Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2016) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable

Open-source advancing front surface meshing for MeshKit

Evan VanderZee[a], Vijay Mahadevan[a], Iulian Grindeanu[a]

[a]*Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, United States of America*

Abstract

We present details on the advancing front surface meshing algorithm that has been developed to provide surface meshing capabilities for the mesh generation component (MeshKit) that is part of the SIGMA toolkit. The open-source implementation has been abstracted to work with geometry and mesh data-structure agnostic interfaces, thereby providing an extensible and efficient tool for performing parallel mesh generation on complex geometries. The algorithm can be readily adapted to utilize either on-node or inter-node parallelism by utilizing the graph-based architecture in MeshKit and can be easily extended to use different templates for advancing the front or different two-dimensional approximations to the surface geometry. Preliminary results for several test problems are shown and performance of the algorithm in comparison to NETGEN is presented.

Peer-review under responsibility of the organizing committee of IMR 25.

Keywords: mesh generation; parallel; extensible; advancing front

1. Introduction

The advancing front method for mesh generation has been used to generate meshes for many 2-D and 3-D geometries and has been utilized for a variety of applications over the years. The roots of the 2-D version of the algorithm go back to at least 1985 [1], when Lo proposed a technique that bears resemblance to an earlier 3-D meshing method described by Nguyen-Van-Phai [2]. In 1997, Schöberl advertised the NETGEN library, an open-source implementation of the advancing front method in both two and three dimensions, and described the concept of matching the front with various rules or templates to decide when to add points and elements to the mesh [3].

In this research note, we describe an implementation of the 2-D advancing front technique that is being integrated into the MeshKit open-source library (publicly available since 2011 [4]). MeshKit supports generation of meshes with a variety of methods, as well as preparation of the geometry for meshing and post-processing of meshes, including mesh smoothing. NETGEN's implementation of the 3-D advancing front technique is among the mesh generation methods that MeshKit makes available. It was easy for MeshKit to provide an adaptor to that implementation because it works with input consisting of facets bounding a volume. However, the interface to NETGEN's 2-D advancing front algorithm for OpenCascade (OCC) geometry is not general enough to be invoked directly from MeshKit. The authors

* Evan VanderZee. Tel.: +1-630-252-4938 ; fax: +1-630-252-6073.
E-mail address: vanderzee@anl.gov

Peer-review under responsibility of the organizing committee of IMR 25.

also explored the possibility of integrating a surface mesher from GRUMMP [5], but that method may insert points on a surface's bounding edges, which is undesirable for parallel mesh generation. Eventually the authors chose to implement another open-source advancing front surface meshing algorithm, designing it to be flexible and extensible enough to handle complex mesh generation workflows.

Triangle meshing methods previously supported by MeshKit include Triangle [7], which works with only planar straight line graphs and does not support generating triangle meshes for curved geometry, and a CAMAL triangle meshing algorithm that is not open-source. The addition of this surface meshing algorithm to MeshKit completes a parallel tetrahedron element mesh generation pipeline that is completely open-source. At the geometry engine level the pipeline involves OpenCascade. The support for OpenCascade is provided through an open-source version of the Common Geometry Module (CGM) [6], which provides uniform interfaces to query and manipulate multiple solid geometry models.

The 2-D advancing front implementation discussed in this paper is inspired by the original NETGEN implementation but improves upon it in several ways. These include:

- support for meshing only specified surfaces,
- support for utilizing on-node or inter-node parallelism through surface decomposition,
- easier adaptation to alternative geometry modeling engines,
- easier experimentation with two-dimensional geometry approximations, and
- more general definition of rules to advance the front.

The next three sections of the paper discuss these improvements in greater detail. After that there is a section that shows preliminary results of the algorithm. The paper closes with some thoughts about future work that might build on this implementation.

2. Meshing Specified Surfaces

Many use cases for mesh generation require capability to apply a meshing method to only specific surfaces of a geometric model. A good example of this is sweep meshing. For sweep meshing, a quadrilateral mesh that exists on a source surface is swept through a volume, creating layers of hexahedra through the volume until it reaches the target surface, where a transformation or replication of the mesh from the source surface is applied. In order to do this, the source surface must first be meshed by itself. One means of creating an initial quadrilateral mesh on the source surface that is to start by making a triangle mesh of the source surface and convert the triangle into a quadrilateral mesh. MeshKit intends to support this using the jaal quadrilateral meshing method of Verma and Tautges [8].

Another important reason to support meshing specified surfaces of a geometric model is that this makes it possible to divide up and parallelize the problem of meshing a geometric model in a natural way. Each geometric edge can be meshed as a separate subproblem. Then each geometric surface can be meshed independently. Finally each volume can be meshed independently. Since the advancing front method preserves the boundary mesh, it fits this parallel meshing paradigm well.

3. Changing the Geometry Model

The first argument to the method that does the advancing front surface meshing is an `AF2LocalTransformMaker`. This pure virtual class, together with another pure virtual class named `AF2LocalTransform` form the entire interface between the advancing front method and the geometry model.

The `AF2LocalTransform` class contains two methods. These methods, named `transformFromSurface` and `transformToSurface`, handle mapping 3-D points that are on the surface to 2-D points in a planar local approximation to the surface and mapping points from that 2-D approximation back to 3-D points that are on the surface. The `AF2LocalTransformMaker` class has only one method. This method, the `makeLocalTransform` method, takes a 3-D baseline edge together with 3-D points and other 3-D edges on the advancing front that are near the baseline edge, and returns an `AF2LocalTransform` that is appropriate to use in the neighborhood of those points and edges on the 3-D surface. For surfaces that are planar, an `AF2LocalTransformMaker` might use a simple rotation and

translation of the 2-D coordinate system of its plane in every neighborhood, but for surfaces that have high curvature or no parameterization, it can use a different planar approximation in each neighborhood.

This simple interface between the algorithm and the geometry engine makes it fairly easy to pull the method out of MeshKit and use it with another geometry engine. In fact, in the interest of comparing the overhead of MeshKit and CGM with direct calls into OCC, the authors pulled the algorithm out of MeshKit and implemented the geometry engine with direct calls into OCC. The version with direct calls to OCC is the basis for runtime performance comparisons in the results section. The authors also successfully adapted the implementation to use a proprietary geometry engine.

The key motivation for using a flexible adaptable interface to the geometry engine was to facilitate researching different methods for approximating a 3-D surface with a 2-D coordinate system. The ease of pulling the advancing front surface meshing algorithm out of MeshKit and using it with other geometry engines is a bonus. The only method implemented so far for 2-D approximation to a surface embedded in 3-D produces a plane with a normal vector coinciding with the mean of the normal vectors at the endpoints of the baseline. Points are projected to this plane using orthogonal projection and moved back to the surface by finding the intersection point of a line orthogonal to the plane with the surface.

There are other ideas that could be explored for transforming between a 3-D surface and a 2-D approximation. When a surface parameterization is available, utilizing that as the 2-D approximation might be significantly faster than querying the exact geometry. This may or may not be a good approximation, though, if the surface has high curvature. Points could be transformed from the surface onto the plane by intersecting a line orthogonal to the surface with the plane and transformed from the plane back to the surface by finding the point on the surface that is closest to the point on the plane. Maybe a conformal mapping between the surface and a plane would be the best thing to use in some contexts.

4. Defining Rules to Advance the Front

A full discussion of rule definition is outside the scope of this paper, but a brief introduction will give a flavor of what is possible. The basic structure of a rule includes a generic definition of something that must currently exist on the advancing front and something new that will be added to the advancing front. Along with the definition of what must exist, there is a definition of what may not exist — a concept called a free zone — to ensure there is enough open space that advancing the front will not infringe on the existing mesh. Most rules change depending on a quality level so that the rule will create high-quality elements if possible but can adapt to a difficult situation and create lower-quality elements if necessary.

As described, this rule definition structure was available in the NETGEN implementation. It is generalized somewhat in the MeshKit implementation, however, and brought to the forefront by including the list of rules in the constructor for the algorithm. There are two main areas where MeshKit generalizes the rule definition. The first area is the definition of the free zone. NETGEN supported free zones that have the shape of a convex polygon. The `AF2FreeZone` pure virtual class of the MeshKit implementation allows anything that can test whether points are contained in the free zone and whether lines intersect the free zone. The other area of generalization is defining the actual location of points that are added by a rule. NETGEN allowed a linear transformation. In MeshKit, the pure virtual class `AF2PointTransform` allows any transformation that may depend on the actual position of the parts of the advancing front that match what the rule specified had to exist.

5. Preliminary Results

The implementation has been applied on more than twenty geometry models, with varying degrees of complexity. The algorithm terminates successfully on all of these models if the sizing is sufficiently small and the starting edge mesh is of reasonable quality. Figure 1 shows visualizations (created in the VisIt environment [9]) of the resulting mesh for two of these models. The results demonstrate that the implementation can accommodate a variable mesh sizing function as well as a uniform mesh sizing function. The meshes shown in the figure are the direct outcome of the advancing front method, without smoothing applied. In the model of the die, in particular, it is possible to see

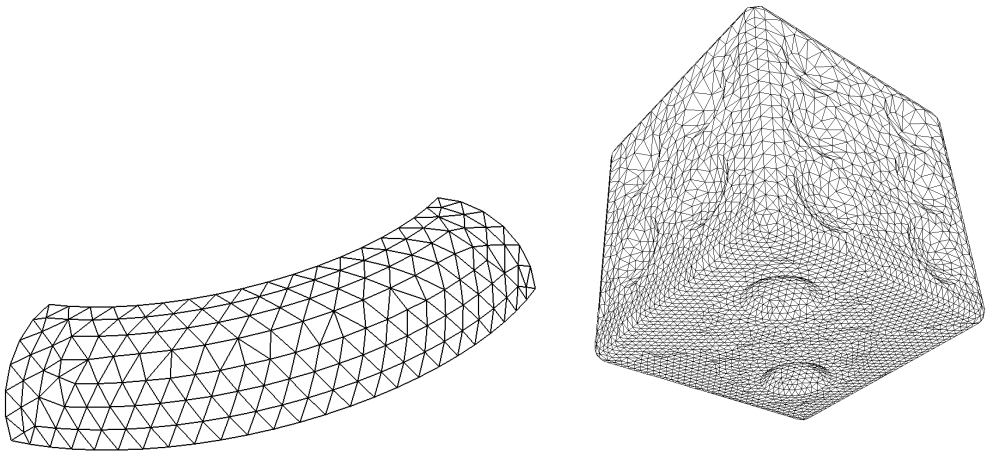

Fig. 1: A uniform size mesh of a portion of a torus and a variable size mesh of a die

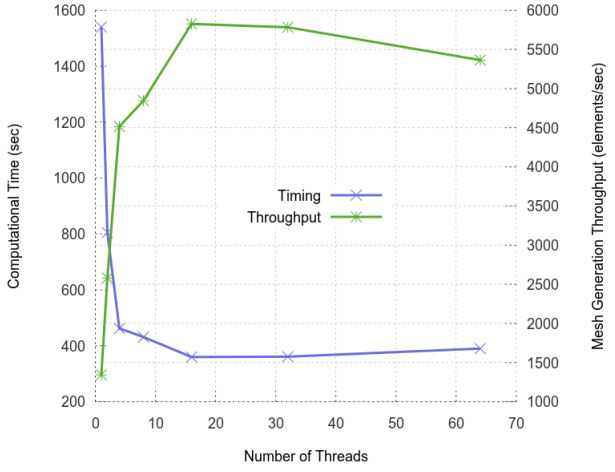

Fig. 2: Strong (thread) scaling performance for generating about 2M triangles on the surface of the die

that high quality elements near the boundary give way to poorer quality elements as the advancing fronts started from different curves collide. The quality would improve with smoothing.

Meshing each of the 176 edges and 73 faces in the geometry model of the die is a standalone subproblem. The direct OCC implementation has preliminary support for thread-parallel decomposition to accelerate the surface mesh generation. Figure 2 reports the scalability and mesh generation throughput for a uniform size mesh of the die. Due to a small number of surfaces and load imbalance, even with dynamic scheduling, the parallel efficiency quickly drops beyond 16 cores. In the future, we plan to utilize the dependency handling of the faces on the edges exposed through the graph-based MeshKit design and further improve the runtime efficiency with better load balancing strategies.

To compare the performance of the advancing front implementation with the optimized NETGEN implementation for OCC geometry models, we generated uniform-size meshes of three different models—a brick with a square hole, a circular cylinder, and the 6-sided die—at various resolutions. Performance results expressed in terms of the number of elements generated per second are shown in Table 1. At the highest resolution the mesh generation throughput of our implementation and the NETGEN implementation saturate at about 8000 el/s and 27000 el/s on average, respectively.

6. Conclusions and Future Work

Several improvements to the implementation are planned for the future. The first improvement relates to determinism. As it stands, the order of attempts to advance the front depends on memory addresses, so the algorithm execution

Table 1: Performance comparison of advancing front implementations

Mesh Resolution	AFOCC (BrickHole)	Netgen (BrickHole)	AFOCC (Cylinder)	Netgen (Cylinder)	AFOCC (Die)	Netgen (Die)
0.5	18971.22	16989.74	8105.44	16134.28	4572.67	917.83
0.25	23092.22	30735.89	7933.17	28002.99	5498.11	2124.50
0.125	22615.01	39618.70	8179.03	36591.23	6822.95	6600.35
0.0625	22089.94	34202.69	7648.09	35494.88	7673.76	15985.67
0.03125	18172.82	27689.48	6782.78	28337.89	7740.20	24552.23
0.015625	11554.66	23275.57	5446.82	33640.24	7143.01	23625.14

path is not reproducible on subsequent runs, making it harder to debug and verify results. Some conventions need to be established to bring a consistent order to advance the front.

The runtime efficiency of the initial implementation is poor compared to NETGEN's implementation. The authors suspect that the biggest contributor to poor performance is using complicated geometric computations to build a 2-dimensional approximation when the parameter space of the surface parametrization is a sufficient 2-dimensional approximation. In any case, future work is required to analyze and improve the runtime performance of the algorithm.

The authors also would like to enhance the implementation to support parallel meshing of a single surface. This includes ideas to improve the preliminary on-node threading support and inter-node message passing implementations to accelerate the computation of the surface meshes. Investigations on using the surface area of a geometric face, and the specified sizing function as parametric indicators can provide hints on predicting the total computational load. Such hints can help improve the performance considerably through better surface decomposition techniques and thread scheduling strategies.

Another avenue for research is attempting to adapt the advancing front implementation to paving scenarios. It is possible to define a different set of rules that would add quadrilateral elements rather than triangular elements, and Schöberl suggested this possibility in his NETGEN paper [3]. Investigations are needed to verify if the quadrilaterals created in regions of front collision are of acceptable quality. Handling front collisions is typically more complicated in paving than advancing front triangle meshing, and might require a different definition of the front than the implementation discussed here.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Nuclear Energy, Nuclear Energy Advanced Modeling and Simulations (NEAMS) program, under the contract DE-AC02-06CH11357.

References

[1] S. H. Lo, A new mesh generation scheme for arbitrary planar domains, International journal for numerical methods in engineering 21 (1985) 1403–1426.

[2] Nguyen-Van-Phai, Automatic mesh generation with tetrahedron elements, International journal for numerical methods in engineering 18 (1982) 273–289.

[3] J. Schöberl, NETGEN an advancing front 2d/3d-mesh generator based on abstract rules, Computing and visualization in science 1 (1997) 41–52.

[4] T. J. Tautges, J. Kraftcheck, J. Porter, MeshKit: an open-source library for mesh generation, in: Proceedings, SIAM conference on computational science & engineering, SIAM, Reno, NV, 2011.

[5] C. Ollivier-Gooch, GRUMMP version 0.7.0 user's guide, 2016.

[6] T. J. Tautges, CGM: a geometry interface for mesh generation, analysis and other applications, Engineering with computers 17 (2001) 299–314.

[7] J. R. Shewchuk, Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator, in: Applied computational geometry towards geometric engineering, Springer, 1996, pp. 203–222.

[8] C. S. Verma, T. Tautges, Jaal: engineering a high quality all-quadrilateral mesh generator, in: Proceedings of the 20th international meshing roundtable, Springer, Paris, France, 2011, pp. 511–530.

[9] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, P. Navrátil, VisIt: An end-user tool for visualizing and analyzing very large data, in: High Performance Visualization–Enabling Extreme-Scale Scientific Insight, 2012, pp. 357–372.