LA-UR-23-21715

Approved for public release; distribution is unlimited.

Title:	Using CUBIT to Create Unstructured Mesh Models for MCNP Simulations
Author(s):	Armstrong, Jerawan Chudoung
Intended for:	Report
Issued:	2024-09-24 (rev.1)

Los Alamos
NATIONAL LABORATORY

Using CUBIT to Create Unstructured Mesh Models for MCNP Simulations

Jerawan Armstrong

1 Introduction

The Monte Carlo N-Particle (MCNP)[1] transport code version 6 (also known as MCNP6) has the capability for tracking particles on unstructured mesh (UM) geometry models embedded into constructive solid geometry (CSG) cells [1]. This feature has been developed for performing calculations of complex geometry models because creating CSG models is a time-consuming and error-prone process as the complexities of geometries increase. An MCNP UM calculation requires UM geometry input files. The UM capability was originally designed to work with UM models created with the Abaqus/CAE software [2] and ASCII input files that it generates. The Abaqus-formatted input files needed for MCNP UM calculations must have the correct Abaqus syntax and meet the additional MCNP requirements. Several software packages can generate a UM model formatted as an Abaqus input file. Cubit, the Sandia National Laboratory automated mesh generation toolkit, can generate a UM model formatted as an Abaqus input file [3]. However, the Abaqus input files created by Cubit cannot be used for MCNP simulations. A Python script has been developed to convert an Abaqus file created by Cubit to an Abaqus file that the MCNP code can process. This report describes the process of using Cubit to create UM models for MCNP calculations.

2 Generating Unstructured Mesh Models

This section assumes that users have basic knowledge of using Cubit to generate 3D mesh models. There are several ways to generate mesh geometries in Cubit, and the process of creating UM models in Cubit for MCNP simulations include

- Use Cubit to create a solid geometry or import a CAD model into Cubit.
- Prepare a model for meshing.
- Check mesh qualities and volumes. Ideally, solid and meshed volumes should be well-matched. It may be impossible to generate a mesh whose volume is well-matched with a solid volume, and hence an expert judgement is needed on how to mesh a geometry.

Creating mesh models for complex geometries is not an easy task and the mesh generation process is iterative. After a mesh model is generated, the following steps are crucial for generating MCNP UM models.

- Create materials where each material name must end with a dash or underscore following by a material number. This material name specification is an MCNP requirement.
- Create blocks and assign materials to the blocks. Cubit blocks are Abaqus parts, and Abaqus parts are instanced to create an Assembly.
- Export a mesh model as an Abaqus input file where each block must be instanced. The MCNP code uses the instances in an Abaqus input file to create a global tracking model. Abaqus instances are used to create MCNP pseudo-cells.

[1] MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory for the U.S. Department of Energy. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ®designation as appropriate. Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademark@lanl.gov. For the purposes of visual clarity, the registered trademark symbol is assumed for all references to MCNP within the remainder of this report.

The commands for creating UM models can be put in a journal file. An example of journal file is shown in Appendix A where this journal file generates an Abaqus input file named *model.inp*. Since the Abaqus input files exported from Cubit do not meet the MCNP requirements, a Python3 script shown in Appendix B can be used to convert an Abaqus file created by Cubit to an Abaqus file that the MCNP code can process. For example, running the following Python command will produce a new Aabqus input file named *model_ready.inp*:

```
python cubit_to_mcnp.py model.inp
```

The following Python script has the same function of using the above Python command line.

```
from  cubit_to_mcnp import cubit_to_mcnp
abqcubit = 'model.inp'       # Abaqus input file created by Cubit
abqmcnp  = 'model_ready.inp' # Abaqus input file to be created for MCNP calculations
cubit_to_mcnp(abqcubit,abqmcnp)
```

References

[1] Joel A. Kulesza et al. *MCNP® Code Version 6.3.0 Theory & User Manual*. Tech. rep. LA-UR-22-30006, Rev. 1. Los Alamos, NM, USA: Los Alamos National Laboratory, Sept. 2022.

[2] Dassault Systemes. *Abaqus/CAE - finite element analysis software*. https://www.3ds.com/products-services/simulia/products/abaqus/abaquscae/.

[3] Cubit Code Developers. *CUBIT 16.02 User Document*. Tech. rep. SAND2021-12663 W. Albuquerque, NM, USA: Sandia National Laboratories, New Mexico, 2021.

Appendix A An Example of Cubit Journal File

Cubit commands can be written into an ASCII file called a journal file. An example of Cubit journal file for creating an Abaqus input file for MCNP simulations is shown in this Appendix. This journal file is for for creating a mesh model and exporting this model into an Abaqus input file named *model.inp*. Note that MCNP cannot process this Abaqus input file and the Python script shown in Appendix B can be used to convert an Abaqus input created by Cubit to an Abaqus input that the MCNP code can process.

```
# -----------------
# Create Geometries
# -----------------
reset
view reset
from  100 50 100

create sphere radius 10
volume 1 name "sph_1"

create cylinder height 20 radius 5.55
volume 2 name "cyl_1"
rotate cyl_1 about y angle 90
move cyl_1 x 15

create sphere radius 19.95
volume 3 name "sph_2"

# intersect volume 2 3 - Updated volume(s): 3; Destroyed volume(s): 2; Created volume 3
intersect volume 2 3

# unite volume 1 3 - Updated volume(s): 1; Destroyed volume(s): 3; Created volume 1
unite volume 1 3
volume 1 rename "part_1"

create sphere radius 10.2
volume 4 name "sph_3"

create cylinder height 20 radius 5.75
volume 5 name "cyl_2"
rotate volume 5 angle 90 about y
move volume 5 x 15

create sphere radius 19.95
volume 6 name "sph_4"

# intersect volume 5 6 - Updated volume(s): 6; Destroyed volume(s): 5; Created volume 6
intersect volume 5 6
#
# unite volume 4 6 - Updated volume(s): 4; Destroyed volume(s): 6; Created volume 4
unite volume 4 6

# subtract volume 1 from 4 keep: create volume 7 and keep volume 4
subtract volume 1 from 4 keep
volume 7 rename "part_2"
delete volume 4

Create sphere radius 19.75
volume 8 name "sph_5"

# subtract volume 1 7 from 8 keep: create volume 9 and keep volume 1 7 8
subtract volume 1 7 from 8 keep
```

```
volume 9 rename "part_3"
delete volume 8

create sphere radius 19.95
volume 10 name "sph_6"

# subtract volume 1 7 9 from 10 keep: create volume 11 and keep volume 1 7 9 10
subtract volume 1 7 9 from 10 keep
volume 11 rename "part_4"
delete volume 10

#----------------
# Mesh Geometries
#----------------
part_1 Scheme Tetmesh
mesh part_1

part_2 Scheme Tetmesh
part_2 size auto factor 4
mesh part_2

part_3 Scheme Tetmesh
part_3 size auto factor 3
mesh part_3

part_4 Scheme Tetmesh
part_4 size auto factor 4
mesh part_4

#-----------------
# Create Materials
#-----------------
create material "Air-100" property_group "CUBIT-ABAQUS"
modify material "Air-100" scalar_properties "DENSITY" 0.001288

create material "Steel-200" property_group "CUBIT-ABAQUS"
modify material "Steel-200" scalar_properties "DENSITY" 7.8240

create material "Powdered_Aluminum-300" property_group "CUBIT-ABAQUS"
modify material "Powdered_Aluminum-300" scalar_properties "DENSITY" 1.223

#--------------
# Create Blocks
#--------------
block 1 add part_1
block 2 add part_2
block 3 add part_3
block 4 add part_4

block 1 add material "Air-100"
block 2 add material "Steel-200"
block 3 add material "Powdered_Aluminum-300"
block 4 add material "Steel-200"

block 1 name "part_1_Air"
block 2 name "part_2_Steel"
block 3 name "part_3_Al"
block 4 name "part_4_Steel"

#----------------------------
# Export an Abaqus Mesh Model
#----------------------------
export abaqus "model.inp"\
instance block 1 source_csys 0 target_csys 0\
instance block 2 source_csys 0 target_csys 0\
instance block 3 source_csys 0 target_csys 0\
instance block 4 source_csys 0 target_csys 0\
dimension 3  overwrite  everything
```

Click  to download this Cubit journal file.

Appendix B Python Script

A Python3 script called cubit_to_mcnp.py has been developed for converting an Abaqus input file exported from Cubit to an Abaqus input file that the MCNP code can process. This Python script is shown in this Appendix.

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file

import os

def cubit_to_mcnp(cubitabq, mcnpabq):
    """
    Process an Abaqus input file created by Cubit to write an Abaqus input file that MCNP can process.

    Parameters
    ----------
    cubitabq : STR
        An Abaqus input filename created from CUBIT
    mcnpabq : STR
        An Abaqus input to be created so that MCNP can process this file

    Raises
    ------
    OSError
        An Abaqus input file (cubitabq) does not exists.

    Returns
    -------
    text : List of string
        List of lines written into an Abaqus input file (mcnpabq)
    """
    if not os.path.isfile(cubitabq):
        raise OSError(f'file does not exist: {cubitabq}')

    print(f'\nRead and process an Abaqus input file: {cubitabq}')
    with open(cubitabq, "r", encoding='utf-8',errors='ignore') as f:
        text, total_nodes, total_elements = _read_buffer(f)

    print(f"    total_nodes = {total_nodes}")
```

```
    print(f"    total_elements = {total_elements}")
    print(f'Write an Abaqus input file ready for MCNP: {mcnpabq}\n')
    g = open(mcnpabq, "w")
    g.write(''.join(text))
    g.close()

    return text

#-----------------------------------------------------------------------
def _read_buffer(f):
    """
    Read and process an Abaqus input file exported from Cubit.
    """
    total_nodes = 0
    total_elements = 0

    text = ["**\n**\n** PARTS\n**\n**\n"]

    # read and process part blocks
    step   = 0
    cur_part=""
    while True:
        line = f.readline()
        if not line: raise ValueError('end of file while reading part data')

        if step == 0:
            if line.lower().startswith('*part,'):
                print(f"    {line.strip()}")
                text.append(line)
                cur_part = line
                step = 1

            elif line.lower().startswith('*assembly,'):
                text.append("**\n** ASSEMBLY\n**\n**\n")
                text.append(line)
                text.append("**\n")
                break

        elif step == 1:
            if line.lower().startswith('*node,'):
                print(f"        {line.strip()}")
                text.append("*NODE\n")
                node_numbers = {}
                node_id = 0
                while True:
                    line = f.readline()
                    if line.startswith("*"): break
                    node_id += 1
                    i = line.find(",")
                    n = line[:i]
                    new_line = line.replace(n,str(node_id))
                    node_numbers[n] = str(node_id)
                    text.append(new_line)

                if line.lower().startswith("*element,"):
                    raise ValueError("there is no comment line before *ELEMENT keyword line: Cubit writes a different file format")
                if node_id == 0:
                    raise ValueError(f"no node data line in part {cur_part}")
                step = 2

        elif step== 2:
            if line.lower().startswith("*element,"):
                print(f"        {line.strip()}")
                words = line.split()
                if words[1].lower().startswith("type=s4r"):
                    text.append("*ELEMENT, TYPE=C3D4\n")
                elif words[1].lower().startswith("type=c3d4"):
                    text.append("*ELEMENT, TYPE=C3D4\n")
                elif words[1].lower().startswith("type=c3d8"):
                    text.append("*ELEMENT, TYPE=C3D8\n")
                else:
                    raise ValueError(f"this code implementation is not for element type: {words[1]}")

                el_id = 0
                while True:
                    line = f.readline()
                    if line.startswith("*"): break
                    words = line.strip().split(",")
                    el_id += 1
                    tmp = [str(el_id)+", "]
                    for w in words[1:]:
                        m = node_numbers.get(w)
                        tmp.append(m+", ")
                    new_line = "".join(tmp).strip()+"\n"
                    text.append(new_line)

                if el_id == 0:
                    raise ValueError(f"no element data line for {cur_part}")
                step = 3

        elif step == 3:
            if line.lower().startswith("*element"):
                raise ValueError("this code implementation is only for one *ELEMENT keyword line in a part")

            if line.lower().startswith("*solid section,") or line.lower().startswith("*shell section,"):
                print(f"        {line.strip()}")
                print(f"        number of nodes: {node_id}")
                print(f"        number of elements: {el_id}")
                total_nodes += node_id
                total_elements += el_id
                words = line.split(" ")
                mat   = ""
                for w in words:
                    if w.lower().startswith("material="):
                        i = w.rfind("-")
                        if i >= 0:
                            mat = w[i+1:]
                        else:
                            i = w.rfind("_")
                            if i >= 0: mat = w[i+1:]
                        break

                if len(mat)==0:
                    raise ValueError(f"no material in section line {line}")

                text.append(f"*ELSET, ELSET=statistic_material_{mat.strip()}, generate\n")
                text.append(f"1, {el_id}, 1\n*END PART\n**\n")
                step = 0
```

```
    # Read the lines after the *ASSEMBLY line
    while True:
        line = f.readline()
        if not line: raise ValueError("end of file while reading assembly data")

        if line.lower().startswith("*instance,"):
            text.append(line)
            while True:
                line = f.readline()
                if not line: raise ValueError("end of file while processing instance data")
                text.append(line)
                if line.lower().startswith("*end instance") :
                    text.append("**\n")
                    break

        elif line.lower().startswith("*end assembly"):
            text.append(line)
            break

    # Read the material lines
    text.append("**\n**\n** MATERIALS\n**\n**\n")
    while True:
        line = f.readline()
        if not line: break
        if line.lower().startswith("*material,"):
            text.append(line)
        elif line.lower().startswith("*density"):
            text.append(line)
            line = f.readline()
            if not line: raise ValueError("end of file while reading a density value line")
            text.append(line)

    return text, total_nodes, total_elements

#-------------------------------------------------------------------------
if __name__=='__main__':
    import argparse

    text ='** Extract data from an Abaqus input file exported from CUBIT and write an Abaqus input file that MCNP can process **'
    parser = argparse.ArgumentParser(description=text)

    parser.add_argument("abaqus_filename",
                        type=str,
                        help="Abaqus filename exported from CUBIT")

    parser.add_argument("-o", "--output", metavar="<mcnpabq.inp>",
                        type=str,
                        help="Abaqus filename to be created")

    args = parser.parse_args()

    inp  = args.abaqus_filename
    if args.output is None:
        basename = os.path.basename(inp)
        name     = basename.split('.')[0]
        out      = name+'_ready.inp'
    else:
        out      = args.output
    text = cubit_to_mcnp(inp,out)
```

Click  to download this Python file.