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.

Meshkit

prepared by

Rajeev Jain, Timothy J. Tautges
Mathematics and Computer Science Division, Argonne National Laboratory

September 30, 2013

SUMMARY

MeshKit is an open-source library for mesh generation and related algorithms. MeshKit is designed both for users that perform mesh generation and for developers that develop meshing algorithm/application(s). MeshKit includes new meshing algorithms implemented first in this library, as well as interfaces to meshing algorithms/packages developed in DOE Labs and elsewhere. MeshKit uses a directed graph-based approach for organizing meshing problems that supports both traditional BREP-driven meshing as well as more general meshing processes and tools. One such tool is the Reactor Geometry (and mesh) Generator (RGG); RGG generates hexagonal and rectangular reactor geometries and meshes from text-based input in serial and parallel. It has been used to support a variety of reactor simulation codes and reactor types. Although primarily a library, MeshKit is also supporting a collaboration with Kitware to develop a graphical/GUI interface to the library.

TABLE OF CONTENTS

Summary	i
Table of Contents	ii
List of Figures	iv
List of Tables.....	iv
1 Introduction.....	5
2 Dependencies and Optional External Components	5
2.1 Mandatory Components	6
2.2 Optional Components.....	6
3 Design	7
3.1 Graph-Based Organization of Meshing Problem	7
3.2 MeshKit Code Design	9
3.3 Documentation	11
3.4 Python Scripting.....	11
4 Algorithms	11
4.1 Structured Meshing	11
4.1.1 Structured Block Mesher: SCDMesh	11
4.1.2 Embedded Boundary Mesher: EBMesh	12
4.2 Unstructured Meshing.....	13
4.2.1 Interval Matching Algorithm: IntervalAssignment	13
4.2.2 Simple Vertex Mesher: VertexMesher	14
4.2.3 Edge Mesher: EdgeMesher.....	14
4.2.4 Sweeping Algorithm: OneToOneSweep	14
4.2.5 Mapping Algorithm: TFIMapping.....	14
4.2.6 2D Quad Mesher: QuadMesher	15
4.2.7 Extrusion Algorithm: ExtrudeMesh	15
4.2.8 Parallel Mesh Generation: ParallelMesher	16
4.2.9 External: CAMAL Library: CAMALPaver, CAMALTriAdvance, CAMALTetMesher	17
4.2.10 External: Triangle Library: TriangleMesher.....	17
4.2.11 External: NetGen Library: NGTetMesher	17
4.3 Mesh Modification	18
4.3.1 Mesh Decimation Algorithm: QslimMesher	18
4.3.2 Copy Algorithms: CopyGeom, CopyMesh	18
4.3.3 Boundary Layer Mesher: PostBL	18
4.3.4 Mesh-Based Geometry Modifications: MBGeomOp, MBSplitOp, MBVolOp	19
4.3.5 Mesh Merging Algorithm: MergeMesh.....	20
4.3.6 Sealing Facet Based Geometry: MakeWaterTight	20
4.3.7 External: Mesh Optimization Algorithm: MesquiteOpt.....	20
4.4 Application Specific.....	20

4.4.1 Nuclear Reactor Modeling: Assembly Geometry Creation Tool: AssyGen.....	20
4.4.2 Nuclear Reactor Modeling: Core Mesh and Geometry Creation Tool: CoreGen	21
4.4.3 Climate Application: Ice Sheet Mesh Generation	22
5 Conclusions.....	23
6 Acknowledgements.....	24
7 References.....	24
Appendix A. Configure/Build/Installation of MeshKit	25

LIST OF FIGURES

Figure 1. The graph for a 2-step tetmesh-then-refine operation, along with the root and leaf nodes of the graph.	8
Figure 2. Digraph-based mesh generation example: sweep-meshing a cylinder.	9
Figure 3. CAD geometry and the structured mesh generated by SCDMesh algorithm.	12
Figure 4. Input and output from EBMesh tool. (a) STL 3D “statue” and “horse” models. (b) Cartesian mesh elements of 3D STL models.	13
Figure 5. (a) Input one dimension line, (b) Output mesh.	14
Figure 6. (a) 2D Rectangle geometry, (b) A meshed 2D rectangle.	14
Figure 7. Results from quad mesher algorithm.	15
Figure 8. Generation of an extruded mesh by copy/moving a single unit cell into four, then extruding axially.	16
Figure 9. Results from parallel tetrahedral mesh generation of a VHTR geometry (a) Geometry file indicating partitions. (b) Comparison of speed-up obtained by using different partitioning methods.	17
Figure 10. Trivial example showing input and output mesh from QslimMesher.	18
Figure 11. User specified digraph for creating reactor assembly mesh with boundary layers.	18
Figure 12. 19 assembly reactor core mesh: (a) Original mesh. (b) Close-up of original mesh showing fluid and gap regions. (c) Close-up of original mesh showing boundary layers on fluid and gap regions.	19
Figure 13. First two stages of the geometry/mesh process, where AssyGen and CUBIT/MeshKit are executed for each assembly type.	21
Figure 14. (a) Simple example demonstrating CoreGen input and output files. (b) (A) Copy/move task distribution among processors for the same problem in Fig. 14(a) and (B) final core-mesh with numbered assemblies are shown.	21
Figure 15. Homogenized EBR-II core with XX09 assembly. Inlet, outlet, isometric and sectioned views are zoomed to shown the fuel and other instrumentation pins in the model.	22
Figure 16. Stages in creation of Ice sheets mesh model using satellite data. (a) Region of interest. (b) BREP-based geometry model. (c) Final hexahedral mesh showing elevation data. (d) Some simulation results.	23

LIST OF TABLES

Table 1. Simple C++ code showing mesh generation process in MeshKit.	10
---	----

1 Introduction

Mesh generation is a challenging field of science that has made rapid advancement over the years. New discretization techniques, improvements in computer hardware and robust solver methods have driven the research in this field. While several open source and commercial mesh generation tools were already available when this project was started, each of these tools were deficient in some way, either being too restrictive in their licensing (e.g. GPL), focused on only one specific meshing technology (triangles, tetrahedral), or lacking support for other parts of the meshing process (e.g. geometric model query). MeshKit is meant to overcome these problems, as well as to serve as a delivery vehicle for specific meshing tools and algorithms being developed for various program sponsors.

The MeshKit design philosophy is two-fold. First, it provides a collection of meshing algorithms and other tools commonly needed for mesh generation (coordination of BREP-based meshing process, mesh smoothing, etc.). The tools and algorithms in this release are sufficient for basic generation of both triangle/tetrahedron and quadrilateral/hexahedron meshes on CAD-based and discrete model geometry. Second, MeshKit can also serve as a platform for advanced meshing algorithm research, providing infrastructure and lower-dimensional mesh generation algorithms that would otherwise have to be developed before researching advanced algorithms. MeshKit also supports the connection of external meshing algorithms to the rest of the library, enabling the use of proprietary or experimental algorithms.

MeshKit is implemented in C++, and provides a traditional C++-based API for interactions with other codes. A Python interface is also provided, for interactive access to the library. Various options are available for graphical visualization of meshing results. The MOAB library implements a ParaView plugin that can be used to import MOAB models directly into ParaView. The VisIt visualization tool can also be configured and built with MOAB to provide a similar import capability. MOAB can also export mesh and associated data in various formats, including Vtk. Finally, a collaboration is underway with KitWare, Inc, to build a graphical/GUI tool for interacting with MeshKit; this will support an interactive, GUI-driven mesh generation process, similar to that provided with CUBIT [1] and various other proprietary or commercial meshing tools.

The report is structured as follows. In Section 2 we highlight all required and optional external packages that MeshKit relies on for various functionalities. Section 3 describes the design philosophy and overall organization of the library. Summary of current algorithms and results are given in Section 4. The report is concluded in Section 5 with an outline of future work.

2 Dependencies and Optional External Components

MeshKit is implemented as a library to allow its use in both interactive and non-interactive applications. Similarly, it uses separate libraries for representation and query of both geometry and mesh data, to allow external applications to import and query those data without having to depend on MeshKit and all the libraries it depends on. Although this

complicates slightly the configuration and build process for MeshKit, it also increases flexibility and versatility. Libraries used by MeshKit are either mandatory or required, depending on function. Both types are described next. The locations of both mandatory and optional dependencies for MeshKit are specified at configuration time (see Appendix A for configure/build instructions for MeshKit).

2.1 Mandatory Components

MeshKit relies on several libraries for representation and evaluation of geometry and mesh data. The CGM library [2] is used for query and modification of CAD-based geometry. CGM provides a consistent topological model and interface to geometry in a variety of solid model formats, including ACIS [3] and Open.CASCADE [4]; the Open.CASCADE interface is significant because this provides a fully open-source solution for CAD model interactions. CGM also provides “virtual geometry” functionality, which can be used to suppress details in the topological model without corresponding modifications to the CAD geometry. MeshKit uses the MOAB library for its mesh representation [5]. MOAB represents structured, unstructured, and polygonal/polyhedral meshes in an array-based format that is both memory and time efficient. Although not used in MeshKit, a third library, Lasso [6], can be used to recover mesh to geometry relations, for supporting advanced processes like Adaptive Mesh Refinement. In addition to the native APIs in CGM and MOAB, various algorithms in MeshKit use the ITAPS geometry (iGeom) and mesh (iMesh) interfaces [7]; these algorithms can be connected directly to any geometry or mesh libraries providing implementations of these interfaces.

2.2 Optional Components

MeshKit includes several Argonne-developed as well as external meshing and related packages whose licenses are compatible with direct inclusion. MeshKit can also be configured to use external mesh generation packages. At the current time, these include:

- **Triangle:** The widely popular Triangle [8] provides a 2D triangle mesh generation algorithm.
- **CAMAL/CUBIT:** Sandia National Lab’s proprietary CAMAL library [9] has been integrated and it provides access to tetrahedral, tri-advance and paving algorithms.
- **NetGen:** An open source automatic tetrahedral mesh generation library [10].
- **Mesquite:** Open source mesh optimization package MESQUITE [11] has also been integrated to work with any of the above meshing algorithms or MeshKit’s native algorithms.
- **IPOPT:** Ipopt [12] an optimization library can also be configured with MeshKit. This library is used by interval assignment algorithm developed in MeshKit.
- **MPI:** Some of the parallel capabilities in MeshKit also make use of the MPI libraries.

3 Design

In this section the design philosophy, code design, documentation and python scripting interface of MeshKit are presented.

3.1 Graph-Based Organization of Meshing Problem

Most current meshing environments are targeted toward a Boundary REPresentation (BREP)-based approach, backed by a geometric model and associated topology (vertices/edges/faces/regions and adjacency relations between them). The meshing process usually proceeds by meshing BREP entities in increasing topological dimension, starting with vertices, then edges, and so on. However, this model is deficient in several ways. First, not every meshing process needs or has a geometric model representing the entire domain to be meshed. The best example of this is the Reactor Geometry (&mesh) Generator (RGG) tool [13], where individual assembly types have geometric models but are then copy/moved into a lattice of assembly models forming a reactor core. Second, a meshing procedure may not involve only a once-through meshing of each BREP entity; again, RGG is a good example of this, where the first part of the process involves meshing BREP models, but the last step involves copy/moving mesh subsets into a larger core lattice. Finally, the procedure-driven approach to meshing represented by most CAD-based meshing tools fails to capture the parallelism and dependency structure that can be found in most meshing problems (including BREP-based ones); representing and exploiting this richer structure provides more flexibility while still being applicable to BREP-based problems.

MeshKit models the general meshing problem as a directed graph-based process, with graph nodes representing individual steps in the process and graph edges representing dependencies between those steps. For convenience, the graph always has a single root and leaf node, with one or more possibly-independent paths between them. Each graph node represents an explicit step in the meshing process, whether that involves generation of new mesh or performing some other operation on existing mesh. The part of the model operated on by that operation is stored as input for that node, along with any control parameters specific to the operation. The meshing process is executed by traversing the graph twice, once in reverse direction (from leaf to root), to perform necessary setup actions and create upstream graph nodes not explicitly created by the application, then in forward direction, to perform the action represented by each graph node.

For example, in the case of tetrahedral meshing then refinement, the graph would have two nodes, one for generating the initial mesh, and the other for refining that mesh. The (single) edge linking the two nodes represents the dependency between these operations. The graph for this simple process is shown in Fig. 1. In many cases, tetrahedral meshing is a fully automatic process, with the user specifying only the meshing scheme (represented by a “tetmesh” graph node), along with a target geometric volume and mesh size (input data on that graph node). The refinement process is represented by a second node, with refinement level specified as input to that node.

Figure 1. The graph for a 2-step tetmesh-then-refine operation, along with the root and leaf nodes of the graph.

From the user point of view, the meshing process in Fig. 1 is quite simple. However, inside the meshing implementation, there are more steps involved than simply generating a volume mesh then refining it, especially for BREP-based meshing. For example, in most cases, tetrahedral meshing of a BREP volume does not happen in one step, but rather uses a previously-generated triangle mesh of its boundary as a starting point. Similarly, generation of the triangle mesh for each BREP face usually starts from a discretization of the edges bounding the face. These operations should be represented by distinct nodes in the graph, since they are performed by separate meshing algorithms and we would like to associate graph nodes with individual meshing operations. On the other hand, these nodes can be constructed automatically (based on input requirements of the volume meshing operation), and forcing the user to understand the full graph may be overwhelming even for this simple process. We resolve this issue by having graph nodes be one of two types, those generated explicitly by the user and others generated automatically to meet requirements of downstream graph nodes. Automatically-generated nodes appear during the “setup” phase of traversal; since they are requirements of the graph nodes generating them, naturally they appear upstream in the graph. Since the graph is being traversed in reverse order when these nodes are created, they will be visited during the same “setup” traversal, and may generate new graph nodes to fulfill their dependency requirements. When setup traversal arrives at the (single) root node, all graph nodes have had their dependency requirements specified, and the “execution” phase of traversal can begin.

An example of a more complex meshing process is shown in Fig. 2. The user-specified part of the problem is a sweeping operation of a single source surface mesh to a single target surface. The user specifies that sweeping should be used to mesh volume V1 (which also results in generation of the mesh on the target surface S2), and that the source surface S1 should be meshed with a “QuadMesh” algorithm. Based on that input, the Sweep algorithm determines that surface S3 must be meshed with a MapMesh algorithm, and these algorithms require the meshing of edges bounding these surfaces. To meet requirements inherent in all quad and hex-based meshing algorithms, an IntervalMatch algorithm must be used to solve global interval constraints on all edges and surfaces bounding the volume. This tool must be executed before meshing of any edges or surfaces, and thus appears first in the graph. Automatically-created graph nodes are drawn in a different color in Fig. 2, to indicate their difference from explicitly-created nodes. Note that there are far more automatically-created nodes in the graph, and that the user need only concern themselves with the explicit nodes.

Figure 2. Digraph-based mesh generation example: sweep-meshing a cylinder.

Using a graph-based approach for the meshing process has several advantages over the more traditional BREP-based method:

1. It is easy to make local changes to the meshing process and to determine exactly which other steps of the meshing procedure need to be re-executed. Namely, if one graph node is changed (usually because input affecting the meshing operation is changed), that node and all descendants in the graph must be re-executed.
2. Conflicting input in the meshing specification appears explicitly as cycles in the (directed) graph, making them easy to detect and avoid.
3. The resulting graphs are much less complex than those representing the BREP, while still capturing the relevant dependencies between meshing operations.
4. Parallelism is represented explicitly in the graph, and can be exploited by executing independent nodes concurrently. (See Section 4.4.2 for an example).

3.2 MeshKit Code Design

MeshKit is implemented in C++, and is best used by applications through that programming language. Broadly, the library is implemented in terms of a library instance class, classes used for the graph and various types of graph nodes, and utilities for other types of data involved in the meshing process. All geometry and mesh data is stored in the CGM and MOAB libraries, respectively, with a corresponding class in MeshKit strictly for convenience and for maintaining geometry-mesh relations.

Four fundamental classes in MeshKit are:

1. **MKCore:** This is the instance class that keeps a reference to geometry, mesh and relational interfaces instances used in MeshKit. This class also is derived

from MKGraph, so each instance of MKCore represents a single mesh graph as well.

2. **ModelEnt:** This is a convenience class for accessing the mesh and (if present) the geometry library representations of this object. There will always be a mesh library representation (in the form of a MOAB Entity Set), since that representation is inexpensive to construct/represent and is needed to store mesh. ModelEnt also keeps a reference to the MKCore instance to which it is associated, and can retrieve references to the geometry and mesh instances through that class.
3. **MeshOp:** This class is derived from MeshScheme, which is derived from GraphNode. Each MeshOp or MeshScheme can be inserted in the MKGraph, and must supply definitions of the setup_this() and execute_this() functions (which define the actions to be done during the setup and execute phases of graph traversals, respectively).
4. **SizingFunction:** This class represents a sizing function that can be used by meshing algorithms. SizingFunction instances are stored in an indexed table in MKCore, allowing them to be shared by more than one MeshOp.

In its simplest form, MeshKit can be used by specifying the geometry, a mesh scheme, and a desired mesh size, then executing the meshing operation. Table 1 shows a simple example, where a geometry file is loaded, then volume entities are meshed with the NetGen tet mesher. Behind the scenes, a tri mesher is added to the graph, then setup is called on that tri mesher; it in turn creates an edge mesher and assigns all geometric edges to it. If not specified directly, the size used to mesh each entity is the first one registered with the MeshKit instance. Once generated, the mesh can be retrieved with various functions on the MeshKit instance, or directly through the MOAB or iMesh interfaces.

Table 1. Simple C++ code showing mesh generation process in MeshKit

```
MeshKit::MKCore mk; // by default, creates geometry, mesh, and relations instances
mk.load_geometry(filename); // load a geometric model into MeshKit
MeshKit::MEntVector model_ents;
mk.get_entities_by_dimension(3, vols); // get all geometric volumes
MeshKit::SizingFunction esize(mk, -1, 0.25); // create a sizing function for all mesh
mk.construct_MeshOp("NGTetMesher", vols); // construct a tet mesher and set graph
mk.setup_and_execute(); // execute the meshing graph
```

For more complicated meshing processes, a directed graph of mesh-based operations can be specified, with each graph node representing an operation on one or more model entities. This can represent a simple topology-driven meshing process, where entities are meshed in order of increasing topological dimension; or it can be a sequence of more general steps, for example a mesh, smooth, refine sequence. The graph can be set up before any meshing is done.

3.3 Documentation

The MeshKit library is documented in a set of files that can be processed with the Doxygen tool. MeshKit documentation is generated nightly from the code repository and made available online at <http://www.mcs.anl.gov/~fathom/meshkit-docs/html/index.html>.

3.4 Python Scripting

MeshKit is intended primarily as a library for mesh generation and associated tools. As such, it does not contain direct support for graphics/GUI interactions nor a command language for driving the tool. However, MeshKit does provide Python interfaces to its tools and classes, which can be used to drive MeshKit interactively and from script files. This interface is constructed partly based on PyTAPS [14], which has very detailed documentation and examples guide online.

4 Algorithms

In a short span of time several algorithms in MeshKit have been developed and published. Overall, algorithms can be classified based on licensing, functionality, element type, dimension etc. Here we classify the algorithms based on their functionality.

4.1 Structured Meshing

MeshKit uses MOAB for mesh representation, which has active development in the field of structured mesh representation in serial and parallel. At present there is no native structure mesh writer, the resulting mesh files can be written in unstructured-ASCII Vtk file format that can be converted to solver specific structured mesh representation using some other tool. Two key algorithms developed in MeshKit for generating structured meshes are described next.

4.1.1 Structured Block Mesher: SCDMesh

This MeshOp generates a simple rectangular structured mesh, sized to completely surround ModelEnt(s) using a geometric entities bounding box. Options for grid size, mesh representation and axis type are defined for providing more control over the mesh generation process. It supports MeshKit's EBMesher and other structured mesh algorithm(s). Mesh generated using SCDMesh algorithm is shown in Fig. 3.

Figure 3. CAD geometry and the structured mesh generated by SCDMesh algorithm.

4.1.2 Embedded Boundary Mesher: EBMesh

EBMesh tool can generate Cartesian meshes for solvers that use embedded boundary algorithms. It uses ray-tracing technique based on hierarchical Oriented Bounding Box (OBBs) in MOAB. Each mesh cell is distinguished as being inside, outside or on the boundary of the input geometry, which is determined by firing rays parallel to x/y/z coordinates. EBMesh tool can directly import CAD-based solid model formats and facet-based formats, output from SCDMesh can be also used as input to EBMesh. Boundary cells created by this tool have edge-cut fraction and volume cut fraction information for each material. Detailed explanation, results and comparison with other related tools could be found in an International Meshing Roundtable paper published in 2010 [15]. Fig. 4 shows input STL format files produced by 3D scanning, which have complex boundary representations and the Cartesian mesh generated by EBMesh tool.

Figure 4. Input and output from EBMesh tool. **(a)** STL 3D “statue” and “horse” models. **(b)** Cartesian mesh elements of 3D STL models.

4.2 Unstructured Meshing

MeshKit has substantial amount of infrastructure and algorithms for generating unstructured meshes. MOAB-based representation of the mesh enables access to mesh in chunks rather than through individual entities. It can represent triangular, quadrilateral, tetrahedral, hexagonal, polyhedral and higher-order elements. MeshKit also benefits from various tools such as tree representation, smoothing, adaptive mesh refinement, parallel mesh communication, and parallel read/write options in MOAB. MeshKit uses CGM for CAD geometry representation that currently supports OCC and ACIS geometry kernels. This enables all MeshKit algorithms to function with any geometry engine supported by CGM. Some robust and widely used external unstructured meshing algorithms such as Triangle and NetGen have been interfaced with MeshKit. Brief description and results of algorithms currently available in MeshKit are presented.

4.2.1 Interval Matching Algorithm: *IntervalAssignment*

Interval assignment is the problem of assigning an integer number of mesh edges to each curve so that the assigned value is close to the goal value, and all containing surfaces and volumes may be meshed independently and compatibly. It is one of the key algorithms for meshing complicated geometries and preventing meshing algorithms to fail from invalid sizing specified by the user. Interval assignment algorithm in MeshKit uses a new optimization function and approach (NLIA), it is found to be faster and more efficient compared to other interval matching algorithms. Integration of this algorithm with other MeshKit algorithms is work in progress. Implementation details and findings of interval matching algorithm can be found in this year’s proceedings of International Meshing Roundtable [16].

4.2.2 Simple Vertex Mesher: VertexMesher

This is the basic algorithm that meshes a vertex. Input ModelEnt(s) to this algorithm must be geometric vertices. VertexMesher mesh operation is inserted into the meshing graph during the setup phase of most edge meshers.

4.2.3 Edge Mesher: EdgeMesher

EdgeMesher generates a mesh for edges, creating the nodes and line segments on edges. There are four schemes for edge mesher: equal, bias, dual-bias and curvature. It internally calls VertexMesher. Fig. 5 shows the input and output from an example documented in MeshKit code repository.

Figure 5. (a) Input one dimension line, (b) Output mesh.

4.2.4 Sweeping Algorithm: OneToOneSweep

OneToOneSweep algorithm generates an all hexahedral mesh by sweeping the source mesh to the target surface. It uses a harmonic function to mesh the target surface with good quality, avoiding expensive smoothing operations. The interior nodes between the source and target surface are generated using cage-based deformation method. Implementation details and results of sweeping algorithm can be found in this year's proceedings of International Meshing Roundtable [17].

4.2.5 Mapping Algorithm: TFIMapping

This MeshKit algorithm generates an all-quad mesh by transfinite interpolation with i, j parameters. Fig. 6 shows the input geometry and output mesh from an example documented in MeshKit code repository.

Figure 6. (a) 2D Rectangle geometry, (b) A meshed 2D rectangle.

4.2.6 2D Quad Mesher: QuadMesher

Quad mesher is a MeshKit native algorithm that produces a high quality, isotropic all-quadrilateral mesh for an arbitrary complex surface geometry. Two basic steps used in this algorithm are triangle to quad mesh conversion and global mesh cleanup operation. Specific details and results are published [18]. Fig. 7 highlights some of the results produced by quad mesher.

Figure 7. Results from quad mesher algorithm.

4.2.7 Extrusion Algorithm: ExtrudeMesh

ExtrudeMesh is a simple extrusion algorithm that reads in an already meshed 1D or 2D ModelEnt(s) and creates a 2D or 3D mesh respectively. It also allows for extrusion to be specified along a rotation path. In-order to produce a mesh that is fit for simulation, material and boundary conditions from the initial mesh are propagated to the final mesh by specifying grouping sets. CopyMesh algorithm (see Section 4.3.2) also uses the same sets functionality. MeshKit provides three different types of abstractions for handling groupings based on the sets they apply to. These three types of sets are:

1. **Copy sets:** These sets get duplicated and populated with copies of entities in the original copy set.
2. **Expand sets:** These sets are expanded, that is copies of entities are put into same sets containing the entities being copied.

3. **Extrude sets:** These are sets whose contents are replaced with extruded version of original entities.

Consider a simple unit cell, consisting of an inner circular surface, representing a fuel pin, and an outer material, representing a coolant region (see Fig. 8). Both surfaces are meshed, as are the bounding model edges and vertices. On this model, we have identified one group of elements in set “F”, corresponding to fuel, and another group as coolant set “C”. We have also identified a set of mesh faces as volume “V1”. Sets F and C are designated as “Expand” sets, that is, any copies of mesh faces in these sets will be added to those sets; in other words, the sets will expand to include copies of the contained faces. The set V1 is identified as a “Copy” set; any copies of the mesh faces in that set will be put into a copied set. Next, the pair of surfaces is copy/moved to get four pairs of surfaces, arranged in a square 2x2 grid. The F and C sets expand to include the corresponding face copies, while the V1 set is copied three times. Next, the sets of mesh edges bounding the collection are added to a set designated as “side”, and this set is identified as an “Extrude” set. In addition, the F, C, and V1-V4 sets are also identified as extrude sets. In the final operation, the eight surfaces are extruded into the 3rd dimension. In each extrude set, each entity is replaced with the next-higher-dimension entity or entities produced by the extraction. So, sets F, C, and V1-V4 receive sets of hexahedra resulting from the extrusion of the quadrilaterals in the respective sets, while the “Side” set receives quadrilaterals, extruded from the edges originally in the set.

Figure 8. Generation of an extruded mesh by copy/moving a single unit cell into four, then extruding axially.

4.2.8 Parallel Mesh Generation: ParallelMesher

ParallelMesher MeshOp in MeshKit uses a CAD-based approach, wherein solid model geometries are distributed using parallel geometry loading feature provided by CGM. MOAB provides parallel mesh representation and communication routines to handle the mesh. ParallelMesher is not limited to a specific meshing algorithm, it can use any tri/tet/quad/hex algorithm available in MeshKit. ParallelMesher meshing procedure is similar to traditional BREP-based meshing with additional communication between the processor after meshing

shared vertex/edge and surfaces. Final mesh from this MeshOp can be saved in parallel using MOAB's parallel writing capability. Fig. 9 shows a hexagonal VHTR reactor geometry generated using AssyGen (see Section 4.4.1). It takes 118 seconds to mesh this geometry with 1.74M tetrahedral elements in serial. In parallel, using 12 processors the same model can be meshed in 14 seconds, which is a speedup of 8.65 and efficiency of 72%. The model for this test uses ACIS geometry and CGM's read-and-delete strategy for distributing the geometry in parallel.

Figure 9. Results from parallel tetrahedral mesh generation of a VHTR geometry **(a)** Geometry file indicating partitions. **(b)** Comparison of speed-up obtained by using different partitioning methods.

4.2.9 External: CAMAL Library: CAMALPaver, CAMALTriAdvance, CAMALTetMesher

CAMAL is a proprietary mesh library developed at Sandia National Library. Three mesh operations namely: CAMALPaver, CAMALTriAdvance and CAMALTetMesher are integrated into MeshKit for generating quadrilateral, triangular and tetrahedral meshes respectively. It must be noted that all these operation internally create an EdgeMesher and VertexMesher graph node.

4.2.10 External: Triangle Library: TriangleMesher

The Triangle library is widely used for generating exact Delaunay triangulation, constrained Delaunay Triangulation, conforming Delaunay triangulations, Voronoi diagrams and high-quality triangular meshes. It has a restrictive license for commercial use. MeshKit provides an interface for this library for generating 2D triangular meshes.

4.2.11 External: NetGen Library: NGTetMesher

NetGen is an LGPL licensed automatic tetrahedral mesh generation library. MeshKit has an interface to this library for generating tetrahedral meshes. Multiple interfaces for generating meshes for same element type provide more flexibility and aid comparison of mesh quality among algorithms.

4.3 Mesh Modification

Copy/move/merge, mesh decimation, boundary layer mesh generation, mesh quality optimization and facet-based algorithms available in MeshKit are described in this section.

4.3.1 Mesh Decimation Algorithm: QslimMesher

Source code of qslim mesh decimation library has been integrated in MeshKit. This algorithm uses edge collapse as a primary simplification method. A cost for each possible edge collapse is established using quadric-based error concept. See Section 4.4.3 for usage of this algorithm for climate applications. Fig. 9. shows a trivial example, where 10 triangles from input 2D mesh are decimated to 8 triangles.

Figure 10. Trivial example showing input and output mesh from QslimMesher.

4.3.2 Copy Algorithms: CopyGeom, CopyMesh

CopyGeom and CopyMesh algorithm copy an input geometry or mesh to a specified location specified by the user. Similar to ExtrudeMesh, CopyMesh algorithm also accepts specification of copy, expand and extrude sets. See Section 4.2.8 for an example of CopyMesh with sets specification. Both CopyMesh and CopyGeom algorithms are used by the reactor application MeshOp called CoreGen (see Section 4.4.2)

4.3.3 Boundary Layer Mesher: PostBL

PostBL MeshOp generates boundary layer meshes for an already existing mesh model. Implementation details and results can be found in this year's proceedings of International Meshing Roundtable [19]. Fig 11 shows a users-specified digraph that generates a reactor assembly with a boundary layer mesh from scratch. The AssyGen (see Section 4.4.1) operation generates geometry from a text-based input file describing a reactor assembly. This geometry is input to the QuadMesher (see Section 4.2.6), which feeds into the ExtrudeMesh (see Section 4.2.7) operation to generate a 3D mesh. Then PostBL, based on user-specified boundary layer thickness, bias etc. generates the desired boundary layer elements in the model.

Figure 11. User specified digraph for creating reactor assembly mesh with boundary layers.

PostBL can handle triangular, quadrilateral, tetrahedral and hexahedral meshes. Fig. 12 shows a 19-assembly reactor core that needed meshes along the assembly wall and fuel pin boundary for better fluid flow simulations.

Figure 12. 19 assembly reactor core mesh: **(a)** Original mesh. **(b)** Close-up of original mesh showing fluid and gap regions. **(c)** Close-up of original mesh showing boundary layers on fluid and gap regions.

4.3.4 Mesh-Based Geometry Modifications: *MBGeomOp*, *MBSplitOp*, *MBVolOp*

FBiGeom is an ITAPS-iGeom extension for faceted-based geometry. Several of the iGeom methods are implemented in MOAB. The topology of the model is represented using specific tags and sets in MOAB such as GEOM_DIMENSION, GEOM_SENSE and parent child relationships. MBGeomOp, MBSplitOp, MBVolOp use FBiGeom implementation in MOAB.

MBGeomOp reads in a manifold surface mesh and produces a geometrized surface by populating the database with topology tags and sets; this algorithm first computes the boundary loops and creates the geometry face, curves and vertices. MBSplitOp can be used for editing, cropping and splitting surfaces created by MBGeomOp. MBVolOp creates a B-

REP volume of interest using these mesh-based surfaces generated with MBSplitOp. Section 4.4.3 shows the usage of MBGeomOp, MBSplitOp and MBVolOp.

4.3.5 Mesh Merging Algorithm: MergeMesh

This algorithm reads in a mesh with unmerged vertices and merges them to form a conformal mesh suitable for analysis. This algorithm is used in CoreGen MeshOp after copy/move operations are executed.

4.3.6 Sealing Facet Based Geometry: MakeWaterTight

MakeWaterTight tool removes gaps, overlaps and discontinuous topology between surfaces. It uses MOAB to read facet based models produced by a solid modeling engine in CGM. It is assumed that the feature size is greater than the facet tolerance, and the facet tolerance is greater than the merge tolerance. Faceted surfaces are skinned to resolve their boundary. Bounding edges of each faceted surface are assembled into loops. Loops are cut into arcs that correspond to faceted curves, using geometric vertices. Each arc is then sealed to its corresponding curve by using node-node and node-edge contraction. The result is a watertight model in which adjacent surfaces share the same faceted edges. Implementation details can be found in an International Meshing Roundtable publication [20].

4.3.7 External: Mesh Optimization Algorithm: MesquiteOpt

Mesquite is an open-source mesh quality optimization package developed at Sandia National Laboratory. It can be used as a standalone operation or it can be coupled with other MeshKit algorithms for mesh quality optimization at the end of mesh generation operation.

4.4 Application Specific

RGG is the nuclear reactor application in MeshKit. Two specific mesh operations implemented for this application are AssyGen and CoreGen. Reactor core model generation is broken into three steps: assembly geometry creation, meshing and core mesh creation. Implementation details, examples and enhancements are given in earlier reports [21] [22]. Climate application for generating mesh models of ice sheets is composed of a several unstructured meshing and mesh-based geometry algorithms in MeshKit.

4.4.1 Nuclear Reactor Modeling: Assembly Geometry Creation Tool: AssyGen

AssyGen is the first step of the three-step core mesh creation process, it reads an input file describing a reactor assembly lattice and generates an ACIS or OCC –based geometry file. The second step is meshing, after the first step, user may choose to perform meshing using the CUBIT mesh script generated by AssyGen or using meshing algorithms in MeshKit. The first two steps for two different assembly types are shown in Fig. 13. AssyGen and meshing steps must be performed for each assembly separately. Other results and options provided by AssyGen tool were published in a paper at the International Congress on Advances in Nuclear Power Plants [23].

Figure 13. First two stages of the geometry/mesh process, where AssyGen and CUBIT/MeshKit are executed for each assembly type.

4.4.2 Nuclear Reactor Modeling: Core Mesh and Geometry Creation Tool: CoreGen

CoreGen tool reads an input file describing the reactor core arrangement and generates the reactor core mesh or geometry from its component assemblies. CoreGen uses CopyMesh, ExtrudeMesh, CopyGeom and MergeMesh algorithms in MeshKit. Fig. 14(a) shows the two assembly meshes and an interstice mesh file that form a 19-assembly reactor core. A makefile is generated by CoreGen to automate this process. Fig. 14(b) shows four processors P0-P3 and assemblies numbered in the core, each processor is assigned the task of copy/move(ing) a specific assembly it loads to a specific core location. Speedup results and details of parallel version of CoreGen can be found in [23]. CoreGen has been used to create large meshes such as MONJU, VHTR, HTGR, EBRII, and PWR for different reactor simulation codes.

Figure 14. (a) Simple example demonstrating CoreGen input and output files. **(b)** (A) Copy/move task distribution among processors for the same problem in Fig. 14(a) and (B) final core-mesh with numbered assemblies are shown.

As a part of the collaboration with Kitware to develop a graphical/GUI interface to MeshKit, initial version of GUI for generating RGG models has been developed. Recent mesh generation work involved generating models for complicated assemblies such as XX09. The XX09 assembly is used in Shutdown Heat Removal Tests (SHRT) that demonstrated passive safety features of the EBR-II Experimental Breeder Reactor. A homogenized EBRII core mesh is shown in Fig. 15. This core model contains 7.9M hex8 elements for neutronics mesh and 4.4M hex27 elements for thermo-hydraulics mesh. The geometry for XX09 and other homogenized assemblies are generated using AssyGen, then meshing is performed using CUBIT and finally CoreGen creates the resulting EBRII core mesh.

Figure 15. Homogenized EBRII core with XX09 assembly. Inlet, outlet, isometric and sectioned views are zoomed to shown the fuel and other instrumentation pins in the model.

4.4.3 Climate Application: Ice Sheet Mesh Generation

This project involved creation of hexahedral ice sheet models. Data for creating top surface of ice is obtained from satellite data and bed surface from the ground-penetrating radar data. Sets of points with elevation data are triangulated to obtain the initial approximation of the surfaces. Triangulation is done using Triangle algorithm (see Section 4.2.10). The bed and top surfaces are geometrized using MBGeomOp (see Section 4.3.4), then mesh decimation is done using QSlim (see Section 4.3.1). MBSplitOp and MBVolOp are used to create the region of interest, which is prescribed by using a polygonal line and the direction of splitting. The final geometry is a BREP model of the region of interest and can be meshed using EdgeMesher, CAMALPaver and OneToOneSweep algorithms all described in Section 4.2.

Fig. 16 shows a part of the process for generation of hexahedral mesh for ice sheet modeling. Fig. 16(a) indicates the region of interest with black contour lines on the simplified terrain model. This simplified model is obtained after triangulation of the initial satellite data, geometrization using MBGeomOP and decimation using QSLim. Fig. 16(b) is the BREP model of the volume of interest generated by MBSplitOp and MBVolOp. Fig. 16(c) highlights the elevation data on a hexahedral mesh generated using MeshKit algorithms. Fig. 16(d) shows simulation results of ice flow velocity through the terrain.

Figure 16. Stages in creation of Ice sheets mesh model using satellite data. **(a)** Region of interest. **(b)** BREP-based geometry model. **(c)** Final hexahedral mesh showing elevation data. **(d)** Some simulation results.

5 Conclusions

This report describes the MeshKit v1.0 release, including the MeshKit design, the algorithms available in this version, and installation and use of the library. MeshKit uses a graph-based process for specifying the overall meshing approach, with graph nodes representing meshing and other operations, and graph edges as dependencies between those operations. This approach supports the traditional BREP-driven meshing process found in

most other meshing tools, while also enabling a wider variety of meshing processes not strictly based on BREP models.

This release of MeshKit contains meshing algorithms for both tri/tet and quad/hex mesh generation. In some cases these algorithms are part of 3rd party meshing tools wrapped by MeshKit, while in other cases the algorithms are implemented directly in MeshKit. MeshKit also the Reactor Geometry (& mesh) Generation (RGG) tool; the graph-based design enables RGG to interoperate with other meshing algorithms and allow setup of complete reactor mesh generation problem from creation of assembly geometry to core mesh creation in the same program. Due to its unique and modular approach of development, we believe that the development of this library will benefit other user and developer communities outside the nuclear reactor modeling community.

Over the last one-year new algorithms for post-mesh boundary layer generation, swept mesh generation and interval assignment have been developed in MeshKit. Current work involves integration of interval assignment with other algorithms in MeshKit, adding more examples to the doxygen-based documentation and formalization of CoreGen/AssyGen process with GUI development work performed by Kitware. Several new developments in the area of mesh refinement, automatic mesh scheme selection, support for higher order elements and integrating a mesh quality evaluation library are planned.

6 Acknowledgements

We thank the Fathom group at Argonne, who maintains the libraries required by MeshKit. This work was supported in part by the U.S. Department of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program; by the U.S. Department of Energy Scientific Computing Research, Office of Science; and by the U.S. Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

7 References

1. Sjaardema, G.D, Tautges, T. J, Wilson, T. J, Owen, S. J, Blacker, T. D, Bohnhoff, W. J, Edwards, T. L, Hipp, J. R, Lober, R. R, and Mitchell, S. A (1994). CUBIT mesh generation environment, users manual, vol 1. Sandia National Laboratories, Albuquerque.
2. Tautges, T. J (2005) CGM: a geometry interface for mesh generation, analysis and other applications. *Eng Comput* 17:486–490.
3. Spatial website (2010) <http://www.spatial.com/>
4. Open CASCADE technology website (2000–2010) <http://www.opencascade.org>.
5. Tautges, T. J, Meyers, R, Merkley, K, Stimpson, C, Ernst, C (2004). MOAB: A mesh-oriented database, SAND2004-1592. Sandia National Laboratories, Albuquerque.
6. Lasso: <http://trac.mcs.anl.gov/projects/ITAPS/wiki/Lasso>.
7. Ollivier-Gooch, C., Diachin, L., Shephard, M. S., Tautges, T., Kraftcheck, J., Leung, V., and Miller, M. (2010). An interoperable, data-structure-neutral component for mesh query and manipulation. *ACM Transactions on Mathematical Software (TOMS)*, 37(3), 29.

8. Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quaility Mesh Generator and Delaunay Triangulator. *Applied Computational Geometry Towards Geometric Engineering*. (pp. 203-222)
9. CAMAL - The CUBIT Adaptive Meshing Algorithm Library, Sandia National Laboratories, Albuquerque.
10. Schöberl, J. (1997). NETGEN An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and visualization in science*, 1(1), 41-52.
11. Knupp, P. (2006, September). Mesh quality improvement for SciDAC applications. In *Journal of Physics: Conference Series*, Vol. 46, No. 1.
12. Wächter, A., and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25-57.
13. 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.
14. PyTAPS, website, <https://pypi.python.org/pypi/PyTAPS/>.
15. Kim, H., and Tautges, T. J. (2010). EBMesh: An Embedded Boundary Layer Meshing Tool. In *Proceedings of the 19th International Meshing Roundtable* (pp. 227-242).
16. Mitchell, S. (2014). Simple and Fast Interval Assignment Using Nonlinear and Piecewise Linear. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 203-221).
17. Cai, S., and Tautges, T. (2014). Robust One-to-One Sweeping with Harmonic ST Mapping and Cages: Post-mesh Boundary Layer Generation Tool. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 1-18).
18. Verma, C. S., and Tautges, T. (2012). Jaal: Engineering a high quality all-quadrilateral mesh generator. In *Proceedings of the 20th International Meshing Roundtable* (pp. 511-530).
19. 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).
20. Smith, B., Wilson, P. and Tautges, T. J. (2010). Sealing Faceted Surfaces to Achieve Watertight CAD Models. In *Proceedings of the 19th International Meshing Roundtable* (pp. 177-194).
21. Tautges, T. J., and Jain, Rajeev. (2010). Mesh Copy/Move/Merge Tool for Reactor Simulation Applications. *Report of US DOE, Reactor Campaign*. ANL/MCS-TM-343.
22. Tautges, T. J., and Jain, Rajeev. (2011). Report of FY11 Extension to MeshKit and RGG. *Report of US DOE, Reactor Campaign*. ANL/MCS-TM-316.
23. Jain, Rajeev and Tautges, T. J. (2012). RGG: Reactor Geometry (and Mesh) Generator. *International Congress on the Advances in Nuclear Power Plants, Chiacgo*.
24. CGM Installation Instructions:
<https://trac.mcs.anl.gov/projects/ITAPS/wiki/CgmFromScratch>
25. MOAB Installation Instructions:
<https://trac.mcs.anl.gov/projects/ITAPS/wiki/BuildingMoab>

Appendix A. Configure/Build/Installation of MeshKit

MeshKit is maintained as Open Source Software under an LGPL license, and is therefore distributed in source code form. The library uses several required and optional

libraries which must be built and installed prior to MeshKit installation. MeshKit is currently supported on Linux and Linux-like operating systems (including MacOS); support for Microsoft Windows is under development and should be available by the next release.

Prerequisites

MeshKit requires the following libraries to be installed before configuration:

- CGM: a library for representation, query and modification of geometric models; see [24] for details on obtaining and building CGM.
- MOAB: a library for representing structured and unstructured mesh; see [25] for details on obtaining and building MOAB.
- Autotools: this is a set of Linux utilities for configuring software packages. Autotools can be found in most Linux package managers, and usually consists of the Autoconf and Automake packages.

In addition, if a parallel version of MeshKit is desired, the user must have the Message Passing Interface (MPI) available on their computer; binary versions of MPI can be found in most Linux package managers.

Download, Configure, Build, Install

MeshKit source code is maintained in a world-readable svn repository, located at <https://svn.mcs.anl.gov/repos/fathom/MeshKit/trunk/>. By default, MeshKit uses a GNU Autotools-based configuration process. The following steps should be used to configure, build, and install MeshKit:

- Unpack the source tarball into a directory referred to below as <MK_DIR> and change directory into that location.
- Execute ‘autoreconf –fi’. This executes a series of tools in the autotools suite, storing some generated files in the ‘config’ subdirectory.
- Execute ‘./configure’ with appropriate options. Two configure options are required, specifying the locations of CGM (--with_igeom=<location>) and MOAB (--with-imesh=<location>). Other useful configure options are the installation location (--prefix=<location>) and specifying debug or optimized builds (--enable-debug, --enable-optimized, respectively). For a complete list of options, execute the command ‘./configure –help’. After a successful configuration, a set of Makefile’s are generated in the proper subdirectories.
- To complete the build of MeshKit, execute ‘make’.

- To install MeshKit, execute ‘make install’. If the install location was not specified on the configure line, one can specify a location in this step by using the command ‘make prefix=<location> install’.

For those wishing to use the Python interface, MeshKit and its dependencies should be configured to build shared libraries, using the ‘--enable-shared’ configure option where appropriate.

Once the MeshKit library has been built, it is ready for inclusion into user-developed applications (any MeshKit-packaged programs, e.g. those that constitute RGG, will be installed in the ‘bin’ directory). To aid in building user-developed applications, MeshKit also writes a file ‘meshkit.make’, which can be included directly into application makefiles. This file defines the following make variables useful for building MeshKit-based applications:

- **MESHKIT_INCLUDES, MESHKIT_CPPFLAGS:** compiler options pointing to all directories containing include files available to applications, including those for CGM and MOAB; also, CPP definitions controlling which optional external meshing tools have been configured into MeshKit.
- **MESHKIT_LIBS_LINK:** linker options necessary to satisfy all functions included in MeshKit.

The ‘examples’ subdirectory in the MeshKit source installation contains an example makefile showing how these make variables can be used to compile and link MeshKit-based applications.

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