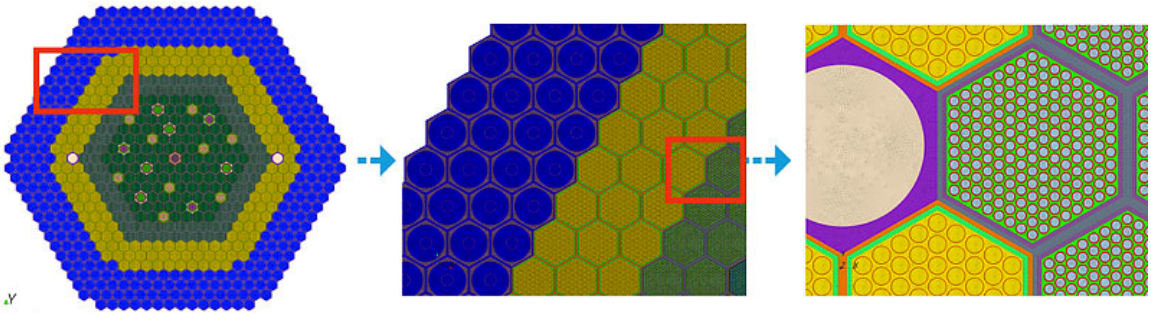
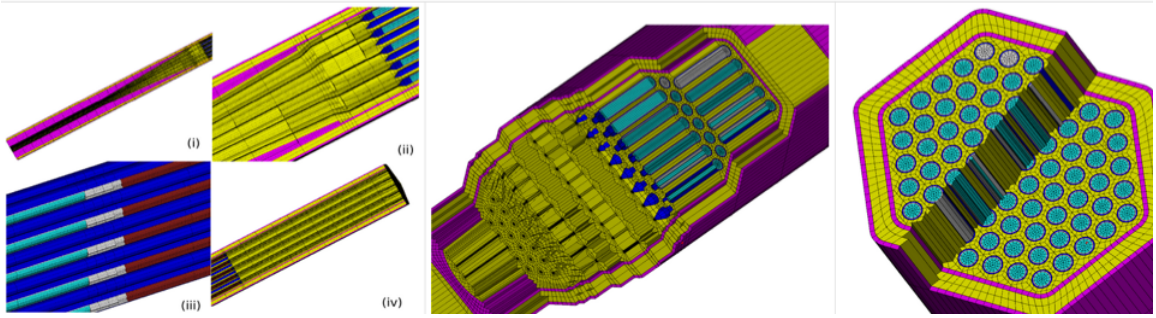
MeshKit Component



Core Model with 101 Million Hex Elements Generated Using RGG

Reactor Geometry (&Mesh) Generator Users Manual – 17th Feb 2017



XX09 Assembly Model With Conical Fuel Pins Generated Using RGG

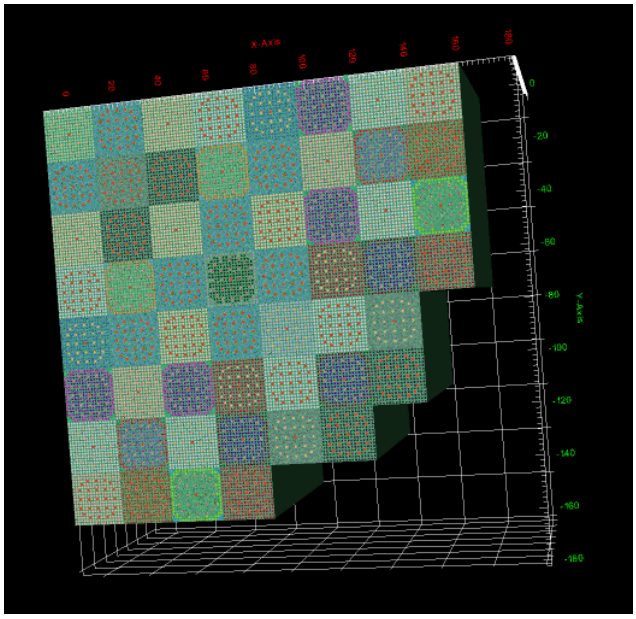
Rajeev Jain
Mathematics and Computer Sciences Division
Argonne National Laboratory
rajeeja@gmail.com

Table of Contents

1 Summary

Different physics such as neutron transport, fluid-flow, thermal expansion and heat transfer must be studied to fully understand the performance and safety aspects of nuclear reactors. The Reactor Geometry (and Mesh) Generator (RGG) [1] [2] [3] toolkit enables the creation of detailed large and complicated reactor models (geometry/mesh) for different types of physics simulations. RGG uses a lattice hierarchy-based approach to create these reactor core models and has been designed to scale on 1000s of processors to create large models with over 1 billion hexahedral elements. For using RGG ***"NO PRIOR MESH GENERATION EXPERTISE IS REQUIRED".***

This document will very briefly describe the core philosophy of RGG, detail the input file language of RGG tools AssyGen and CoreGen and also provide installation instructions. The RGG toolkit is a part of MeshKit library developed by SIGMA at Argonne National Laboratory.



A PWR reactor core model created using RGG

2 Introduction

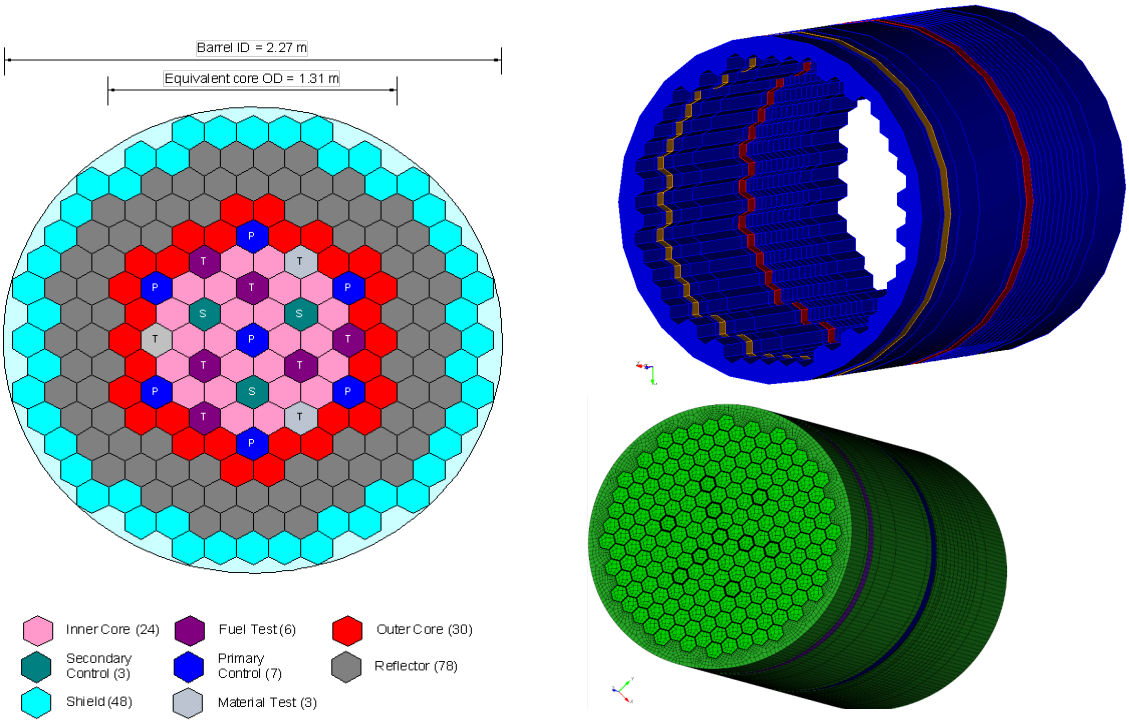
The Reactor Geometry (and mesh) Generator (RGG) toolkit, is a part of the open-source MeshKit library developed at Argonne National Laboratory (ANL). For contributions to MeshKit or RGG use our git repository maintained here: https://bitbucket.org/fathomteam/meshkit/. The RGG toolkit has been used to create PWR, ABTR, VHTR, MONJU and several other types of reactor geometry and mesh models for consumption in state-of-art physics solvers. Documentation and latest news associated with MeshKit and other libraries that MeshKit relies on are available here: http://sigma.mcs.anl.gov/

The following papers explain the RGG methodology and results obtained using RGG:

1. 1. Jain, Rajeev, and Tautges, T. J. (2014). Generating Unstructured Nuclear Reactor Core Meshes in Parallel. In *Proceedings of the 23nd International Meshing Roundtable* (pp. 351-363). http://dx.doi.org/10.1016/j.proeng.2014.10.396

2. Tautges, T. J., and Jain, Rajeev (2012). Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach. *Engineering with Computers*, *28*(4), 319–329. http://link.springer.com/article/10.1007/s00366-011-0236-8

3. Jain, Rajeev, Mahadevan, Vijay and O'bara, Robert. (2015). Simplifying workflow for reactor assembly and full-core modeling. Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, TN, USA. www.mcs.anl.gov/papers/P5320-0315.pdf



ABTR core model created using RGG

3 Installation

See Appendix C for commands used for installation of RGG on a Linux cluster. We are working on completely automating the process for installation of RGG. It must be noted that users MUST have Trelis or CUBIT for using RGG. If you encounter any issues please use this site to report: https://bitbucket.org/fathomteam/meshkit/issues?status=new&status=open

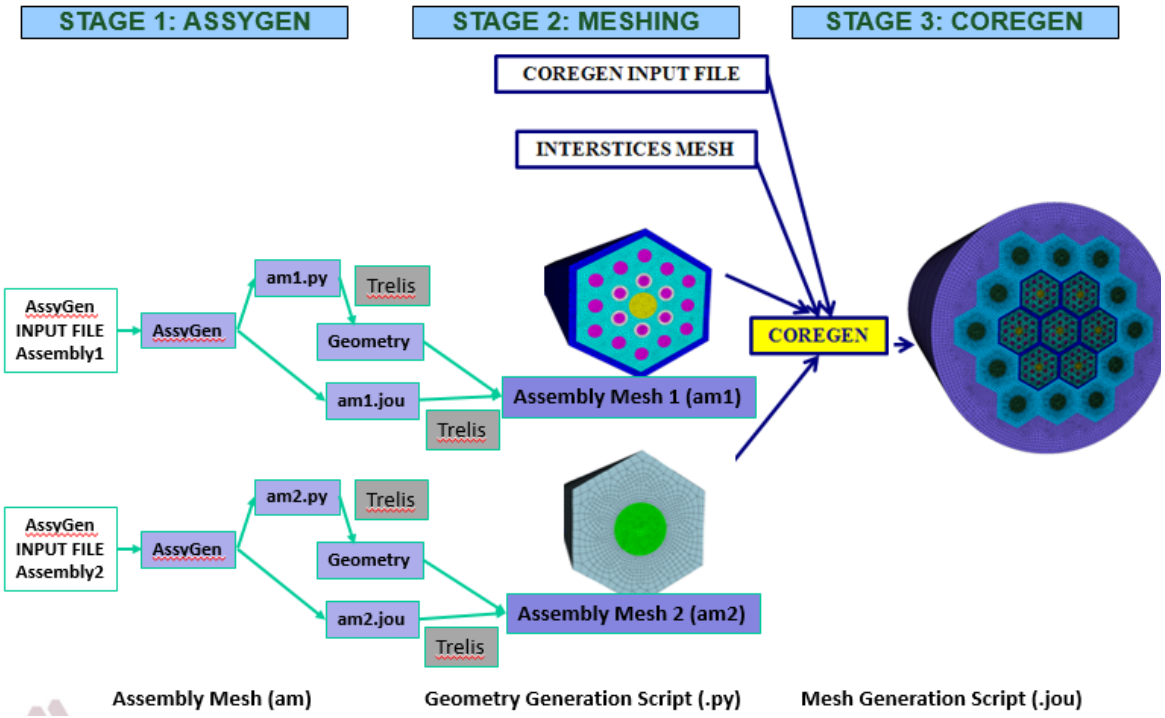
4 RGG Workflow

4.1 3 Stages

Papers listed in section 1 provide much more details on the RGG workflow. The figure below shows the 3-stage workflow for creating core models using RGG.

4.1.1 Stage1: AssyGen

AssyGen is the first step of the three-step core mesh creation process implemented in RGG. In the first stage, AssyGen reads an input file describing a reactor assembly lattice and generates an OCC-based geometry file, along with a template script and a python script for Trelis/CUBIT for generating the geometry and mesh for the assembly using the Trelis/CUBIT meshing toolkit.



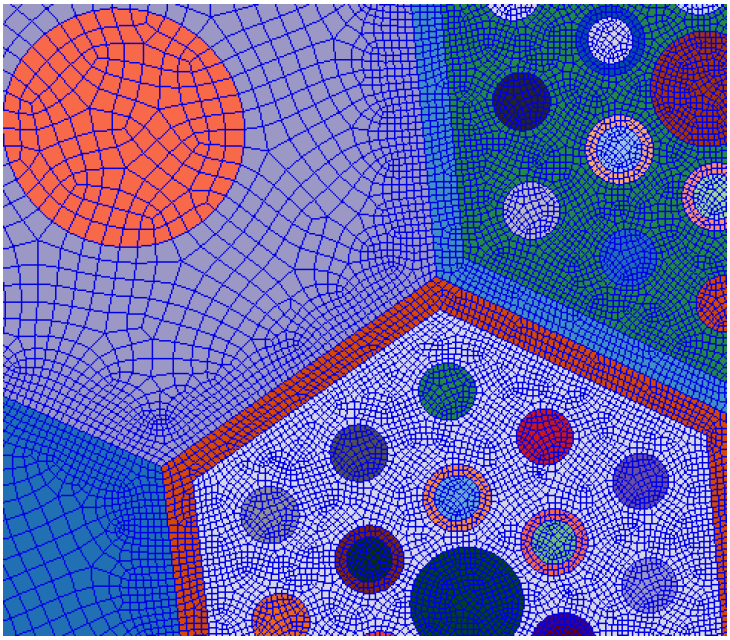
The entire process for creating the model shown below is available in the example folder *simple_hexflatcore* directory. Detailed instructions for running this example are available in section 7.

4.1.2 State 2: Meshing

The second stage is meshing, where the user may choose to perform meshing using the Trelis/CUBIT mesh script generated by AssyGen or using native meshing algorithms or external interfaces to Netgen in MeshKit. AssyGen supports several features, such as axial numbering material and boundary conditions, sectioning, rotation, and information about location of pins. AssyMesher tool in MeshKit is under-development to natively mesh the OpenCascade geometries create by RGG.

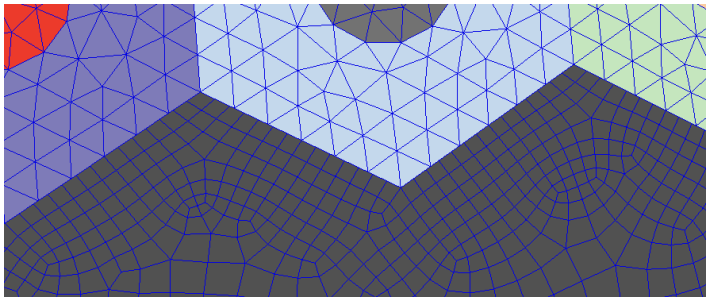
4.1.3 Stage 3: CoreGen

In the third step, the CoreGen tool reads an input file describing the reactor core arrangement (lattice structure) and generates the reactor core mesh or geometry from its component assemblies. Figures below explain the radial mesh sizing features and an invalid (non-conforming) mesh created due to non-matching assembly boundary elements.



Two assemblies with different radial mesh sizes. The one on the left has a radial mesh size of 0.1, while the one on the right has a radial mesh size of 0.3.

These two assemblies have two different radial mesh sizes. The one on the left has a radial mesh size of 0.3, while the one on the right is 0.1. It must be noted that edge interval on the assemblies are set to be equal for a conformal mesh (Figure 25).

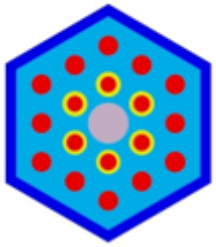


Non-conforming mesh – note how vertices touch edges and edges touch vertices along the border between the blue and hexes.

5 AssyGen Input File

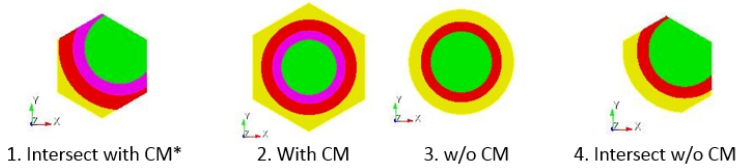
Below is a simple AssyGen input file with the picture of the geometry that is created with the input file, it must be noted that also AssyGen creates detailed meshing scripts for this model. The meshing scripts contains all necessary material and boundary conditions required for setting up reactor simulations.

```
! KEYWORD BASED ASSYGEN INPUT FILE
GeometryType Hexagonal
Materials 5 CR C Fuel F Dt1 D1 Dt2 D2 Mat M1
Duct 2 0.0 0.0  0.0 79.0 9.0 10.0  D1 D2
Pincells 3 1.87960
ControlRod CR 1
Cylinder  1  0.0 0.0 0.0 79.0 1.0 C
FuelCell_1     F2  1
Cylinder  1  0.0 0.0 0.0 79.0 0.5 F
FuelCell_2     F1  1
Cylinder  2  0.0 0.0  0.0 79.0  0.4 0.6 F M1
Assembly 3
    F2  F2  F2
  F2  F1  F1  F2
F2  F1  CR  F1  F2
  F2  F1  F1  F2
    F2  F2  F2
RadialMeshSize 0.15
Rotate Z -30
Center
END
```



5.1 Pincell Types

There are several types of fuel or instrumentation pins that supported by RGG. Figure below shows 4 pincell types with hexagonal lattice. The AssyGen input is highlighted for the 1st example (Intersect with CM*)



1. Intersect with CM* 2. With CM 3. w/o CM 4. Intersect w/o CM

```
! 1. Intersect with Cell Material

Pin1 P1 3 INTERSECT
Pitch  2.5  10.0
Cylinder  3     0.5 0.5     0.0 10.0     1.0 1.3 1.7     m1 m2 m3
CellMaterial 0.0 10.0  m4
```

Note: Same variations can be used for a rectangular pincell also.
*CM - CellMaterial

5.2 AssyGen Keyword List

Below is a list of keywords for AssyGen tool. These are used to specify a reactor assembly input file. The file must have a .inp extension. NOTE: common.inp is a reserved file name for RGG process, when present AssyGen loads it for all assemblies that form the core. This avoids repetition of keywords across all assemblies forming the core model.

1. **Geometry**: Assembly geometry can be output as a collection of 2D surfaces, or as a collection of volumes swept out by those surfaces. If this keyword is not used, AssyGen produces a volumetric model by default.

2. **GeometryType**: This keyword defines the geometry type and can take values 'Hexagonal' or 'Rectangular'. GeomType keyword sets the arrangement of cells specified in the 'Assembly' keyword.

3. **GeomEngine**: This keyword can take values 'ACIS' and 'OCC'. Note, specified engine type matches with geometric engine used for building 'iGeom'. The o/p geometry file has extension '.stp' with 'OCC' and '.sat. with 'ACIS'.

4. **MeshType**: AssyGen can create Cubit journal files for 'hex' and 'tet' meshing of volumes. MeshType keyword can have values: 'hex' or 'tet', the default is 'hex'.

5. **Materials**: Materials keyword is used to define all the material names, names are followed by their aliases. First input to this keyword is the number of materials. Material aliases defined here are used in subsequent keywords to assign materials.

6. **Duct**: The layers of duct wall material are described using this keyword, with the first layer of duct corresponding to the background material in the unit cells of the assembly. The first field specifies the number of layers in the duct wall, followed by the X, Y location of center of the duct wall layers, and the starting and ending Z coordinates for all duct wall material. The width of the layers in X and Y (rectangular) or the flat-to-flat distance (hexagonal)appears next. Next, the material aliases for each layer of background material are specified.

7. **Pincells**: A pincell corresponds to a unit cell type as described in Section 3. This keyword's first input field is the number of pincells. The next keyword is the pitch (hexagonal) or X and Y pitch (rectangular). Alternatively; all pincells can define their pitch using 'Pitch' Keyword.

8. 'Pincell Alias': This is not a keyword, it begins with pincell name defined by the user,followed by the pincell alias and finally the number of subsequent lines this pincell will use to define its properties.

9. **Pitch**: For 'Hexagonal' Geometry this keyword has only one field, the pitch of the hexagonal unit cell. For 'Rectangular' geometry this keyword takes 3 inputs – the X and Y pitch of the unit cell, and the length, width and height of the pincell.

10. **Cylinder**: First input to this command is the number of concentric cylinders in the pincell. Next X, Y of the center and the Z start and Z end of the cylinders is specified. Finally the radii and material alias for the cylinders are specified.

11. **CellMaterial:** This creates a background material for the unit cell, and results in unit cell boundaries appearing explicitly in the assembly. The starting and ending axial position is specified, along with the material alias for the background material.

12. **Assembly:** The lattice arrangement is specified using this keyword. For rectangular lattices,the first two fields indicate the number of unit cells in the X and Y directions; for hexagonal lattices, the first field indicates the number of rings in the lattice.

13. <cell_alias>: Unit cell aliases are given, one line of aliases per row in the assembly starting from top left proceeding to bottom right.

14. **Rotate:** Rotates the entire assembly model by the specified angle (in degrees) around the specified axis. The second keyword is axis and the third is angle is degree.

15. **Center:** If no axis is specified, it centers the assembly about X and Y axis. 'X' or 'Y' or 'Z' can be specified as the second keyword for centering the assembly about a specific axis.

16. **Section:** Subtracts everything on the negative side of the specified. if Reverse is given, the positive side of that axis. If offset is given, model is sectioned at that coordinate on the specified axis, instead of at the zero value.

17. **Move**: This keyword is followed by three double precision numbers indicating the distance(in X, Y and Z direction) by which the entire model is moved.

18. **AxialMeshSize:** Mesh size in the axial direction; for assemblies where the "Geoemtry Surface" keyword is not specified, this size is specified in the meshing script. When varying axial volumes are created an axial mesh size must be specified for axial regions.

19. **RadialMeshSize**: Mesh size in the radial direction; this size is written to the journal file for meshing the assembly model.

20. **TetMeshSize:** This keyword is used to specify a double precision number specifying the 'tetmesh' size for the entire model.

21. **NeumannSet_StartId**: When creating core mesh, individual assembly meshes mush have different Neumann set Id's. This keyword allows the specification of an integer that marks the startid of Neumann sets in the assembly mesh.

22. **MaterialSet_StartId**: Similar to Neumann sets, Material sets, must also have different Id's. This keyword allows the specification of an integer that marks the startid of material sets in the assembly mesh.

23. **End**: This command marks the end of AssyGen input file.

24. **EdgeInterval**: Specify the interval on outer edges of the assembly, often needed to constraint meshing on the skin of the model

25. **MergeTolerance**: Tolerance of merging geometric entities in the model.

26. **CreateSideSet**: Option to disable sideset creation for CUBIT journal file. Takes values 'yes' or 'no'.

27. **CreateFiles:** This creates assygen input files with block numbers shifted. It is useful, when multiple assemblies with same configuration and different materials is desired.

28. **MeshScheme**: For concentric cylinder or outermost ducts scheme (CUBIT) "hole" is often desired. This keyword when specified sets all such concentric surfaces to scheme hole. Default scheme is pave.

29. **StartPinId:** When using info keyword (on), this keyword can be used to define the start pin id's for a particular assembly.

30. **HBlock:** When material blocks are created by using hex elements varying in Z direction HBlock is used. This keyword takes 3 arguments – number of blocks long Z, start Z height and end Z height.

32. **SaveExodus:** Flag to indicate that final assembly must be saved as a .exo file. Default is .h5m.

33. **MeshScheme:** Takes a string: "Hole" or "Pave". Default is Pave. Some assemblies with too many concentric cylinders that are closely spaced are hard to mesh using 'pave' scheme, so scheme 'hole' must be used.

34. **NumSuperBlocks:** Superblocks is specified when some of the material blocks must be combined to form a super block. This keywords takes in the total number of such block required by the assembly model.

35. **Superblocks:** This keyword takes several 3 arguments: Id, alias and contents of the superblock.

36. **List_MaterialSet_StartId:** Only used when using superblocks and createfiles option together, this keyword takes in an integer specifying the startid of material set in a particular AssyGen file.

37. **List_NeumannSet_StartId:** Only used when using superblocks and createfiles option together, this keyword takes in an integer specifying the startid of Neumann set in a particular AssyGen file.

38. **BLMaterials**: After defining materials BLMaterials can be specified as "BLMaterials 1 Fluid2 0.1 4", where 1 indicates the number of material for boundary layers, after this for each material – material name, boundary layer thickness and number of layers are specified. See example "simple_rectcore" s1.inp, uncomment this line and visualize s1 assembly mesh finally created.

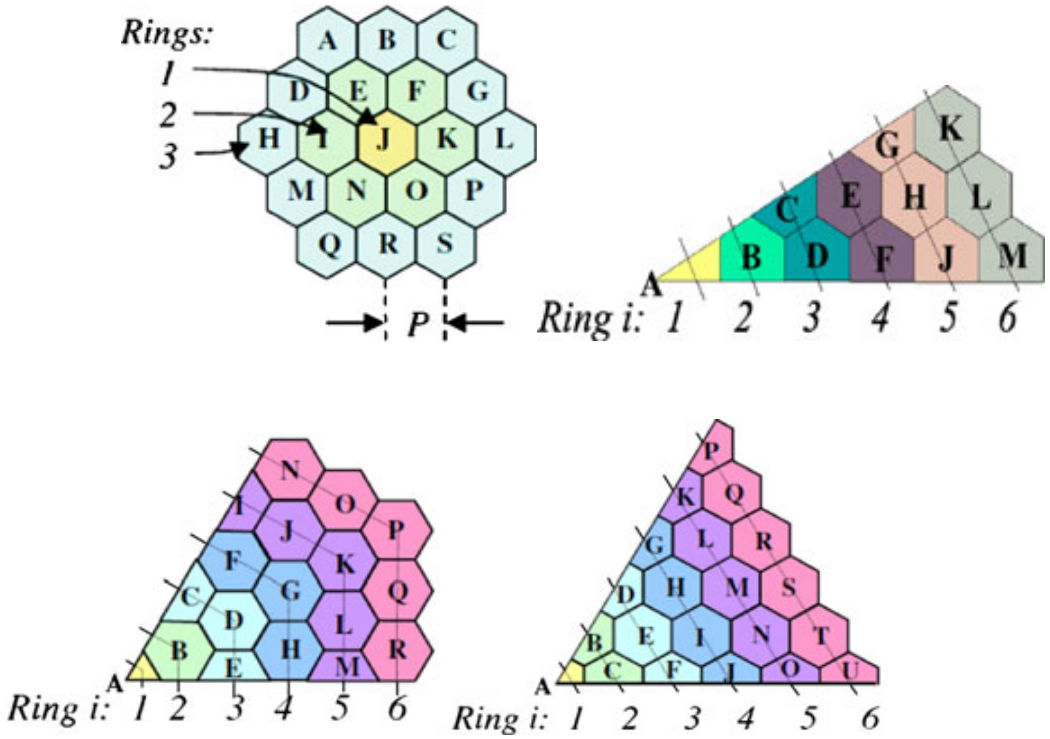
Note: '&' sign can be used for continuing the line input on the next line.

6 CoreGen Input File

Section 6.1 shows the 4 different types of hexagonal cores supported by CoreGen. Section 6.2 presents a list of keywords for CoreGen tool. These are used to specify a reactor core input file. The file must have a .inp extension.

6.1 Types of Hexagonal Cores

RGG has built-in support for a variety of hexagonal-based cores, including full hexagonal cores, $1/6^{th}$ hexagonal flat cores, $1/6^{th}$ hexagonal vertex cores, and $1/12^{th}$ hexagonal cores.



6.2 CoreGen Keyword List

1. **Geometry**: Surface or volume core models can be generated; if unspecified, the default is to generate a volume model.

2. **GeometryType**: The GeometryType is specified as rectangular, hexflat, or hexvertex (see section 2 of this paper for a description of the hexagonal lattice types).

3. **GeomEngine**: This keyword typically takes the 'OCC'. Note, specified engine type matches with geometric engine used for building 'iGeom'. The o/p geometry file has an extension of '.stp' with 'OCC' or '.facet' with 'Facet' engines.

4. **Extrude**: When using this keyword, the 'Geometry' must be 'surface. The next two keywords define the height of the model in Z-direction and the number of divisions in Z-direction.

5. **Assemblies**: This keyword specifies the number of different assembly types, and the pitch of the lattice arrangement (two values for a rectangular lattice, and one for a hexagonal lattice).

6. <mesh_file>: For each assembly type, a file name containing the mesh for that assembly type and the alias used to reference that assembly type appear on a separate line.

7. **Lattice**: The number of assemblies in the X and Y directions (GeometryType: Rectangular) or the number of rings (GeometryType: Hexflat or Hexvertex) in the core.

8. <assy_alias>: Assembly aliases are listed in order of appearance in the core lattice of the specified type, as described in Section 2.2. Note the number of assemblies in a given core depends on the lattice type and the parameters specified with the Lattice keyword. Assembly aliases in the lattice should all appear on one line. Line continuation characters '&' can be used to break lines in the file, to make the input file easier to read.

9. **MergeTolerance**: This is the tolerance value used for merging the nodes of assembly meshes.

10. **NeumannSet**: It can take values 'top', 'bot' 'side' or 'sideall'. This keyword is used to create top, bottom and side surface sidesets for the entire core model. When using 'side' keyword equation of the plane (XY plane, since Z direction is along the height – always) ax + by + c. For the example below NeumannSet 101 will be created on a plane along the line specified by x + 11.75 = 0
Example usages:
NeumannSet **Top 99**
NeumannSet **SideAll 100**
NeumannSet **Side 101 x 1 y 0 c 11.75**

11. **OutputFileName**: OutputFileName with extension can be set using this keyword. If not specified, o/p file name is <coregen_input_file>.h5m

12. **Background**: If specified, this keyword specifies an assembly mesh that fills interstices regions between assemblies, along with any structure surrounding the core, such as core barrel or reactor vessel.

13. **End**: This command marks the end of CoreGen input file.

14. **ProblemType:** Copy/move operations can be performed on 'Mesh' or 'Geometry'. Default value of this keyword is 'Mesh'. When using geometry files in 'Assemblies' keyword this keyword should be setup to copy/move geometry.

15. **SaveParallel**: When using parallel-capable CoreGen. This keywords offer enables option for saving mesh file.
Takes values 'Multiple', 'Both' and 'One
ONE: one file per processor
MULTIPLE: one file per processor and one combined file
ONE: one combined file (Default)

16. **INFO**: It takes arguments ON or OFF. This dumps a .csv file with centroid's of assembly locations.

17. **MeshINFO**: On or Off. This dumps a file with coordinates of centroids of all the elements, followed by the pin and assembly number corresponding to this element.

18. **Symmetry:** Takes in an integer. For rectangular assemblies it is 1. For hexagonal assemblies it can take values 6 (1/6 core) , 12 (1/12th core) and 1 (full core). This keyword sets the array sizes for assemblies specified using the "Lattice" keyword.

7 Running Examples

There are very detailed examples demonstrating all the available capability of RGG. There are available as a part of the installation. All these examples can be used to create various types of assembly and core models shown in this document.

As of 17th Feb 2017 there are 8 example folders as listed below:

1. **coregen_file_shift**
2. **simple_hexflatcore**
3. **simple_rectcore**
4. **sixth_hexflatcore**
5. **sixth_hexvertexcore**
6. **twelfth_hexflatcore**
7. **vhtr-2d-extrude**
8. **vhtr-hexvertex**

Each example folder consists of a makefile that guides the entire mesh generation process. The users might have to run make multiple times. Finally, a core model will be created in each of the directories.

7.1 Displaying and Manipulating the Mesh

Upon successful creation of a mesh, the core model is typically saved in .h5m file format and can be visualized using VisIt developed by LLNL.

Alternatively, the .h5m file can also be converted to .exo file using the mbconvert tool in MOAB. The exodus file can be visualized using ParaView/VisIt. Modification of the final .exo file can also be performed in any preprocessing software that reads/writes exodus mesh file format.

Simulation scientists must inspect the mesh for the following parameters:

- Volumes
- Boundary
- Surfaces
- Neumann Sets
- Dirichlet Sets
- Material Sets

For parallel simulations, the mbpart tool in MOAB can to partition the final .h5m mesh created. See commands in Appendix C for sample mbpart command. The final partitions can also be visualized using VisIt.

8 Other Core Types and Features

MeshKit and RGG are tools that were created to aid and save model creation time for simulation scientists. The overall objective is to provide automation with carefully devised points for user intervention. This section list RGG related development and caveats.

8.1 RGG GUI and Fully Open-Source Alternative

Fully open-source meshing algorithms such as parallel quad-paver, swept mesh generator as a part of MeshKit are under development. These meshing algorithms will allow RGG to be completely open-source. Moreover, this work will also enable control of the fine-grained parameters for better quality of the final mesh produced by RGG. Currently, the parallel performance bottleneck in RGG is serial meshing of the reactor assembly geometries; this parallel quadrilateral-meshing algorithm will also alleviate that bottleneck.

8.2 Boundary Layer Generation

RGG offers interfaces to add boundary layers around a particular material with a user specified thickness, intervals and tolerance. See Appendix D for details on keywords and an example of boundary layer creation. RGG can also utilizes the MeshKit/PostBL [4] algorithm to add boundary layers after the final core model has been generated by CoreGen tool.

8.3 ATR and Other Lattice Types

ATR uses a circular pattern, which can be described as a lattice, RGG currently does not support ATR modeling. Development of CMB [6] for the general geometry and meshing tool that is currently under development and can be used for modeling and meshing of ATR and other non-symmetric types reactor core models.

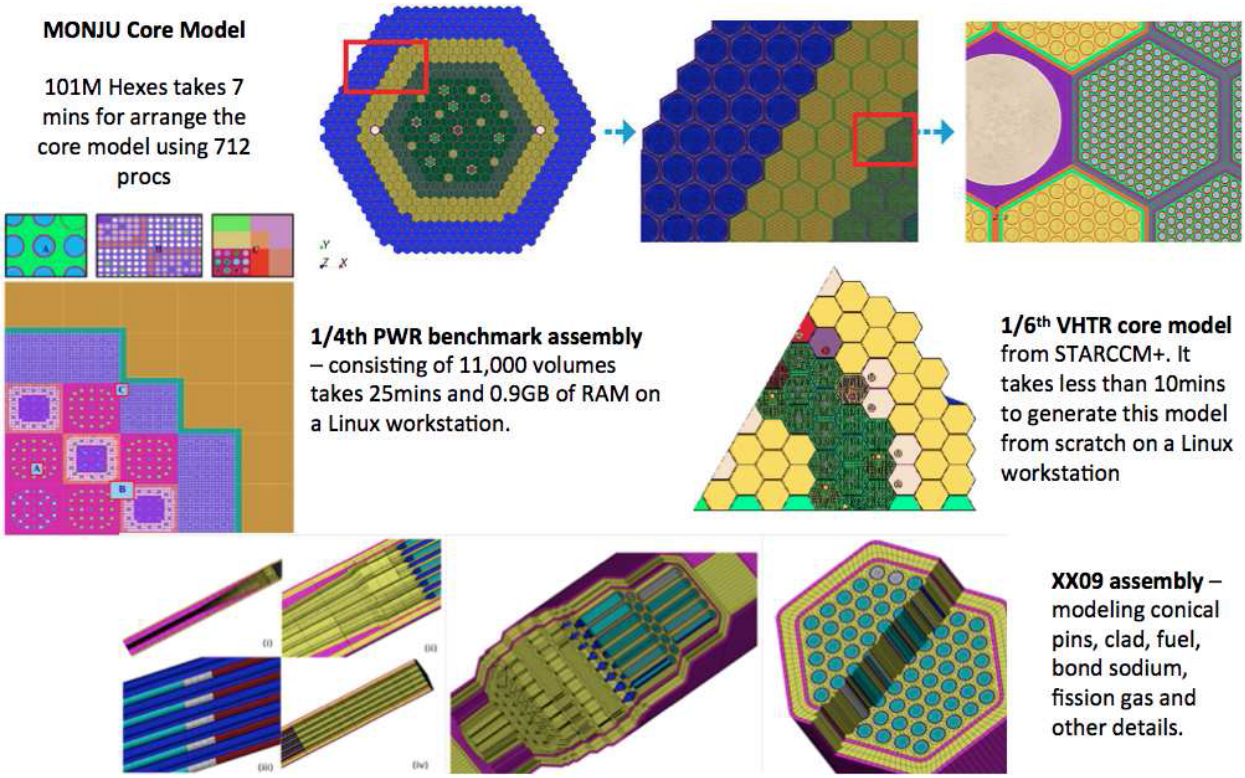
Acknowledgments

We thank the SIGMA group at Argonne, who maintain the libraries required by MeshKit and Kitware Inc. for collaborating on the development of RGG GUI application. This material was based on work supported in part by the U.S. Department of Energy, Office of Nuclear Energy, Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program and by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program and by the U.S. Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

References

1. Jain, Rajeev, and Tautges, T. J. (2014). Generating Unstructured Nuclear Reactor Core Meshes in Parallel. In *Proceedings of the 23nd International Meshing Roundtable* (pp. 351-363). http://dx.doi.org/10.1016/j.proeng.2014.10.396
2. Tautges, T. J., and Jain, Rajeev (2012). Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach. *Engineering with Computers*, *28*(4), 319–329.
3. Jain, Rajeev, Mahadevan, Vijay and O'bara, Robert. (2015). Simplifying workflow for reactor assembly and full-core modeling. Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, TN, USA.
4. Jain, Rajeev, and Tautges, T. J. (2014). PostBL: Post-mesh boundary layer generation tool. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 445-464).
5. Buildbot: http://gnep.mcs.anl.gov:8010/ and MeshKit doxygen page: http://ftp.mcs.anl.gov/pub/fathom/meshkit-docs/index.html
6. http://computationalmodelbuilder.org

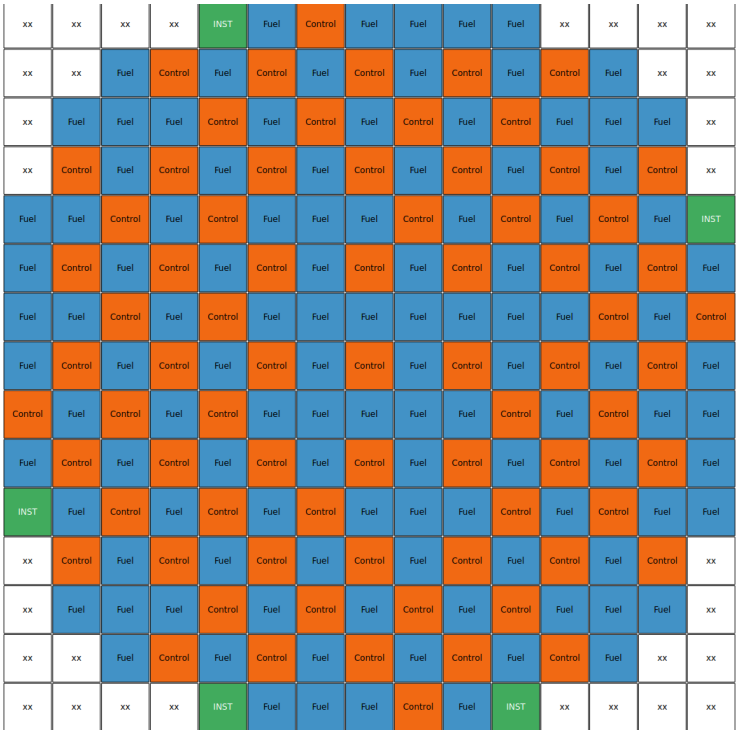
Appendix A: Few Actual Reactor Models Created Using RGG GUI



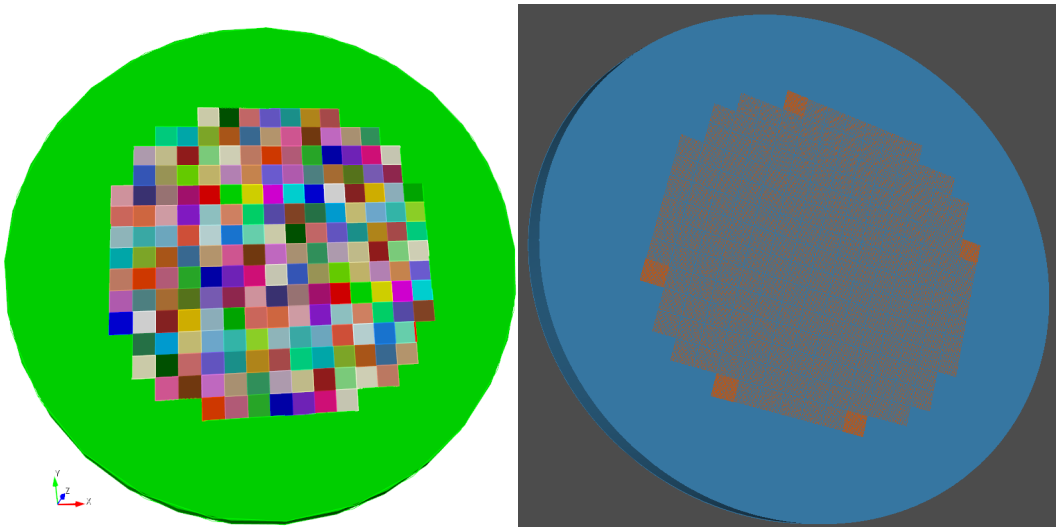
Highlights of CPU time taken along with models details for MONJU, 1/4th PWR, 1/6th VHTR and XX09 (EBR2 core) assembly.

Appendix B: Westinghouse Four Loop PWR Core

The Westinghouse pressurized water reactor nuclear power plant document available online [13], describes the reactor core model shown in Fig. 5(a). The core model consists of 193 total assemblies, and geometrically there are three different types of assemblies. Core model shown in Fig. 4(a) and 4(b) are formed with assemblies with fewer pincells (1x1, 3x3 and 4x4). In the actual model, each assembly is a 17x17 lattice (shown in Fig. 3(b)), the complete reactor core model shown in Fig. 5(c). The final mesh model is hard to visualize since the mesh size is too small for viewing and hence only the geometry and a simplified model (Fig. 4) are shown here. The complete core creation for this model is automatic i.e., small changes in material, assembly or pin cell arrangement and the creation of the outer vessel are automatically handled by RGG. It must however be noted that specialized geometries such as grid spacer creation are not supported implicitly yet and must be modeled separately. While modeling the outer vessel or other external meshes, the surfaces/curves where the reactor assembly and external pieces meet must have coincident nodes for merge process to work and conformal core mesh model. If this is not the case, the CoreGen process will fail with appropriate error messages. On 192 CPU cores, the final mesh model for this core containing 6M 3D hexagonal elements, which can be created in less than 10 mins from scratch. A majority of this time is taken in the serial assembly and interstices mesh generation process. Parallel CoreGen step takes only 90 seconds to assemble the core and create the final mesh, where the majority of the time is taken by parallel merge and save steps. The maximum memory used by a processor during the mesh generation process for this model was about 620MB.



(a)



(b, c)

2D layout of the four-loop reactor core (editable using RGG GUI) (b) CoreGen geometric core creation is shown with homogenized assemblies, (c) Geometric model with 193 hetrogenous assemblies and outer core vessel (assemblies have 17x17 pins).

Appendix C: Installation Commands on a Linux Cluster

Set your .bashrc as follows:

```
source /opt/intel/compilers_and_libraries/linux/bin/compilervars.sh intel64
export MPI_DIR=/usr/local/
PATH=/opt/Trelis-16.2/bin:/home/rajeev.jain/install/moab/bin:/home/rajeev.jain/install/meshkit/bin:$PATH
LD_LIBRARY_PATH=/usr/local/lib/:/usr/local/bin/:$LD_LIBRARY_PATH
```

Run the following commands:

```
  33  unset SSH_ASKPASS
  34  git clone https://rajeeja@bitbucket.org/fathomteam/cgm.git
  36  git clone https://rajeeja@bitbucket.org/fathomteam/moab.git
  37  git clone https://rajeeja@bitbucket.org/fathomteam/meshkit.git

 270  wget http://www.cs.sandia.gov/~kddevin/Zoltan_Distributions/zoltan_distrib_v3.83.tar.gz
 278  tar -xvf zoltan_distrib_v3.83.tar.gz
 280  cd Zoltan_v3.83/
 283  autoreconf -fi
 284  mkdir build
 285  head ~/moab/build1/config.log
 286  mkdir build
 287  cd build/
 289  ../configure --prefix=/home/rajeev.jain/install/zoltan CC=/usr/local//bin/mpicc
CXX=/usr/local//bin/mpicxx F77=/usr/local//bin/mpif77 FC=/usr/local//bin/mpif9 --enable-optimize
 290  make -j4
 291  make install
  43  cd cgm/
  44  autoreconf -fi
  47  ls
  48  mkdir build1
  49  cd build1/
  62  ../configure --prefix=/home/rajeev.jain/install/cgm CC=$MPI_DIR/bin/mpicc
CXX=$MPI_DIR/bin/mpicxx F77=$MPI_DIR/bin/mpif77  FC=$MPI_DIR/bin/mpif90 --disable-debug --
enable-optimize
  64  make -j4
  65  make install
  66  make check
  89  cd moab/
  90  autoreconf -fi
  91  mkdir build1
  92  cd build1/
  99 ../configure --prefix=/home/rajeev.jain/install/moab --with-cgm=/home/rajeev.jain/install/cgm/ --
download-hdf5 --download-netcdf --download-metis --download-parmetis CC=/usr/local//bin/mpicc
```

CXX=/usr/local//bin/mpicxx F77=/usr/local//bin/mpif77 FC=/usr/local//bin/mpif90 --disable-debug --enable-optimize --enable-imesh --enable-irel --with-zoltan=/home/rajeev.jain/install/zoltan/;make -j4;make install

106 cd meshkit/

107 autoreconf -fi

108 mkdir build1

109 cd build1/

150 ../configure --prefix=/home/rajeev.jain/install/meshkit --with-igeom=/home/rajeev.jain/install/cgm/ --with-imesh=/home/rajeev.jain/install/moab/ --download-netgen CC=/usr/local//bin/mpicc CXX=/usr/local//bin/mpicxx F77=/usr/local//bin/mpif77 FC=/usr/local//bin/mpif90 --disable-debug --enable-optimize --enable-rgg16

151 make -j8

152 make install

156 cd install/meshkit/examples/rgg/simple_rectcore/

set trelis path in makefile

179 vi makefile

178 make s1.cub

180 make s1.cub

181 /home/rajeev.jain/install/moab/bin/mbsize s1.cub

182 make s2.cub

183 make

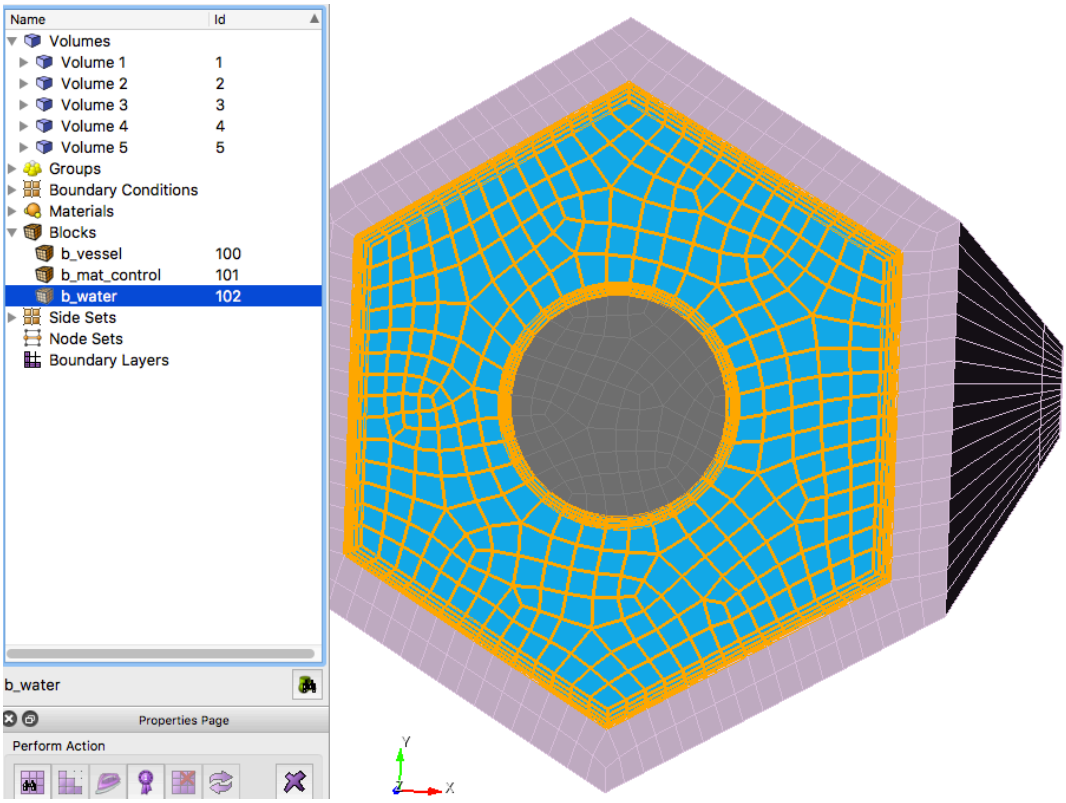
184 /home/rajeev.jain/install/moab/bin/mbsize simple_rectcore.h5m

185 /home/rajeev.jain/install/moab/bin/mbsize -t simple_rectcore.h5m

Appendix D: Automatic Boundary Layer Creation for CFD

```
! ########################################################
!   This file will create two materials:
! 100 - Duct will have 4 layers at the end as boundary layer
! 101 - is a normal water cylindrical duct
! ########################################################
GeometryType Hexagonal
Geometry Volume
Materials 5 Vessel VL Mat_Control Mat_Control water water water_bl1 water_bl1 water_bl2 water_bl2
BLMaterials 2 water_bl1 0.2 3 water_bl2 0.7 4
duct 3 0.00000 0.00000 0.00000 100.000 9.40000 10.0000 12.0000 water water_bl2 VL
pincells 1
Pin1 P1 2
pitch 10.0000 0
cylinder 2 0.00000 0.00000 0.00000 100.000 2.00000 2.20000 Mat_Control water_bl1
Assembly 1
P1
RadialMeshSize 0.40000
Center
Rotate Z 30.0000
AxialMeshSize 50.0000
meshtype Hex
Scheme Hole
neumannset_startid 100
materialset_startid 100
end
```

The above AssyGen input file creates the mesh below, the highlighted light orange is material 3 b_water.



Boundary layer elements near fuel pin and near the duct – water material highlighted