XLI CILAMCE
XLI Ibero-Latin American Congress on Computational Methods in Engineering
16-19 November, 2020 FOZ DO IGUAÇU PARANÁ - BRAZIL

Multi-scale meshing for 3D discrete fracture networks

Pedro Lima[1], Philippe R.B. Devloo[1], José Villegas[2]

[1]*School of Civil Engineering & Architecture, UNICAMP*
Av. Albert Einstein 951, 13083-852, Campinas/SP, Brazil
pedrolima.ec@outlook.com, phil@unicamp.br
[2]*School of Mechanical Engineering, UNICAMP*
Rua Mendeleyev 200, 13083-860, Campinas/SP, Brazil
josebvillegass@gmail.com

Abstract. The geometric description of a Discrete Fracture Network (DFN) in the context of multi-scale methods, involves the ability of inserting multiple fractures in a predefined coarse mesh, while building volumetrical elements of smaller scale around the surface of these fractures in order to create sub-meshes inside the coarse elements. This paper presents an approach for automatic finite element meshing of fractured reservoirs suited to Multi-scale Hybrid-Mixed methods (MHM). The code is written in C++ and largely relies on two open source finite element libraries: NeoPZ and Gmsh. The main steps to the method involve: locating intersections and refining elements at those points, building a data structure that associates each element of a fracture surface to the coarse volume that encloses it, and then generate a sub-mesh of fine elements around the fractures to fill these coarse elements, without altering originally defined nodes in the coarse mesh. In order to improve the quality of geometrical elements to be generated, strategies of moving intersection points and features simplifications are also presented. Results show that the proposed technique can efficiently construct adequate 3D meshes. While relying on neighbourhood information and consistent element topologies available from NeoPZ's geometric meshes, enables optimization of multiple algorithms of geometric search that would, otherwise, require a considerable amount of floating-point operations.

Keywords: Finite element meshing, Discrete fractures, Fractured porous media, Reservoir simulation.

1 Introduction

A frequent choice for modeling the behavior of flow in fractured porous media is the Discrete Fracture Network (DFN) method (Devloo et al. [1]; Jaffré et al. [2]; Jing and Stephansson [3]). In this type of method, fractures are discretely represented by (d-1)-dimensional elements in a d-dimensional finite element mesh.

The existence of fractures that affect flow and transport are characteristic of different types of porous media for domains ranging from millimeters to hundreds of kilometers [4]. To such kind of problem, recent effort has been put into the creation of Multi-scale Methods, which are better suited approaches (see Efendiev and Hou [5] and references therein). One of such methods is the Multi-scale Hybrid-Mixed (MHM) method, introduced by Paredes [6] and Harder et al. [7], which successfully associates hybrid and mixed finite element formulations with a multi-scale approach.

These methods provide high precision and efficient simulations for problems with multiple scale heterogeneities, but carry the requirement of special types of meshes that cover two levels of discretization. Therefore, as the number of fractures increases and the network of these discrete fractures grows in complexity, meshing such domains becomes quite a challenging task. In fact, it will quickly reach unreasonable volumes of work for three-dimensional problems.

In this context, inspired by the MHM formulation introduced and implemented by Devloo et al. [1] for fractured porous media, a demand is clearly identified for a tool that handles the multi-scale mesh generation for these problems.

Multiple research works have provided solutions for mesh generation of Discrete Fracture Networks. The closest example might be found in Erhel et al. [8], where a conforming mesh generator was implemented with some optimizations to avoid sharp angles, and the mesh is applied in mixed hybrid finite element methods. Nevertheless, these authors consider only the flow through fractures with impervious rock matrix, and find no need for volumetrical meshing. For their implementation, fractures are represented by planar ellipses in $\mathbb{R}^3$ that can be truncated by the boundaries of the domain. Other noteworthy DFN meshing methods can be found in [9–13]

In the multi-scale methods context, Akkutlu et al. [14] give a formulation for the Generalized Multi-scale Finite Element Method (GMsFEM) applied to DFNs in porous media, but with not much discussion on how they use neighbourhood information to associate the fine level discretization to the coarse elements.

As a common denominator to these researches, the mesh generation step is undeniably a central constriction since convergence and stability of numerical techniques, as well as accuracy of approximation, are significantly affected by the quality of discretization. In principle, as Fourno et al. [9] point out, degenerate element geometries that do not satisfy the quality constraints should be identified and removed from the final mesh.

The present study focuses on the mesh generation problem of representing porous fractured reservoirs. Section 2 explains how the methodology is constructed and what motivates the algorithms with the assumptions that guarantee their robustness. Section 3 contains example meshes that can be built with these algorithms, together with the discussion of their performance and efficiency. Section 4 concludes the text with discussions and comments on further works.

2 Methodology

The context of Multi-scale Hybrid-Mixed methods imposes interesting constraints on the geometric description of a Discrete Fracture Network. The chief requirements are for sub-meshes that are built around the fracture surface elements to fill each coarse element, and a data structure that consistently associates fine/coarse elements.

This section presents the methodology behind the codes developed for this research. It should be noted that the main goals are:

- Conserve the coarse mesh as given by the user;
- Insert one fracture at a time and follow a methodology through which fractures do not need to concern with the existence of previous fractures.

The entirety of the code is written in C++, publicly available at https://github.com/pedrolima92/dfnMesh, and largely relies on methods available in two open source finite element libraries, namely NeoPZ[1] (Devloo [15]) and Gmsh[2] (Geuzaine and Remacle [16]).

2.1 Initial setup and fracture representation

The problem setup starts with a coarse mesh that make up the first level discretization of the domain. In such a mesh we create a skeleton mesh of lower dimensional elements that occupy the space of the lower dimensional sides of each macro element. These skeleton elements are where most of the operations will happen.

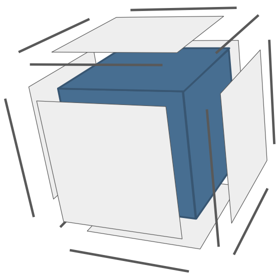
Figure 1. Face and rib elements around a coarse volume

We represent fracture locations as convex planar polygons in $\mathbb{R}^3$. Aiming for simplicity of input data, these are read by the program as a 3xN matrix in which every column is a corner point of the polygon, numbered counter-clockwise from zero to N-1.

In order to compute intersections of a fracture plane with the mesh, it is useful to determine its orientation first. This is done through the computation and storage of two tangent vectors $\mathbf{t}_0$ and $\mathbf{t}_1$, and a normal vector $\mathbf{n}_2$.

For example, Figure 2 shows a plane formed by four points, and those points are the base to compute the vectors that define its orientation.

[1]NeoPZ project: `https://github.com/labmec/neopz`
[2]Gmsh project: `http://gmsh.info/doc/texinfo/gmsh.html`

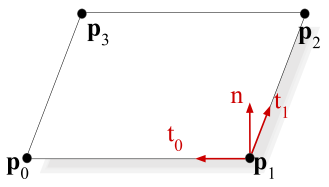

Figure 2. Plane points with its respective axes.

It is imperative to our method that, at the insertion of each fracture, we first check if the set of points that define the polygon are indeed coplanar.

2.2 Intersecting edges

With an oriented and consistent polygon, the following operation is the search for intersected edges. This search can be broken down on two main steps: First we group every node in the mesh as being above or bellow the fracture plane. Second, for each 1D element whose nodes are on opposite sides of the plane, we check if the intersection of that element and the plane is within the polygon bounds.

2.3 Intersecting faces

Relying on neighbourhood information available from NeoPZ's geometric meshes, we follow on constructing the intersections from the intersected ribs. Intersected faces are then found simply by verifying the presence of neighbour ribs, that were found in the previous step, occupying the space of its 1D sides.

The refinement pattern that follows can thus be determined specifically to each cut (and its permutations) that may rise from the problem of two 3D planar convex polygons intersecting. Classical topologies available in NeoPZ limit our possible intersections with 2D elements, as we build them from intersected edges, to 12 topologically consistent refinement patterns (Figure 3). This information is easily built from the position of intersection points and, given the domain where they are contained, the split shape is efficiently computed with no requirement for floating point operations. After finding the pattern, we simply inform the face which refinement pattern it should follow, and it is able to divide itself, storing its children in the geometric mesh data structure.

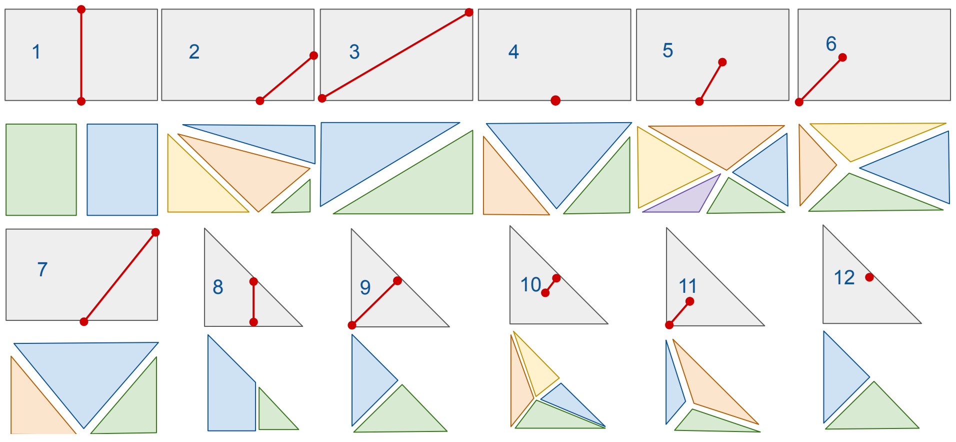

Figure 3. Possible refinement patterns for our methodology.

2.4 Repairing/coalescing intersections

In aiming for improved quality of the meshes and robustness of the code, some corrections need to be imposed on the refinements adopted for intersected elements. These corrections are, in essence, the coalescing of intersection points to their closest neighbour side, whenever tolerable distances or angles have been violated.

At the rib level, the one tolerance that can be violated is the distance of the intersection node to either one of the nodes. The handling of this violation is simply snapping the would-be-new point down to the closest old node.

At the face level, we require every angle resulting from a refinement to be greater than a tolerable angle. So we measure every angle of each child element of the refinement and, if any angle violates this condition, we move the intersection nodes down to their closest node.

2.5 Surface meshing

After the mesh elements have been properly refined to conform to the new fracture, we move on to the surface meshing of the fracture itself.

To mesh the fracture surface, we use Delaunay meshing algorithms from the Gmsh library [16] but need first to establish the constraints on that surface. In the process of refinement so far, we have built intersection points on each of the intersected elements, and the constraints for the triangulation of the surface arises naturally from the connection of these point. Such connections will happen in 2 ways: First, those that are part of the faces' domains. We simply connect them; Second, those that are part of the contour of the fracture. Which we connect by matching points that exist in the domain of faces and are at minimal distance to either one of the edges of the fracture. For that, we need to insert the corners of the fracture, and sort these nodes counter clockwise by distance to the initial node of the edge to which they have been matched.

The modifications made to the refinements of face elements imply that the surface of the fracture should be, more often than not, non-planar. This is a significant imposition that we should not expect Gmsh to deal with in a reliable manner. So another operation that is indispensable before feeding the constraints to Gmsh, is the projection of points onto the original fracture plane. This operation limits somewhat the robustness of the code since big distortions to the fracture may generate surface projections that are poor representation of the desired mesh.

Another implication of the intersection coalescing is that elements of the mesh can now be incorporated into the fracture surface. Therefore, to avoid duplicate elements created by Gmsh, we search for face elements that should be incorporated and feed it to Gmsh as 2D constraints that should not be meshed. The condition for a face to be incorporated is simply to have all of its edges found to belong to the set of defined 1D constraints.

2.6 Identify enclosed elements by volume

With properly triangulated fracture surfaces and original mesh refined in accordance, we are now able to use Gmsh's library to build the volumetrical mesh around the fractures. However, Gmsh does require specific data structures that have to be set first.

A required information that is central and not promptly obtainable from NeoPZ's data structures, is that volumes know which surfaces are enclosed (embedded) within them.

To naively compute such information by checking every fracture surface element against every volume is quite an expensive operation. So, to efficiently build this data, we rely again on neighbourhood information from NeoPZ geometric elements.

By navigating neighbours until a face of the original skeleton mesh is found, we can quickly filter the volumes that may enclose a face down to 2 candidates (since a face can only interface two volumes in a conformal 3D mesh).

2.7 Volume meshing

Finally, to get the volumetrical submeshes around fractures and inside macro elements, we use the data-structures built up to this point.

Geometric entities in Gmsh, as described by Geuzaine and Remacle [16], are defined from the boundary representation of their topologies. Points are simply built from their coordinate vector in $\mathbb{R}^3$, curves are bounded by a pair of points, surfaces are bounded by an oriented loop of curves, and volumes are bounded by a shell of surfaces. For this, we build the shells that define the volumes using the sub-elements defined during the refinement of 2D elements to incorporate the intersections with fractures. While fracture surface elements are 2D elements embedded inside those volumes.

Conveniently, the interface of the code with Gmsh can be done through their API (Application Programming Interface), which means the code is fully automatized without the need for Gmsh's graphical interface.

3 Application examples

In this section, we consider two examples of application for our methodology. All the results presented here were generated with public available code that is setup at `https://github.com/PedroLima92/papers/2020/cilamce/dfnMesh`.

The first example, whose static graphical representation can be seen in Figure 4, consists of a moving fracture that sweeps through a mesh of 8 coarse elements. The choice for a moving fracture was made in order to attest to the robustness of the code, which safely intersects the coarse mesh and continuously inserts the fracture along the chosen interval. The code is executed multiple times from the initial position of insertion, at the center of the domain, outward to its limits. The tolerable length is set to 0.8, such that the fracture surface gets coalesced and incorporated into the original mesh whenever this tolerance is violated.

An animation that shows the complete interval of movement of the fracture can be consulted by the reader at the informed repository.

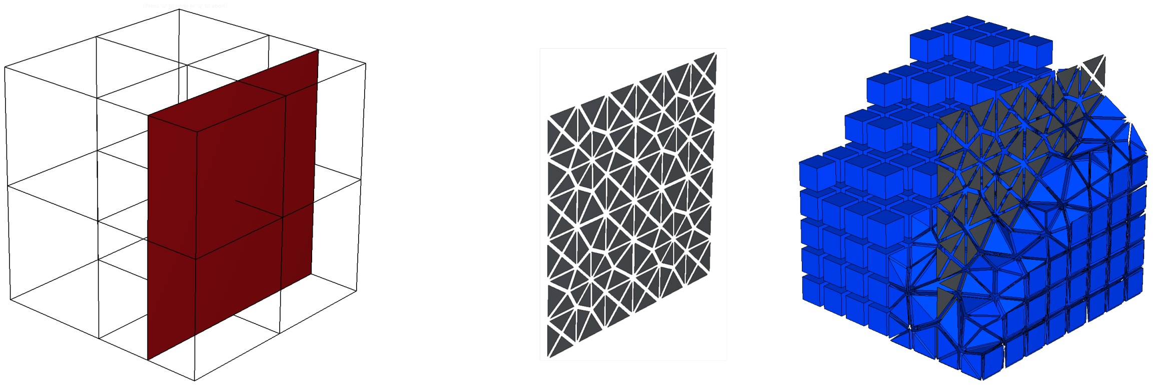
(a) Polygon and coarse mesh (b) Fracture surface mesh (c) Sub-mesh

Figure 4. Example 1. A single fracture sweeping an interval of the domain

The second example, whose graphical representation can be seen in Figure 5, consists of a pair of intersecting fractures in the same coarse mesh used in the first example. Here we use this example to demonstrate the capabilities of the method to insert a new fracture through the same steps used in the creation of an isolated fracture. As the new fracture is inserted, the surface of the previous fracture is refined to conform.

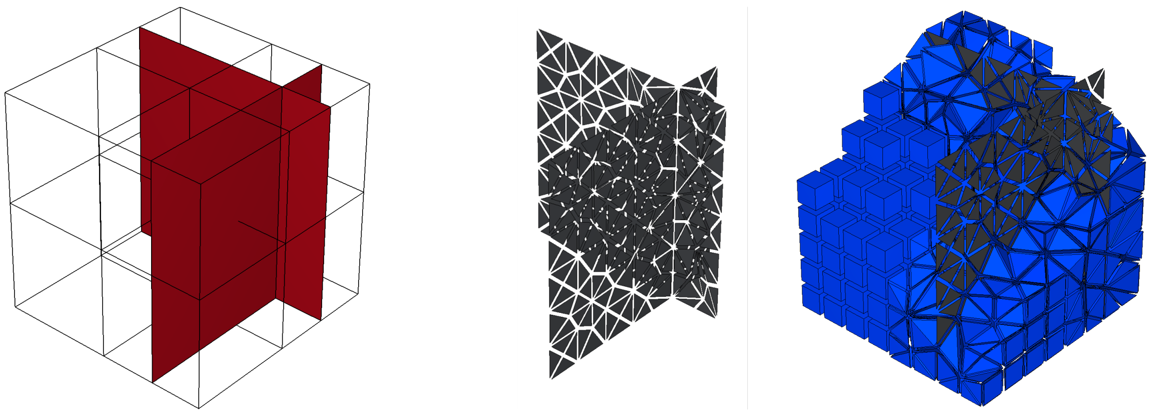
(a) Polygon and coarse mesh (b) Fracture surface mesh (c) Sub-mesh

Figure 5. Example 2. Two insersecting fractures

4 Conclusions

In this paper, we have proposed a robust technique for meshing fractured porous media in the context of multi-scale methods. We should highlight that the core choices that stem the whole method come from starting with a coarse mesh, and defining intersections from the one-dimensional elements created as skeletons superposed on the original mesh edges.

One important goal set up and achieved so far in this project is to follow a path through which single fractures leave the mesh in proper settings, such that new fractures do not have to distinguish from elements that are part of a previous fracture or the original coarse mesh. In summary, we take as a guiding philosophy, that, if the methodology works for one fracture, it works for multiple fractures.

It should be noted that this is an ongoing and early stages research. As such, many improvements that aim to increase the robustness and scalability of the code are expected in the near future.

Some current limitations ought to be clear to the reader. For example, our method is exclusively dependant on at least one edge of the mesh being intersected. Without this minimal requirement, the presented approach cannot associate the fracture to the domain.

Another important limitation is the lack of robust treatment for the edges of the fractures. Consequently, a proper use of the code is currently dependant on fractures completely crossing the boundaries of the domain. For that, these are actually one of the current focus of research efforts of the authors, and follow up papers will contribute to address this matter.

Other future developments of this research should expect efforts to reduce the strain put on Gmsh meshing algorithms, by identifying polyhedral volumes created by the insertion of fractures, which are much simpler domains to generate volumetrical meshes.

Acknowledgements. The authors aknowledge the support of Brazilian Higher Education Improvement Coordination (CAPES) and Secretaria Nacional de Educación Superior, Ciencia, Tecnología e Innovación (Senescyt, Ecuador).

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Devloo, P. R. B., Teng, W., & Zhang, C. S., 2019. Multiscale hybrid-mixed finite element method for flow simulation in fractured porous media. *CMES - Computer Modeling in Engineering and Sciences*, vol. 119, n. 1, pp. 145–163.

[2] Jaffré, J., Mnejja, M., & Roberts, J. E., 2011. A discrete fracture model for two-phase flow with matrix-fracture interaction. *Procedia Computer Science*, vol. 4, pp. 967–973.

[3] Jing, L. & Stephansson, O., 2007. Discrete Fracture Network (DFN) Method. In *Developments in Geotechnical Engineering*, volume 85, chapter 10, pp. 365–398. Elsevier.

[4] Berre, I., Doster, F., & Keilegavlen, E., 2019. Flow in Fractured Porous Media: A Review of Conceptual Models and Discretization Approaches. *Transport in Porous Media*, vol. 130, n. 1, pp. 215–236.

[5] Efendiev, Y. & Hou, T. Y., 2009. *Multiscale Finite Element Methods*, volume 4. Springer New York, New York, NY.

[6] Paredes, D., 2013. *Novos Métodos de Elementos Finitos Multi-Escalas: Teoria e Aplicações*. Phd thesis, Laboratório Nacional de Computação Científica - LNCC.

[7] Harder, C., Paredes, D., & Valentin, F., 2013. A family of Multiscale Hybrid-Mixed finite element methods for the Darcy equation with rough coefficients. *Journal of Computational Physics*, vol. 245, pp. 107–130.

[8] Erhel, J., De Dreuzy, J. R., & Poirriez, B., 2009. Flow simulation in three-dimensional discrete fracture networks. *SIAM Journal on Scientific Computing*, vol. 31, n. 4, pp. 2688–2705.

[9] Fourno, A., Ngo, T. D., Noetinger, B., & La Borderie, C., 2019. FraC: A new conforming mesh method for discrete fracture networks. *Journal of Computational Physics*, vol. 376, pp. 713–732.

[10] Mustapha, H., 2011. G23FM: A tool for meshing complex geological media. *Computational Geosciences*, vol. 15, n. 3, pp. 385–397.

[11] Hyman, J. D., Gable, C. W., Painter, S. L., & Makedonska, N., 2014. Conforming delaunay triangulation of stochastically generated three dimensional discrete fracture networks: A feature rejection algorithm for meshing. *SIAM Journal on Scientific Computing*, vol. 36, n. 4, pp. 1871–1894.

[12] Zhang, Y., Gong, B., Li, J., & Li, H., 2015. Discrete fracture modeling of 3D heterogeneous enhanced coalbed methane recovery with prismatic meshing. *Energies*, vol. 8, n. 6, pp. 6153–6176.

[13] Wang, Y., Ma, G., Ren, F., & Li, T., 2017. A constrained Delaunay discretization method for adaptively meshing highly discontinuous geological media. *Computers and Geosciences*, vol. 109, n. January, pp. 134–148.

[14] Akkutlu, I. Y., Efendiev, Y., & Vasilyeva, M., 2016. Multiscale model reduction for shale gas transport in fractured media. *Computational Geosciences*, vol. 20, n. 5, pp. 953–973.

[15] Devloo, P. R. B., 1997. PZ: An object oriented environment for scientific programming. *Computer Methods in Applied Mechanics and Engineering*, vol. 150, n. 1-4, pp. 133–153.

[16] Geuzaine, C. & Remacle, J. F., 2009. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, vol. 79, n. 11, pp. 1309–1331.