LA-UR-24-28679

Approved for public release; distribution is unlimited.

Title:	A Python Toolkit to Read and Process LNK3DNT Files
Author(s):	Kitamura, Anthony Rei Armstrong, Jerawan Chudoung MacQuigg, Michael Robert
Intended for:	2024 MCNP User Symposium, 2024-08-19/2024-08-22 (Los Alamos, New Mexico, United States)
Issued:	2024-08-12

A Python Toolkit to Read and Process LNK3DNT Files

Anthony Kitamura
Mentors: Jerawan Armstrong, Robbie MacQuigg

2024 MCNP User Symposium
August 19-22, 2024

Outline

- LNK3DNT File Processing
  - LNK3DNT file format
  - Processing of LNK3DNT file
  - Text outputs
- EXODUS-II File Conversion
  - EXODUS-II file usage
  - EXODUS-II history
  - EXODUS-II file format
  - LNK3DNT to EXODUS-II mesh conversion
  - EXODUS-II test cases
- Conclusions and Future Works

LNK3DNT File

- LNK3DNT is a binary, structured mesh file format (primarily used in PARTISN and MCNP) [1, 2].
  - Need the EMBED card to use in MCNP
- An element in a LNK3DNT file contains material identifiers & densities (atom [at/b-cm] or mass [g/cc]).
  - 1 is for void in LNK3DNT.
  - 0 is for void in MCNP.
  - A density type must be specified in MCNP via *dentype* keyword in EMBED card for v. 6.3.1 and higher.

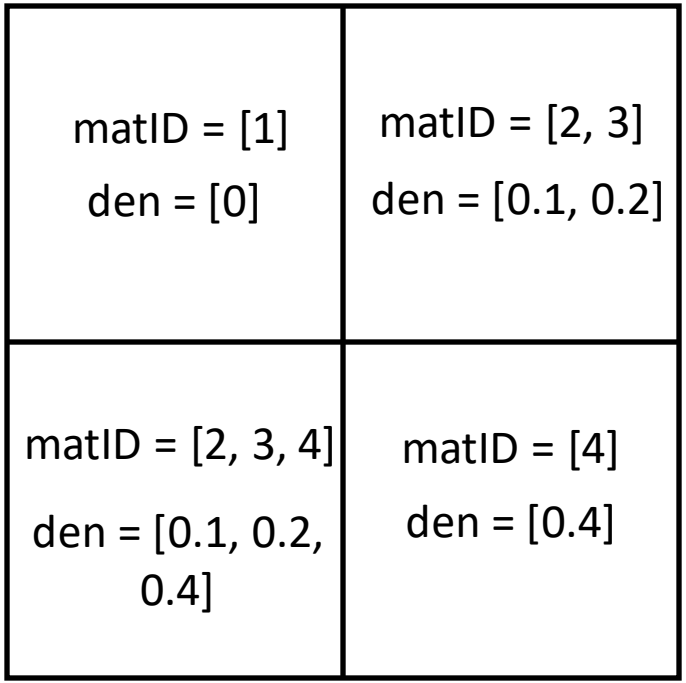


Figure 1. Simple Description of a 2D LNK3DNT mesh with mixed materials (i.e., an element can contain more than one material).

LNK3DNT Header Data

- Header data contains important file information:
  - igeom: geometry type
  - nzone: number of zones (i.e., number of materials in a file)
  - ninti: number of elements along X
  - nintj: number of elements along Y
  - nintk: number of elements along Z
  - nmxsp: maximum number of materials in an element
- The header data is used to define the LNK3DNT data structures, i.e., the material IDs, densities, and mesh.
  - Total number of elements in mesh = ninti x nintj x nintk

```
offset += rb
rb = 4*27
dt = np.dtype('i4,' * 27)
data = np.fromfile(infile, dt, offset=offset)[0]
igeom = data[0]      # geometry  type
nzone = data[1]      # number of zones
ncinti = data[4]     # number of 1st dim. coarse mesh
ncintj = data[5]     # number of 2nd dim. coarse mesh
ncintk = data[6]     # number of 3rd dim. coarse mesh
ninti = data[7]      # number of 1st dim. mesh
nintj = data[8]      # number of 2nd dim. mesh
nintk = data[9]      # number of 3rd dim. mesh
nmxsp = data[23]     # number of mixing materials
```

LNK3DNT Data Structure

- The array structure of the material ID(s) and density(s) can be visually represented as the figures below, where the dimension is the number of elements times nmxsp.

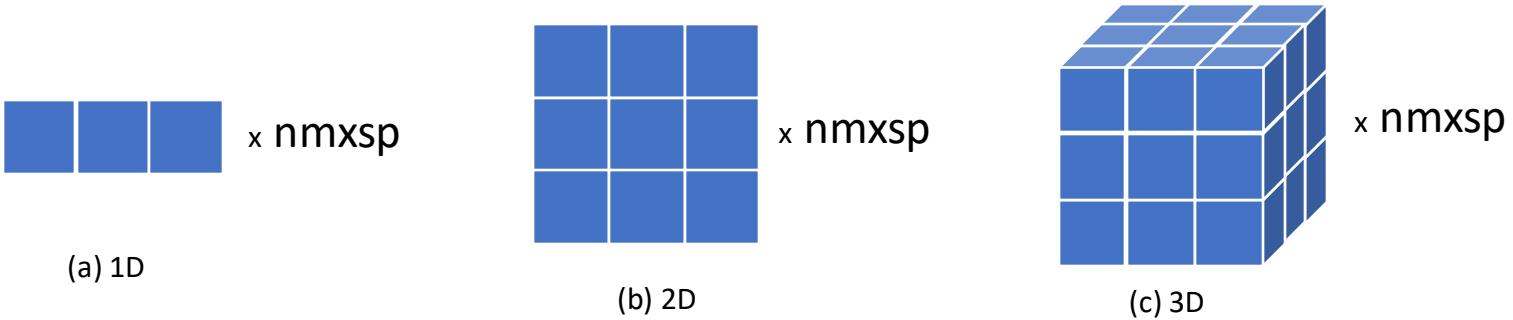


Figure 2. Visual representation of the data structure in LNK3DNT file.

Python Tool for Processing LNK3DNT Files

- Developed a Python code to process LNK3DNT files.
  - Used numpy.fromfile to read data in a LNK3DNT file.
  - Used Multiproccessing.Pool to parallelize material sorting.
  - Used numpy arrays to vectorize the handling of data.
  - The code writes an ASCII file containing information about the LNK3DNT file, such as material IDs and densities, and an MCNP skeleton input file.

```
n = ninti*nintj*nintk*nmxsp
offset += rb
rb = 4*n
dt = np.dtype('i4,'*n)
self.mat = np.array(list(np.fromfile(infile, dt, offset=offset)[0]))
self.umat = np.unique(self.mat)

offset += rb
dt = np.dtype('f8,'*n)
self.rho = np.array(list(np.fromfile(infile, dt, offset=offset)[0]))
```

```
if ncpus <= 1:
    mdata = l3d.process_all_materials()
else:
    ncpus = min(ncpus,os.cpu_count())
    nmats = len(umat)
    if nmats <= ncpus: csize = 1; ncpus=nmats
    else: csize = int(nmats/ncpus)
    with Pool(ncpus) as pool:
        mdata = pool.map(l3d.process_material,umat,chunksize=csize)
```

Output File: LNK3DNT Information

```
Header data in cone.l3d:
  3    name: lnk3dnt
  4    user: developm
  5    version: 6.4.0
  6    ivers: 4
  7    igeom: 14
  8    geom: XYZ_MESH
  9    nzone: 4
 10    ncinti: 60
 11    ncintj: 60
 12    ncintk: 60
 13    ninti: 60
 14    nintj: 60
 15    nintk: 60
 16    nmxsp: 2
 17 total number of elements (model): 216000
 18
 19 mat_number  num_of_elements        min_rho             max_rho
 20        1       207520          0.00000E+00         0.00000E+00
 21        2         1708          1.44000E-01         1.80000E+01
 22        3        60008          2.16000E-02         2.70000E+00
 23        4       162065          8.00000E-06         1.00000E-03
 24        5          699          6.24000E-02         7.80000E+00
 25
```

```
26 mat_number      xmin(cm)    xmax(cm)    ymin(cm)    ymax(cm)    zmin(cm)    zmax(cm)
27        1      -8.000E-01  6.000E+00 -2.000E-01  6.000E+00 -2.600E+00  6.000E+00
28        2      -2.000E-01  2.000E-01 -4.000E-01  6.000E-01 -3.600E+00  3.600E+00
29        3      -1.000E+00  2.000E-01 -2.600E+00  4.000E-01 -5.000E+00  3.600E+00
30        4      -6.000E+00  5.400E+00 -6.000E+00  5.600E+00 -6.000E+00  6.000E+00
31        5       4.600E+00  5.400E+00  4.400E+00  5.600E+00  4.000E+00  6.000E+00
32
33 total number of elements (materials): 432000
34
35 number of void elements: 334
36 number of mixed elements: 215666
37 number of non-mixed elements: 0
38
39 xmesh range (cm): -6.0 : 6.0
40 ymesh range (cm): -6.0 : 6.0
41 zmesh range (cm): -6.0 : 6.0
```

Output File: MCNP Skeleton Input Cards

```
Problem Title
c ********************************************************
c *                  Cell Block                          *
c ********************************************************
c
c ========== mesh cells  ======================
 101    0     0    u=1 imp:   $ void cell
 102    2    1.0    0    u=1 imp:
 103    3    1.0    0    u=1 imp:
 104    4    1.0    0    u=1 imp:
 105    5    1.0    0    u=1 imp:
 106    0     0    u=1 imp:   $ background cell
c
c ================ CSG cells ====================
c Modify a fill cell where surface(s) must be used to define a fill cell.
c A fill cell must not crop the mesh boundaries.
c
 107    0      fill=1  imp:   $ fill cell
c
c Add other CSG cell(s) to complete the cell block.
c

c ********************************************************
c *                  Surface Block                       *
c ********************************************************
c Add surface card(s)
c
c A fill cell must not crop the mesh boundaries.
c
c   Mesh Boundaries:
c       (imin, imax)  = (-6.0, 6.0)
c       (jmin, jmax)  = (-6.0, 6.0)
c       (kmin, kmax)  = (-6.0, 6.0)
c
c   PARTISN GEOM = 14 (x-y-z)
c
c Mesh boundaries and GEOM should be used to define surfaces
c used to make a fill cell the cell block.
c

c ********************************************************
c *                  Data Block                          *
c ********************************************************
c The dentype keyword in embed card is only valid for MCNP6.3.1 or later
c The dentype option must be removed if using MCNP6.0-MCNP6.3.
c Note: The dentype in older versions could only be mass.
c If the densities in LNK3DNT are mass densities (g/cc), then dentype=mass.
c If the densities in LNK3DNT are atom densities (at/barn-cm), then
dentype=atom.
c Densities in LNK3DNT must be mass or atom densities.
c Note: The matcell keyword in the embed card maps the LNK3DNT (matID - 1)
to
c the MCNP cell numbers.
c
 EMBED1 meshgeo=lnk3dnt
        mgeoin=cone.l3d  $ LNK3DNT filename (must be lowercase)
        dentype=mass $ type of densities in LNK3DNT; must be mass or atom
        background=106
        matcell=
        0  101
        1  102
        2  103
        3  104
        4  105
```

MCNP Skeleton Input: Case Study

- What if the material numbers in the LNK3DNT are not contiguous?
  - *MATCELL* maps the LNK3DNT material IDs (minus 1) to the MCNP pseudo cells.
  - Materials in the pseudo cells are mapped to the material cards following normal convention.

```
 58 c ======== embed card ====================
 59 EMBED1 meshgeo=lnk3dnt
 60         mgeoin=godiva_mixmat_m.l3d  $ LNK3DNT
filename (must be lowercase)
 61         dentype=mass $ type of densities in LNK3DNT;
must be mass or atom
 62         background=104
 63         matcell=
 64         1  101
 65         4  102
 66         5  103
```

```
6 c ========== mesh cells  ======================
7  101    11    1.0    0    u=1 imp:n=1
8  102    14    1.0    0    u=1 imp:n=1
9  103    15    1.0    0    u=1 imp:n=1


46 c ======== material cards ==============
47 m11 92235.00c 1.0
48 m14 92238.00c 1.0
49 m15 92234.00c 1.0
```

LNK3DNT Mesh Conversion to EXODUS-II

Why convert LNK3DNT to EXODUS-II?

- LNK3DNT files cannot be visualized in ParaView [3].
  - EXODUS-II files can be visualized in ParaView.
- LNK3DNT files have limited usability.
  - EXODUS-II files are used across many DOE codes.
    - Multiphysics Object Orientated Simulation Environment (MOOSE, INL) [4]
    - Sierra Mechanics (SNL) [5]
- CUBIT (SNL) and MFEM (LLNL) meshing software also support EXODUS-II file format (via netCDF formatting) [6, 7].

EXODUS-II File

- EXODUS-I file format was created in the 1980's by Sandia National Laboratories (SNL) [8].
  - EXODUS-II was designed in the 1990's to overcome some deficiencies in the older format (i.e., allow for larger file sizes, improve API, and be independent of the machine used to generate the file) [9].
- EXODUS-II files are written using the netCDF format.

EXODUS-II Mesh Conversion

- Developed a Python code to convert a LNK3DNT file to an EXODUS-II file.
- Used *exodusii*, a standalone Python-wrapped library based off Sandia Engineering Analysis Code Access System (SEACAS) [10].
  - Only requires *netCDF4* Python library to be installed instead of whole suite.
- Mesh conversion handled by “l3d2exo.py” script.
  - If the user wishes to use ”l3d2exo.py”, must install *exodusii* and *netCDF4* Python libraries for use in their Python environment.
  - Currently supports BAR2 (1D), QUAD4 (2D), and HEX8 (3D) geometry types

Code Snip of l3d2exo.py Generation of Element Blocks

- Code snippet of ”set_elements” function.
- Handles the generation of the element blocks, their variables and values.
  - Material IDs
  - Densities

```
# generate the global element connectivity array
elem_conn = self.make_elem_conn()
# map the matIDs to element IDs
matloc = self.sort_mat_prop()
# Define the number of variables
self.no_var = len(self.umat)*2
# Generate the variable names: matIDs and densities
var_names = self.make_var_names()
# Initialize the variable truth table
self.tru_tab = np.zeros((len(matloc), self.no_var), dtype=int)
# Initialize block id, number of elem in block, and variable vals
self.block_id, self.elem_per_blk = 0, []
var_values = []
# Iterate over element blocks
for m in matloc:
    self.elem_per_blk.append(len(matloc[m]))
    self.exo.put_element_block(
        self.block_id, self.geo, self.elem_per_blk[self.block_id],
        self.nodes_per_elem, self.side_per_elem)
    blk_conn = np.take(elem_conn, matloc[m], axis=0)
    self.exo.put_element_conn(self.block_id, blk_conn)
    self.tru_tab[self.block_id,:] = self.make_truth_table(m)
    var_values.append(self.get_var_vals(m, matloc, var_names))
    self.block_id += 1

self.exo.put_element_variable_params(self.no_var)
self.exo.put_element_variable_names(var_names)
self.exo.put_element_variable_truth_table(self.tru_tab)
self.set_var_vals(var_values, var_names)
```

EXODUS-II Element Nodal Format

- An element is defined by the number of nodes.
  - HEX8, QUAD4, and BAR2 elements are defined by 8, 4, and 2 nodes, respectively.
- Elements are grouped into element blocks in an EXODUS-II file.
  - Elements must be of the same “type” (i.e., have the same geometry, number of nodes defining them, and the same variables assigned to them).

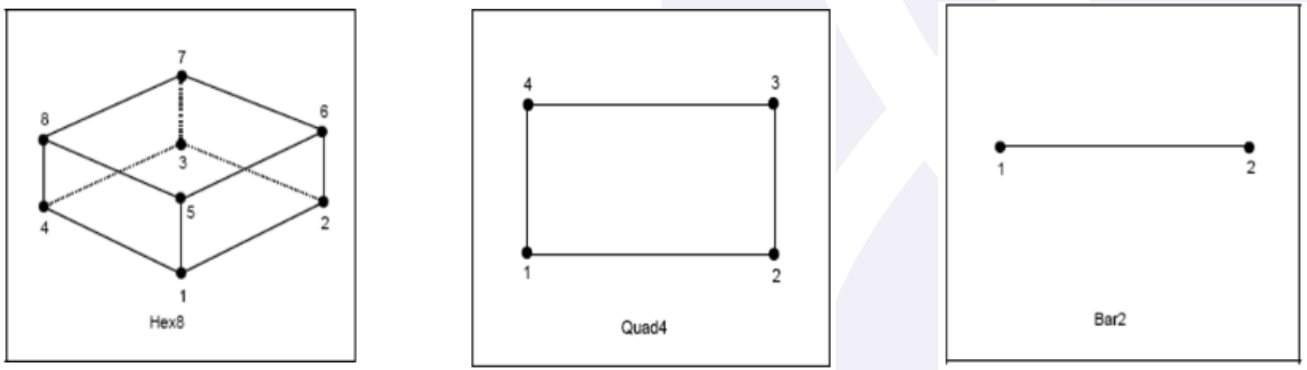


Figure 3. Node ordering for HEX8 (3D), QUAD4 (2D), and BAR2 (1D) [9].

EXODUS-II Element Nodal Format (Cont.)

- EXODUS-II files are written in C++.
  - row-major order (X-axis read first).
- This requires a ‘sorting' algorithm to define the node IDs that define an element.
  - **I, J, K** = Total no. of nodes along X, Y, Z.
  - i, j, k = instance of node along X, Y, Z.

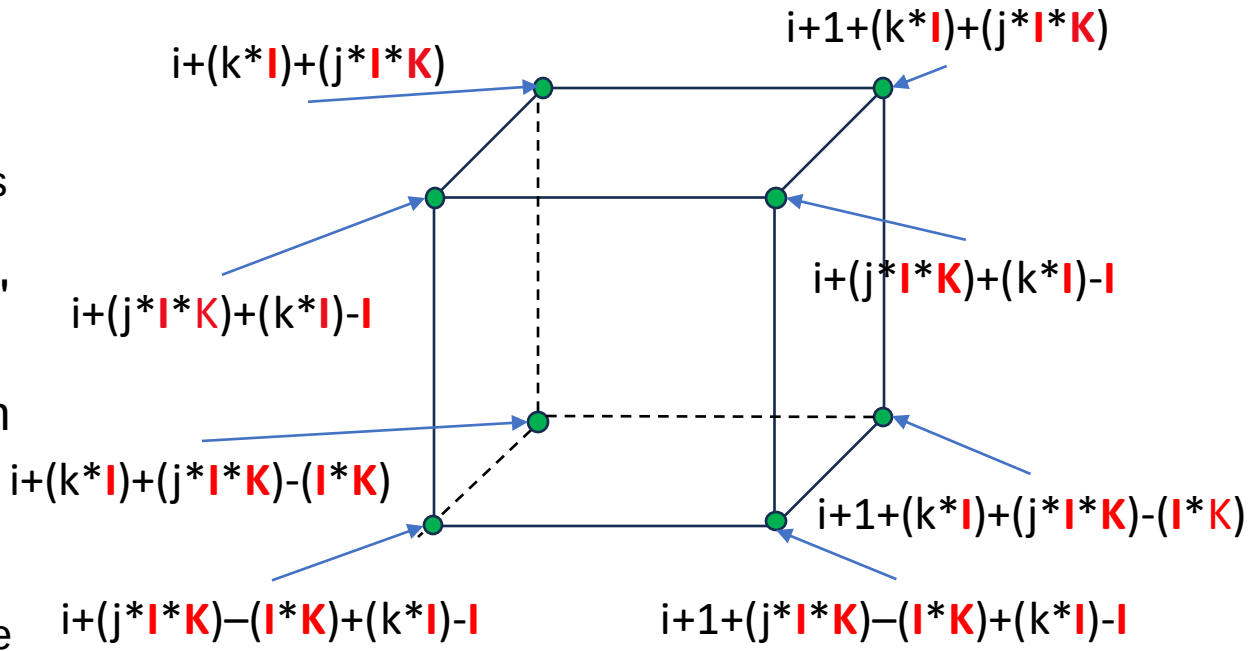


Figure 4. Node ordering scheme for HEX8 element.

EXODUS-II Material Properties

- EXODUS-II can assign element blocks specific variables:
  - Material Identifiers
  - Densities
- Variable truth tables are used to map variables to element blocks.
  - Used to specify which blocks contain what variables.
- Developed an algorithm to map material properties from LNK3DNT to EXODUS-II with element blocks and variable truth tables.

Code Snip of l3d2exo.py Material Sorting

- The sorting of the material properties is handled by four functions in the following order:
  - sort_mat_prop
  - make_truth_table
  - get_var_vals
  - set_var_vals
- Unique materials/material pairs are sorted into element blocks, and then the corresponding material properties are assigned to each block with variable truth tables.

```
def sort_mat_prop(self):
    """
    Sort along each element for unique materials/material pairs
    """
    loc = {}
    for u in self.matIDs:
        iloc = np.where(np.all(u==self.mat,axis=1 ))[0]
        loc[str(u)] = iloc
    return loc
```

```
def make_truth_table(self, m):
    """
    Create the variable truth table for each element block
    """
    tru_tab = np.zeros(len(self.umat), dtype=int)
    i = 0
    for u in self.umat:
        if str(u) in m:
            tru_tab[i] = 1
        i+=1
    tru_tab = np.tile(tru_tab, 2)
    return tru_tab
```

```
def get_var_vals(self, m, matloc, var_names):
    """
    Get variables for each element block
    """
    for var in range(self.no_var):
        if self.tru_tab[self.block_id, var] == 1:
            mat = np.take(self.mat, matloc[m], axis=0).T
            den = np.take(self.den, matloc[m], axis=0).T
            # Below is to catch pure/void regions and prevent duplication
            mat = np.unique(mat, axis=0)
            den = np.unique(den, axis=0)
            values = np.vstack((mat, den))
    return values
```

```
def set_var_vals(self, values, var_names):
    """
    Set the element variables for each block
    """
    for blk in range(self.block_id):
        idx = 0
        for var in range(self.no_var):
            if self.tru_tab[blk, var] == 1:
                print("Written into block: ", blk )
                self.exo.put_element_variable_values(
                    1, blk, var_names[var], values[blk][idx])
                idx+=1
```

Test Problem: cyl01.l3d (RZ)

- EXODUS-II mesh of modified RZ geometry taken from MCNP6 test suite.
  - LNK3DNT generated by MCNP6
  - Refined the mesh to be 100 x 200 (XY).
  - Split the cylinder in half; top is U238, bottom is U235.
  - U238 Density: 19 g/cc
  - U235 Density: 18.7 g/cc
- mcnp6/Testing/features/dawwg/cyl01

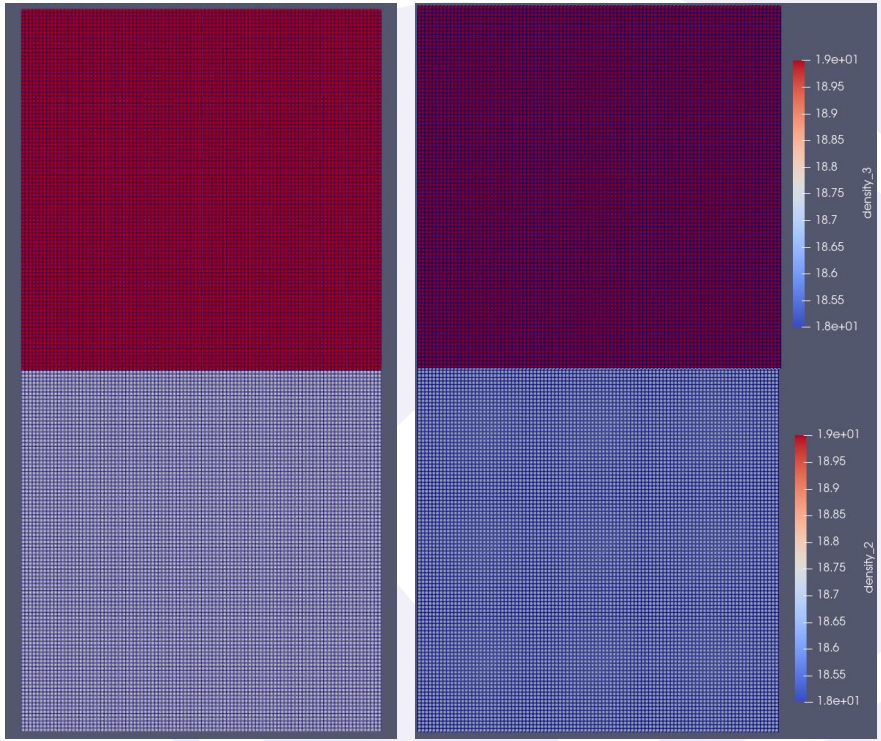


Figure 5. EXODUS-II mesh of modified *cyl01.inp*. Top half is U238, bottom half is U235.

Test Problem: cone.l3d (XYZ)

- EXODUS-II mesh of XYZ LNK3DNT mesh taken from MCNP6 Test suite.
  - Note: LNK3DNT was not made by MCNP; MCNP cannot make mixed material mesh.
- mcnp6/Testing/features/dawwg/cone

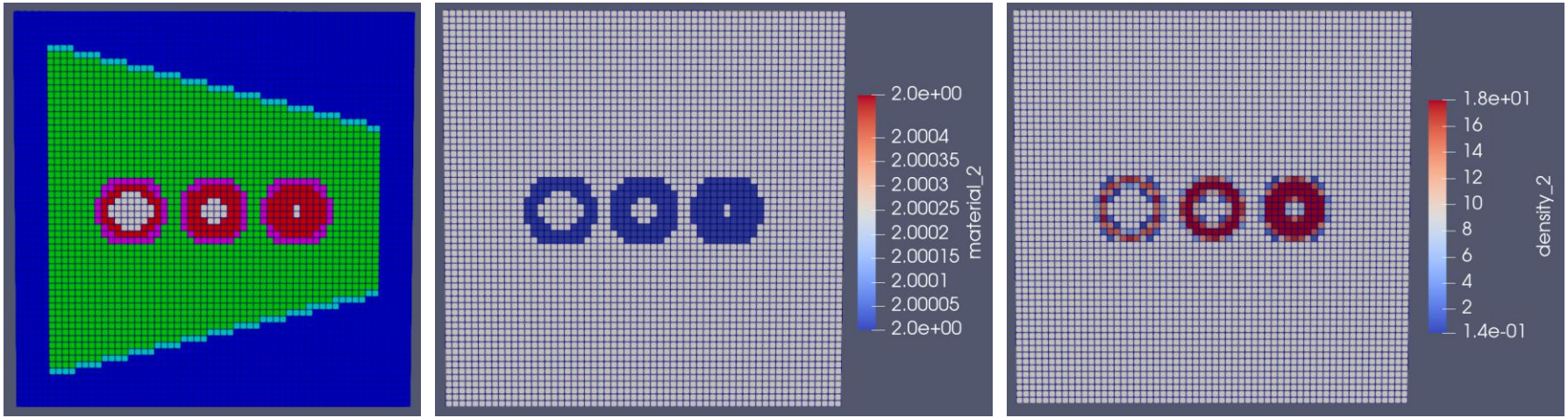
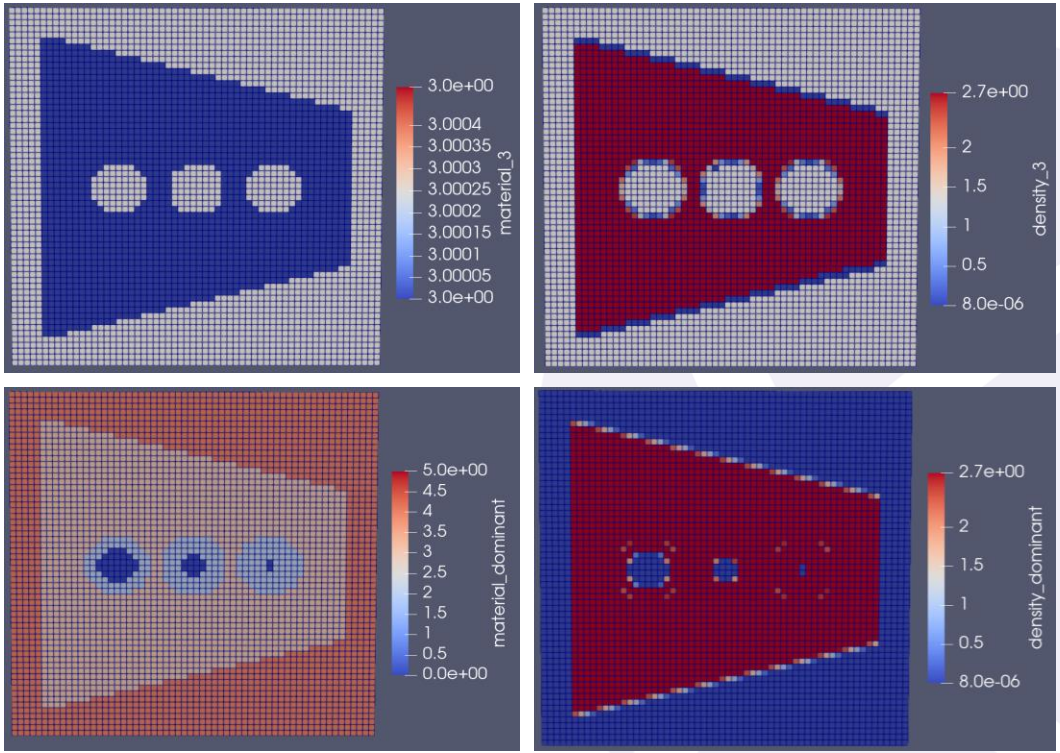


Figure 6. Resulting EXODUS-II mesh from the *cone.inp* (XYZ) test in MCNP6. Material shown on the right are the water spheres.

Test Problem: cone.l3d (Cont.)



l3d2exo.py

MCNPTools

Figure 7. Comparison of Al (material 3) from the EXODUS-II mesh and the VTK model (MCNPTools) [11]. The MCNPTool's generated VTK model cannot show multiple materials in the same cell, but rather displays the most dominant one.

Conclusions and Future Works

- Created a Python toolkit to read and process LNK3DNT files and generate a MCNP skeleton input file and information about the LNK3DNT.
- Developed a Python code to convert LNK3DNT files to EXODUS-II files, allowing the geometry to be visualized.
- Future Works:
  - Import the EXODUS-II mesh to other codes.
    - Griffin, a multiphysics code for nuclear reactors in the MOOSE framework.
  - Refactor code for improved performance.

Questions?

kitamuraa@tamu.edu

References

1. Ray E. Alcouffe, Randal S. Baker, Jon A. Dahl, Erin J. Davis Thomas G. Sallera, Scott A. Turner, Robert C. Ward, and Robert J. Zerr. Partisn: A time-dependent, parallel neutral particle transport code system. Technical Report LA-UR-08-07258, Los Alamos National Laboratory, Los Alamos, NM, USA, September 2017. https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-08-07258
2. Joel A. Kulesza, Terry R. Adams, Jerawan C. Armstrong, Simon R. Bolding, Forrest B. Brown, Jeffrey S. Bull, Timothy P. Burke, Alexander R. Clark, Robert Arthur Forster III, Jesse F. Giron, Avery S. Grieve, Colin J. Josey, Roger L. Martz, Gregg W. McKinney, Eric J. Pearson, Michael E. Rising, Clell J. Solomon Jr., Sriram Swaminarayan, Travis J. Trahan, Stephen C. Wilson, and Anthony J. Zukaitis. MCNP® Code Version 6.3.0 Theory & User Manual. Technical Report LA-UR-22-30006, Rev. 1, Los Alamos National Laboratory, Los Alamos, NM, USA, September 2022. https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-22-30006
3. ParaView, "ParaView: An Open-Source, Multi-Platform Data Analysis and Visualization Application," https://www.paraview.org.
4. Guillaume Giudicelli, Alexander Lindsay, Logan Harbour, Casey Icenhour, Mengnan Li, Joshua E. Hansel, Peter German, Patrick Behne, Oana Marin, Roy H. Stogner, Jason M. Miller, Daniel Schwen, Yaqi Wang, Lynn Munday, Sebastian Schunert, Benjamin W. Spencer, Dewen Yushu, Antonio Recuero, Zachary M. Prince, Max Nezdyur, Tianchen Hu, Yinbin Miao, Yeon Sang Jung, Christopher Matthews, April Novak, Brandon Langley, Timothy Truster, Nuno Nobre, Brian Alger, David Andrˇs, Fande Kong, Robert Carlsen, Andrew E. Slaughter, John W. Peterson, Derek Gaston, and Cody Permann. 3.0 - MOOSE: Enabling massively parallel multiphysics simulations. SoftwareX, 26:101690, 2024. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2024.101690. https://www.sciencedirect.com/science/article/pii/S235271102400061X
5. Frank N. Beckwith, Guy L. Bergel, Gabriel J. de Frias, Kevin L. Manktelow, Mark T. Merewether, Scott T. Miller, Krishen J. Parmar, Timothy R. Shelton, Jesse D. Thomas, Jeremy Trageser, Benjamin C. Treweek, Michael G. Veilleux, and Ellen B. Wagman. “Sierra/solidmechanics 5.6 user’s manual”. doi: 10.2172/1861945. https://www.osti.gov/biblio/1861945

References (Cont.)

6. Cubit Code Developers. “The Cubit® Geometry and Mesh Generation Toolkit”. Sandia National Laboratories, New Mexico, Albuquerque, NM, USA, 2021. https://cubit.sandia.gov/
7. R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, Tz. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini. “MFEM: A modular finite element methods library”. Computers & Mathematics with Applications, 81:42–74, 2021. doi: 10.1016/j.camwa.2020.06.009
8. W C Mills-Curran, A P Gilkey, and D P Flanagan. Exodus: A finite element file format for pre and postprocessing. Technical report, Sandia National Laboratories, Sep 1988. https://www.osti.gov/servlets/purl/6902151.
9. Gregory D. Sjaardema, Larry A. Schoof, and Victor R. Yarberry. EXODUS: A Finite Element Data Model. Sandia National Laboratories, 2023. https://sandialabs.github.io/seacas-docs/exodusII-new.pdf
10. Seacas. “SEACAS - SEACAS 2023/07/12 documentation”. https://sandialabs.github.io/seacas-docs/sphinx/html/index.html#
11. Cameron Russell Bates, Simon R. Bolding, Colin James Josey, Joel A. Kulesza, Clell Jeffrey Solomon, Jr., and Anthony J. Zukaitis. “The MCNPTools Package: Installation and use”. doi:10.2172/1884737. https://www.osti.gov/biblio/1884737.