

Update on Development of Mesh Generation Algorithms in MeshKit

Mathematics and Computer Science Division

About Argonne National Laboratory

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

DOCUMENT AVAILABILITY

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

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

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

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

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

Disclaimer

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

Update on Development of Mesh Generation Algorithms in MeshKit

prepared by
Rajeev Jain
Evan Vanderzee
Vijay Mahadevan
Mathematics and Computer Science Division, Argonne National Laboratory

September 30, 2015

SUMMARY

MeshKit uses a graph-based design for coding all its meshing algorithms, which includes the Reactor Geometry (and mesh) Generation (RGG) algorithms. This report highlights the developmental updates of all the algorithms, results and future work. Parallel versions of algorithms, documentation and performance results are reported. RGG GUI design was updated to incorporate new features requested by the users; boundary layer generation and parallel RGG support were added to the GUI. Key contributions to the release, upgrade and maintenance of other SIGMA¹ libraries (CGM and MOAB) were made. Several fundamental meshing algorithms for creating a robust parallel meshing pipeline in MeshKit are under development. Results and current status of automated, open-source and high quality nuclear reactor assembly mesh generation algorithms such as trimesher, quadmesher, interval matching and multi-sweeper are reported.

¹<http://www.sigma.mcs.anl.gov>

TABLE OF CONTENTS

Summary	i
Table of Contents	ii
List of Figures	iii
1 Introduction.....	4
2 RGG Enhancements and Reactor Assembly Mesh Generation (AssyMesher)	4
3 Computational Geometry Model	7
4 Open Source Triangle Surface Meshing.....	8
4.1 GRUMMP	8
4.2 NetGen	9
5 Facet-Based Mesh Reader	10
6 Multi-Sweeper	11
6.1 Submapping.....	11
6.1.1 Optimal Corner Assignment.....	11
6.2 Many-to-many Sweeper	12
6.2.1 Edge Patch Imprinting	13
6.2.2 Interior Edge Patch Placement.....	13
6.3 Integration of Sweeping Pieces	13
6.4 Examples	13
7 Conclusions.....	14
Acknowledgments	15
References	15

LIST OF FIGURES

Figure 1. User specified digraph for creating reactor assembly mesh with boundary layers.....	4
Figure 2 In stage 1 AssyGen generates a geometry based on input file. In stage 2 AssyMesher reads the geometry and creates the assembly mesh.....	5
Figure 3. (a) CoreGen MeshOp di-graph. (b) Internal nodes created during setup-phase of CoreGen, shown in box. (c) 2 processor digraph for CoreGen MeshOp.	5
Figure. 4. User specified di-graph with 3 processors and 4 different mesh operations to create a reactor core mesh.	6
Figure 5. Runtime and efficiency plots for CoreGen, UMR and IO.	6
Figure 6. Invalid cylindrical pincell mesh due to geometry representation problem in CGM.	7
Figure 7. User specified digraph for creating reactor assembly mesh with boundary layers.....	8
Figure 8. Right: Car crankshaft in CUBIT. Left: Surface mesh of the crankshaft produced using MeshKit. These two representations can now be linked directly using the new FacetBasedMeshReader.....	11
Figure 9. Structured grid generation of a surface with a three quarters of circle feature by submapping through templates: (a)corner assignment by CUBIT 13.2; (b)structured grid for (a); (c)corner assignment based on templates in this paper; (d)structured grid for (c).....	12
Figure 10. Mesh quality histogram: (a)mesh quality histogram for Figure 1.7(b); (b)mesh quality histogram for Figure 1.7(d)	12
Figure 11. All-hexahedral mesh generation for grid spacer reactor model: (a) assembly geometry model; (b) local zoom-in view of assembly model; (c)single part of grid spacer reactor model; (d)top view for (c); (e)source surface meshes for (c); (f)all-hex meshes for (c); (g)target surface meshes for (c); (h)cut-view of interior node distribution inside the grid spacer model; (i)another cut-view of interior node distribution	14

1 Introduction

We describe in this report the progress made over the past year in incorporating new and state-of-art research algorithms in MeshKit focusing on the major goal of creating nuclear reactor core models. The focus of development was centered on releasing stable version of Reactor Geometry (and mesh) Generator (RGG) GUI with support for boundary layers and parallel reactor core model creation [1]. Considerable efforts were made for development of SIGMA libraries CGM and MOAB as they constitute the geometry and mesh database for MeshKit respectively. New models for Westinghouse PWR reactor core was also created in serial and parallel, details and results of the model are available in our paper [2].

Nuclear reactor assembly mesh generation algorithm requires development of several key algorithms that must work together to create an automated meshing tool. Integration of existing triangle meshing packages NetGen and GRUMMP is in-progress. A new facet-mesh algorithm has been integrated and documented in the doxygen-based documentation framework of MeshKit. Multi-sweep algorithms were developed and compared with CUBIT, integration of this algorithm with native quadmesher and interval assignment is in-progress. Section 2 reports the RGG enhancements, parallel CoreGen and fixes along with development status of open-source reactor mesh generation tool (AssyMesher). Section 3 lists the development efforts and new releases of geometry package CGM. Section 4 highlights the triangle mesher status; Section 5 gives the summary and results of facet-based mesh generation algorithm. Section 6 briefly describes and shows results for the multi-sweeper algorithm and finally, Section 7 lists the conclusion and future work.

2 RGG Enhancements and Reactor Assembly Mesh Generation (AssyMesher)

PostBL(Post mesh Boundary Layer) [3] MeshOp (mesh operation), which generates boundary layer meshes for an already existing mesh model was integrate with RGG GUI application. Figure 1 shows a di-graph based representation for meshing of reactor assembly, Stage 1 (AssyGen) and stage 2 (Meshing) of the three stage RGG workflow [4] (third stage is CoreGen) are shown in Figure 2, where meshing is handled by the AssyMesher MeshOp in MeshKit. AssyMesher is a essentially a combination of state-of-art (open-source) and native algorithms in MeshKit with added automation and knowledge of AssyGen geometry. It is designed to use different meshing algorithms as per the specified MeshKit configuration. CAMAL [5] library (closed source) is currently used and functional for generation of hexahedral meshes on AssyGen generated Open-Cascade geometries. Section 4 and Section 6 highlights the development efforts of open-source algorithms that would be serve as graph node of the AssyMesher MeshOp. Interval assignment algorithms developed in MeshKit would be tied to all these algorithms along with automatic scheme selection for minimal user intervention and intelligent sizing specification. Fully open-source AssyMesher MeshOp is under development.

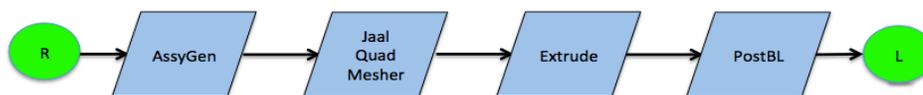


Figure 1. User specified digraph for creating reactor assembly mesh with boundary layers.

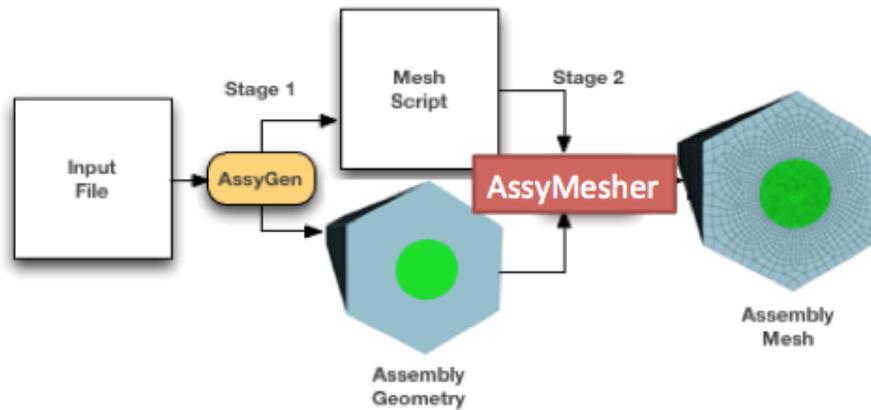


Figure 2 In stage 1 AssyGen generates a geometry based on input file. In stage 2 AssyMesher reads the geometry and creates the assembly mesh.

All RGG algorithms: AssyGen, AssyMesher and CoreGen adhere to the digraph-based design in MeshKit. A simple graph constructed by the user for running CoreGen is shown in Figure 3(a). In the setup traversal phase, CoreGen adds two nodes, “CopyMesh” and “MergeMesh” (shown in the box, Figure 3(b)), to the graph. In the execute phase, based on a user specified input file, the assembly meshes are copy/moved and then merged. After populating the material and boundary conditions for the core model, CoreGen saves the desired core model. Note that in serial CoreGen, CopyMesh does the copy/move of all the assemblies forming the reactor core model. Consider the graph for the same reactor-modeling problem with two processors in Figure 3(c). All the operations—copy/move, merge, boundary/material conditions, setup, and mesh saving—are parallelized, thereby making CoreGen MeshOp truly parallel and enabling creation of large reactor models.

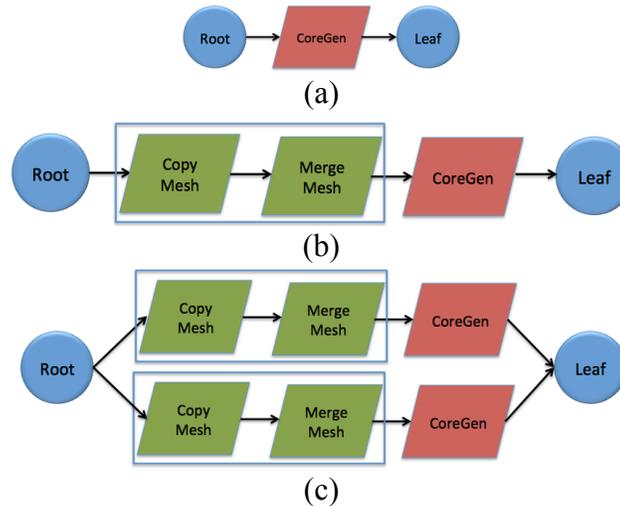


Figure 3. (a) CoreGen MeshOp di-graph. (b) Internal nodes created during setup-phase of CoreGen, shown in box. (c) 2 processor digraph for CoreGen MeshOp.

The parallel graph setup mentioned below demonstrates the creation of reactor core mesh from scratch in one program. In figure 4 four meshing operations (AssyGen, Meshing, PostBL and CoreGen) specified by the user, all form a part of the graph, when running the problem on three processors. A text-based input file provides input parameters for AssyGen,

PostBL and CoreGen. Mesh sizes for geometries created by AssyGen are prescribed using a “sizing function” in the program. The setup phase of CoreGen and Meshing creates more graph nodes that are not shown here. A barrier is required after creation of boundary layers on all the assembly meshes, since CoreGen loads all the assembly meshes that form the core model. Work on open-source AssyGen “Meshing²” or AssyMesher operation is currently in progress. It is a combination of QuadMesh and ExtrudeMesh operation or a CUBIT mesh generation task.

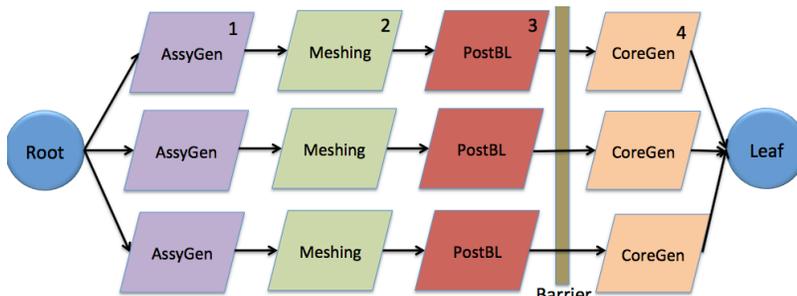


Figure 4. User specified di-graph with 3 processors and 4 different mesh operations to create a reactor core mesh.

Uniform mesh refinement techniques developed in MOAB [6], were utilized in the CoreGen program. After the generation of the core mesh in parallel the resulting mesh on each processor was uniformly refined to levels specified by the user. We studied the performance of our parallel algorithms and I/O routines for rectangular reactor core models. In our weak scalability studies, assemblies were generated with approximately 7500 hexes and subsequently refined two time times resulting in 60K and 480K hexes at each level. The shared entity resolution algorithm used was a geometric proximity based vertex-merge algorithm for both core assemblies and after refinement. We obtained good scaling for this algorithm for upto 1K cores. On the other hand, the parallel IO deteriorated as the number of processors increase (Figure 5). This work is currently in progress and we are investigating the results and also performing new runs for hexagonal reactor cores.

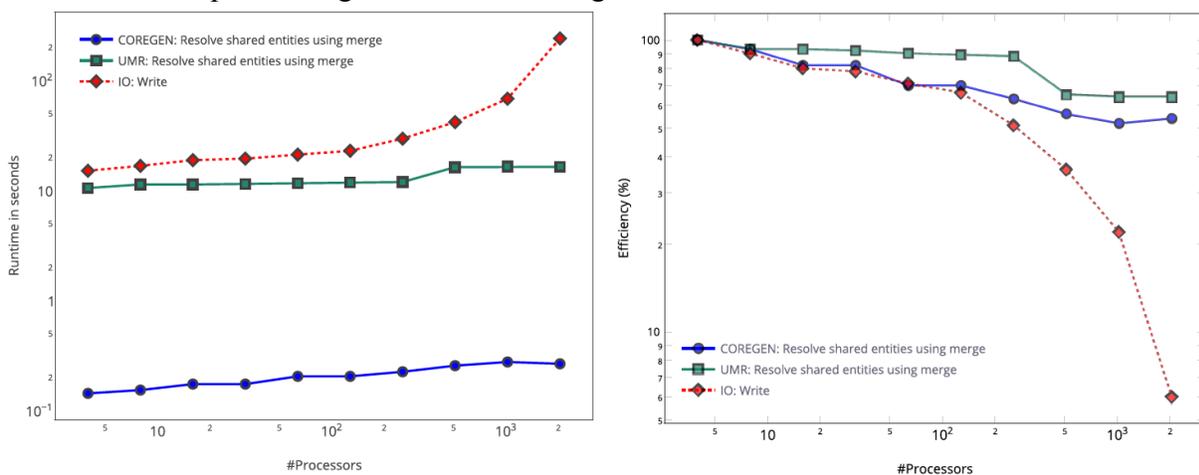


Figure 5. Runtime and efficiency plots for CoreGen, UMR and IO.

3 Computational Geometry Model

The geometry modeling engine that MeshKit uses to represent the geometry that it will mesh is the computational geometry model (CGM). CGM can be built on top of several different geometry libraries, including the CUBIT library from Sandia National Laboratory, which requires a license, and the open source OpenCascade (OCC) library. Historically, the development of CGM and CUBIT were tied together very closely, and CGM works best when it is built with the CUBIT library. CGM does not work as well when it is built with the OpenCascade library. However, there is growing interest in having a robust completely open source build of MeshKit, so work continues to improve the behavior of CGM when it is built with OpenCascade.

In fiscal year 2015, several lines of software development involving CGM built with the OpenCascade library hit a roadblock because of a bug in the way CGM reported the geometric sense of an edge relative to a face and the geometric sense of a face relative to a solid. These are basic queries that report whether the face is to the left or the right of the edge and whether the solid is on the same side as the (positive) normal vector to the face or the opposite side from the normal vector to the face. Figure 6 shows one example of the improper mesh of a cylinder that could result from this problem in CGM.

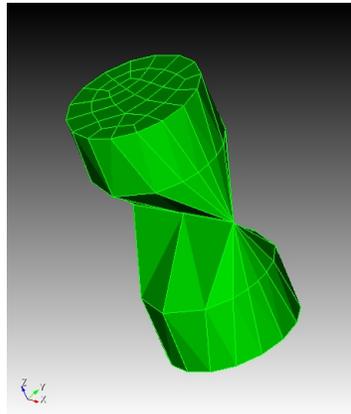


Figure 6. Invalid cylindrical pincell mesh due to geometry representation problem in CGM.

The issue of geometric sense is an important one, but can be confusing because there are a variety of ways to think about it. We resolved the issue for the most common use cases of CGM. Now the same code that once produced the mesh in the Figure 6 produces the mesh shown in Figure 7 when it is run with the CGM library built on the OpenCascade library.

Although no specific instances have been identified yet, software developers suspect that there are some less common use cases where CGM build on top of OCC will report the geometric sense incorrectly. Use cases that involve merging, imprinting, or virtual geometry are areas of concern, but the geometric sense reported in these use cases has not yet been thoroughly tested.

We also continue to keep up to date with the ongoing development of OCC. There were several changes required as we updated CGM to work with more recently released versions of the OpenCascade library.

There are some challenges faced with CGM. There continue to be areas where CGM built on top of OCC does not have the same capability that CGM has when built on top of CUBIT. One of the areas recently discovered is the computation of offset curves. The CUBIT library and OCC library have some differences in the capability they provide, but there is also poor integration of OCC's capability in the CGM software. Another challenge is that the relationship with the CUBIT library, obtaining newer version of CGM and CUBIT are becoming more difficult and thus the integration and new releases of CUBIT. Argonne's CGM developers do not have proper documentation for the most recent releases and are left with trying to guess what arguments are supposed to pass into library methods that have changed.

Some SIGMA software developers have been advocating for significant redesign of the iGeom interface to CGM. There is also a desire to change CGM to allow having multiple instances of CGM within the same thread of the same application. Test coverage is currently quite poor in CGM, and there continue to be memory management and other issues that are flagged by code analysis tools such as valgrind.

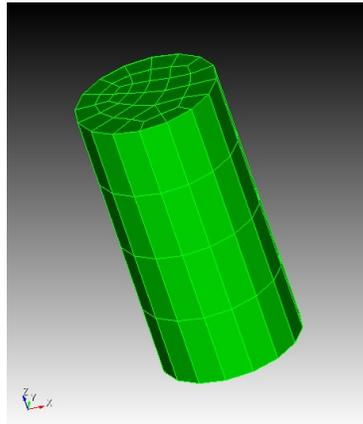


Figure 7. User specified digraph for creating reactor assembly mesh with boundary layers.

4 Open Source Triangle Surface Meshing

MeshKit uses CAMAL as the provider for most of its surface meshing capabilities. It has not had robust open-source implementations of surface meshing capabilities, and one of the aims of MeshKit software development this year was to integrate an open-source solution for triangle surface meshing. The triangle surface mesher might be used as a basis for quadrilateral surface meshing through combining triangles.

4.1 GRUMMP

The project named Generation and Refinement of Unstructured, Mixed-element Meshes in Parallel [7](GRUMMP) seems like a good potential alternative for surface triangle meshing. The algorithmic approach is based on Delaunay refinement rather than the advancing front technique. One challenge to using GRUMMP is that the license is not yet completely open source. In a telephone conversation, Carl Ollivier-Gooch seemed forward to

open up the GRUMMP license so that GRUMMP could be used as an open-source surface triangle mesher, but at the moment the license is still not completely open source.

GRUMMP was originally designed to use the iGeom interface and at one time interacted well with CGM. The release version that is currently provided, however, has not been updated to work with the most recent version of CGM. There was some work done on a nonpublic release of GRUMMP to update it to work with CGM 13.1.

Future work could include further updating that version of GRUMMP to work with later releases of CGM and attempting to integrate GRUMMP into MeshKit.

Another possible avenue for open-source surface triangle meshing would be to provide a new implementation of triangle surface meshing in MeshKit. One possible avenue would be to base that implementation on the advancing front algorithm that is used in NetGen.

4.2 NetGen

The first effort on the open-source triangle surface meshing front was to implement the advancing front triangle surface meshing capability that the NetGen package offers. After making some modifications to the source code of release version 5.2 of NetGen, we were able to build a prototype capability in MeshKit that integrated NetGen's triangle surface meshing capability. However, as discussed in the rest of this section, there were many issues with the integration of NetGen, and it seems that there would not be a satisfactory way to implement NetGen into MeshKit at this time using the interface provided by NetGen. The issues with integrating NetGen include the following.

- NetGen could not be integrated successfully without modifications to the source code by Argonne.
- NetGen needed to read the geometry from file and could not communicate directly with CGM's geometry model that was already in memory.
- NetGen required control over meshing the edges of the surfaces it would mesh.
- NetGen did not provide an interface for specifying a subset of the surfaces to mesh. It always meshed all surfaces of the geometry read from file.
- CGM does not provide a way to save to file a subset of its geometric surfaces and does not provide a means to spin off a second instance, which might otherwise be used to contain copies of a subset of the geometry from the primary instance of CGM and thus work around saving a subset of its geometric surfaces.

The fact that NetGen source code required modification by Argonne does not make it impossible to integrate NetGen, but it would require Argonne to distribute its own modified version of NetGen, which is not desirable.

Since NetGen had to read the geometry from file, it would not have been efficient to use NetGen. Every time NetGen was required to mesh a surface, that surface would have to be written to disk by MeshKit and read from disk by NetGen. Writing to disk is often rather time-consuming and it makes things more complicated in a high-performance computing setting. It does not seem like a good solution when communication at the geometry layer should be possible.

It was not possible to add vertices along edges when communicating with NetGen for NetGen's OCC geometry triangle surface meshing. This meant that NetGen needed to mesh its own edges. This was a problem for multiple reasons. One is that MeshKit had less control over how the edges were meshed. Another is that MeshKit could not integrate NetGen naturally into its graph-based paradigm, which separates the edge meshing from the surface meshing and allows the user to select any desired algorithm for meshing the edges.

Since the OCC triangle meshing capability of NetGen always meshed all of the edges and surfaces of the geometry it read, the only way to get NetGen to mesh a subset of the faces of the full geometric model was to provide a geometry file that contained only a portion of the full geometric model. Although it would be possible to implement this capability in CGM, CGM – and in particular the iGeom interface to CGM that is the only interface to CGM used by MeshKit – does not currently provide a way to export a geometry file that contains only a portion of the full geometry model.

The prototype implementation of NetGen surface meshing provided a way to mesh all edges and all surfaces of a geometric model with NetGen, but it seemed that it would require a significant amount of effort and modification to the iGeom interface, which is supposed to be an interface shared by multiple software vendors, to get to the point where MeshKit could mesh only a subset of the surfaces of a geometric model. At that point, the solution would still have been less than desirable for the other reasons discussed here, so further work on integrating NetGen was suspended.

5 Facet-Based Mesh Reader

When built with iGeom support, MeshKit now has the ability to create surface meshes using the graphics faceting of iGeom's underlying CAD engine. The ability to generate such meshes in MOAB has existed for some time, but the addition of this capability in MeshKit allows one to couple these surface meshes to the underlying geometry entities via MeshKit's mapping of the two via ModelEnts. This allows for creation and surface meshing of geometry in one package. Another advantage of this meshing operation is the possibility of a fully open source workflow in MeshKit algorithms for surface meshing and analysis if building iGeom using OCC, for example.

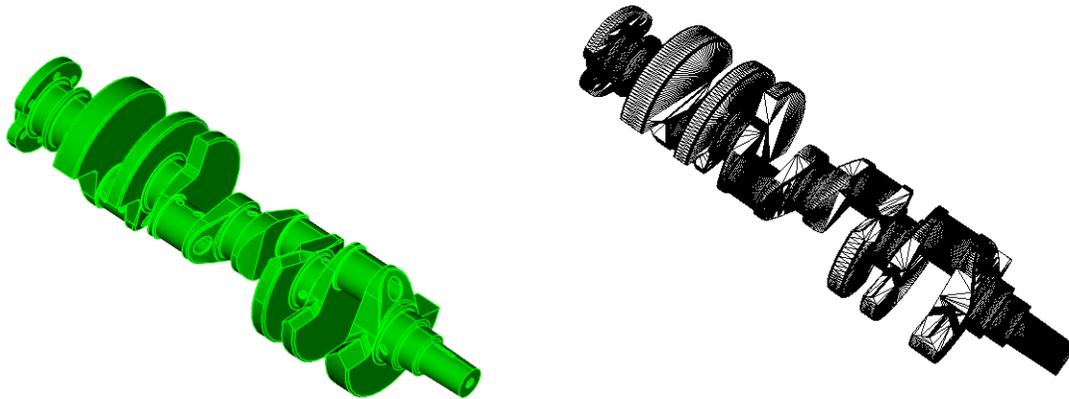


Figure 8. Right: Car crankshaft in CUBIT. Left: Surface mesh of the crankshaft produced using MeshKit. These two representations can now be linked directly using the new FacetBasedMeshReader.

6 Multi-Sweeper

A multi-sweeper consists of several parts, that is, source surface mesh generation, target surface mesh generation, linking surface mesh generation (by submapping) and interior node interpolation inside volumes. A brief overview of the algorithms with some results are given in this section, very detailed results and algorithms are given in a PhD thesis document [8]

6.1 Submapping

Submapping is a meshing tool for generating structured meshes on non 4-sided surfaces and non 6-sided volumes. An important part of submapping is to assign corners in the map to vertices bounding surfaces or edges bounding volumes. TFI mapping MeshOp in MeshKit has been implemented in MeshKit for this purpose.

6.1.1 Optimal Corner Assignment

Submapped surfaces and volume require that corner or edge types bounding a surface or volume sum to four or six, respectively. Templates are used to identify surface submapped corners for a variety of common feature types, and an optimization approach is used to guarantee satisfaction of the submapping constraints ($sum = 4 - 4g$) for submapped surfaces (g is the number of holes). This approach not only works well in the presence of features, but can drastically reduce the user interaction required to set up and mesh models with many features. Vertex type adjustment based on LP.

For the example below, users specify the mesh element size and the algorithm will classify vertices automatically. If there is a multi-connected geometry, it should be virtually decomposed or a path connecting the outmost boundary and interior boundaries should be computed so that the consistent edge parameterization between the outmost boundary and

internal holes can be generated. Note that all the bounding surfaces in all the examples shown below are meshed with structured quadrilateral meshes by the TFI mapping MeshOp.

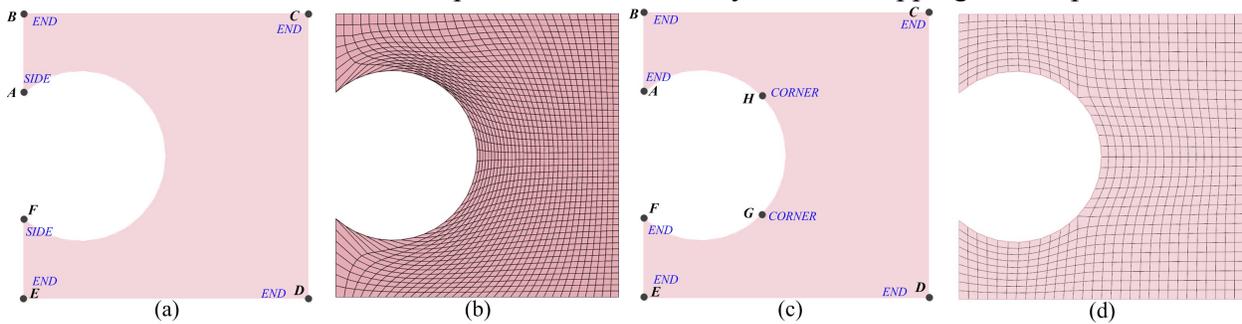


Figure 9. Structured grid generation of a surface with a three quarters of circle feature by submapping through templates: (a)corner assignment by CUBIT 13.2; (b)structured grid for (a); (c)corner assignment based on templates in this paper; (d)structured grid for (c)

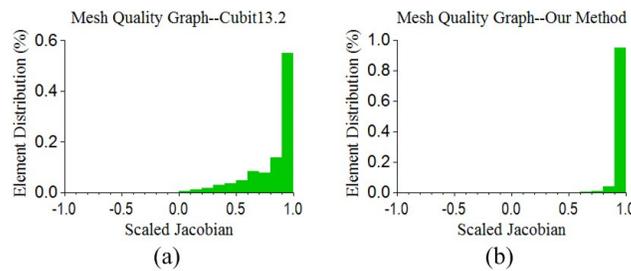


Figure 10. Mesh quality histogram: (a)mesh quality histogram for Figure 1.7(b); (b)mesh quality histogram for Figure 1.7(d)

The first example in Figure 9 shows a surface with a three-quarters round feature, which makes it difficult to generate a valid corner assignment for submapping. CUBIT 13.2 assigns corners for surface vertices as Figure 9(a) and the resulting structured quadrilateral mesh is shown in Figure 9(b). From Figure 9(b) and Figure 10(a), poor mesh quality is produced by submapping due to the lack of an optimal corner assignment. By applying templates described in this paper, two new vertices are virtually inserted and surface vertices are classified as Figure 1.7(c). The corresponding structured quadrilateral mesh is shown in Figure 9(d) with good mesh quality, which is illustrated in Figure 10(b).

6.2 Many-to-many Sweeper

The **1-1** sweeping is a special case of multi-sweeping which includes **M-1** and **M-N** sweeping. Unlike **1-1** sweeping where the source and target surface has the same topology and they are directly connected by the linking surfaces, **M-N** sweeping is characterized by M sources and N targets, which generally have different topologies. The key problem during multi-sweeping is to resolve curves on the source and target surfaces, combined with the resulting source and target surface patches needing to match: more specifically, multi-sweeping requires to solve which source surface or area of a specific source surface will be swept onto a specific target surface or a specific area of a target surface. This is due to the inherent characteristics that each mesh face element on the source surfaces should match only one mesh face element on the target surfaces. This subsequently results in that the interval assignment problem for edges on the source and target surfaces should be solved.

6.2.1 Edge Patch Imprinting

A new edge patch imprinting algorithm for multi-sweeping problems has been developed in this work: inputs are the source surfaces, target surfaces and linking surfaces with their parametric spaces; outputs are a list of source and target surface patches with each s surface patch matching an exactly t surface patch. After imprinting, curves and vertices on the target surfaces with source surfaces are resolved by embedding them on the source surface meshes and vice versa.

6.2.2 Interior Edge Patch Placement

Once the bounding edge patches have already propagated to the next sweeping level along the linking surfaces, their enclosed interior edge patches must be propagated to next sweeping level so that curves on the s/t surfaces can be resolved. As matter of fact, those edges on the source and target surfaces must be reasonably placed and imprinted on surfaces so that there are no distortions or degenerated elements when sweeping the surface meshes through volumes. Detailed algorithm and theory can be found in [8]

6.3 Integration of Sweeping Pieces

Integration of Jaal quadmesher with details of our multi-sweeping algorithm where there are five main steps for swept volume meshes will be incorporated into MeshKit. The sweeping method in this work starts with an input volume without any mesh but with the identified source and target surfaces. Integration of the final version multi-sweeper with Jaal and the rest of MeshKit is currently in-progress.

6.4 Examples

The numerical simulations such as heat transfer (predictions of peak subassembly temperature) and *CFD* are needed to assess a designed reactor. One of the typical example is the grid spacer reactor model shown in Figure 11(a) and Figure 11(b). Generally, the swept volume meshes are preferred for the flow simulation.

The only other feasible method for generating the swept volume meshes for the grid spacer model is to partition the geometry until all the decomposed pieces are *I-I* sweepable. However, there are some shortcomings for the volume decomposition method: generally it is difficult to generate the internal surfaces which are used to guide the volume decomposition; meanwhile, interior nodes are placed separately (logically or physically) in each decomposed subvolumes and poor mesh quality may be produced inside volume. Those disadvantages become very obvious when there is a model with complicated internal structures such as the grid spacer model in Figure 11(b) where the blind twisted blade-like holes are located inside the geometry. A closer look can be taken at Figure 11(c) and Figure 11(d). Therefore, it requires a lot of experiences and takes a lot of efforts to decompose the grid spacer model in a reasonable way so that the acceptable mesh quality of the swept volume meshes can be achieved.

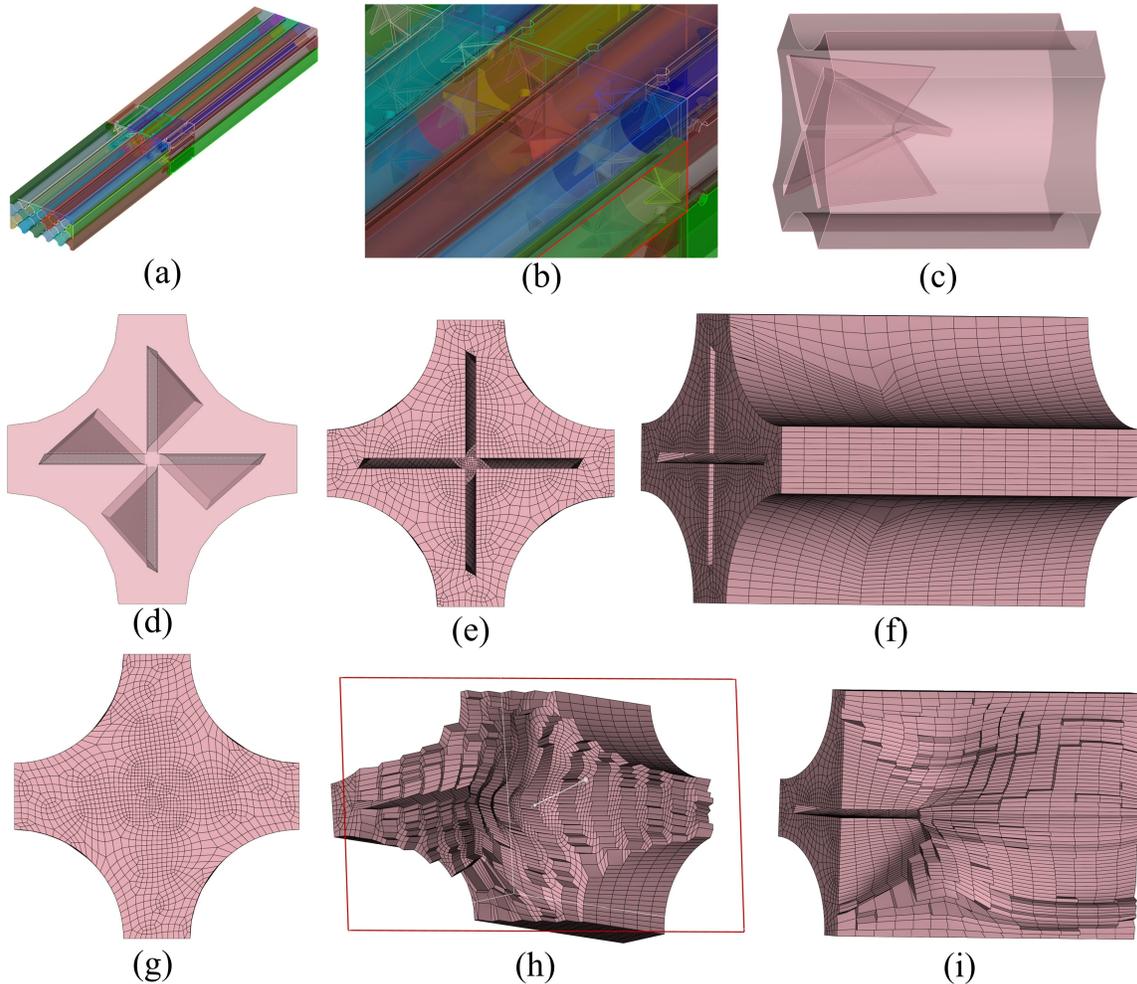


Figure 11. All-hexahedral mesh generation for grid spacer reactor model: (a) assembly geometry model; (b) local zoom-in view of assembly model; (c) single part of grid spacer reactor model; (d) top view for (c); (e) source surface meshes for (c); (f) all-hex meshes for (c); (g) target surface meshes for (c); (h) cut-view of interior node distribution inside the grid spacer model; (i) another cut-view of interior node distribution

The grid spacer model shown in Figure 11(c) is a M-1 sweeping problem where the most difficult part is to locate interior nodes inside the swept volume meshes constrained the bounding surfaces. Instead of using the volume decomposition, our approach uses the edge patch imprinting and generate the target surface meshes as Figure 11(g) which are mapped from the source surface meshes in Figure 11(e). The interior nodes are placed by mapping those interior nodes from the idealized model to the physical model where interior nodes in the idealized model are simply placed based on affine transformation. The resulting interior node distribution is shown in Figure 11(h) and Figure 11(i).

7 Conclusions

MeshKit v1.3 was released with new algorithms for uniform mesh refinement with CoreGen. Numerous enhancements and fixes to RGG algorithms and graph-based design were made to MeshKit. Significant effort was directed towards maintain and upgrading the

CGM version. Development of AssyMesher, which requires an open-source triangle meshing algorithm in progress. At present, a prototype of AssyMesher is available that work with CAMAL (closed source) triangle meshing library.

Collaboration with Kitware continues as we complement each other on development of GUI and algorithms in MeshKit and SIGMA tools. Successful development tie-up has led to development as other useful GUI products such as CMB for CGM and CMB for general algorithms in MeshKit. The capability to load MOAB meshes into Paraview tool by Kitware was also enhanced, along with CMake-based build for all SIGMA libraries.

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. RGG 2.0 release announcement: <http://www.mcs.anl.gov/articles/release-rgg-20-mesh-toolkit-announced>
2. 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.
3. 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).
4. Jain, Rajeev and Tautges, T. J. (2014). NEAMS MeshKit: Nuclear Reactor Mesh Generation Solutions, Charlotte, North Carolina, Apr 2014.
5. CAMAL - The CUBIT Adaptive Meshing Algorithm Library, Sandia National Laboratories, Albuquerque.
6. Jain, Rajeev, and Tautges, T. J. (2014). Generating unstructured nuclear reactor core meshes in parallel. In *Proceedings of the 23rd International Meshing Roundtable* (pp. 351-363). <http://dx.doi.org/10.1016/j.proeng.2014.10.396>
7. GRUMMP: <http://tetra.mech.ubc.ca/GRUMMP/>
8. Cai, S. (2015). Algorithmic Improvements to Sweeping and Multi-Sweeping Volume Mesh Generation. *University of Wisconsin, Madison*.



Mathematics and Computer Science Division

Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov



Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC