation; LyX / LaTeX support
- C. J. Solomon Jr.; verification and validation, requirements, technical review
- Kristofer Zieb; performance testing

MCNP6 unstructured mesh development over the life of this work was funded from the US Department Of Energy's NNSA -- Advanced Strategic Computing (ASC) and Engineering Campaign 7 (C7) programs. We thank these sponsors.

This page intentionally left blank.

Bibliography

[1] X-5 MONTE CARLO TEAM, "MCNP - A General Monte Carlo N-Particle Transport Code, Version 5, Volume I: Overview and Theory," Los Alamos National Laboratory report LA-UR-03-1987 (April 2003).

[2] T. Goorley, et al., "Initial MCNP6 Release Overview," *Nuclear Technology*, 180, 3, 298–315 (2012).

[3] Timothy J. Tautges, Paul P. H. Wilson, Jason A. Kraftcheck, Brandon M. Smith, Douglass L. Henderson, "Acceleration Techniques For Direct Use Of CAD-Based Geometries In Monte Carlo Radiation Transport," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3–7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).

[4] Yican Wu, Qin Zeng, and Lei Lu, "CAD-Based Modeling For 3D Particle Transport," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, Mary 3–7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).

[5] D. Groβe and H. Tsige-Tamirat, "Current Status of the CAD Interface Program McCad for MC Particle Transport Calculations," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3–7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).

[6] Dassault Systèmes Simulia, Inc., "ABAQUS USER MANUALS, Version 6.9," Providence, RI (2009).

[7] CUBIT Geometry and Meshing Toolkit, Version 13.0, February 2011, http://cubit.sandia.gov .

[8] Frank A. Ortega, "General Mesh Viewer User's Manual GMV Version 3.8", Los Alamos National Laboratory report LA-UR-95-2986 (1995).

[9] Roger L. Martz, "The MCNPUM Capability for MCNP6's Unstructured Mesh Feature," Los Alamos National Laboratory report LA-UR-16-23286 (2016).

This page intentionally left blank.