A Computational Tool for the Geometric Modeling of Naturally Fractured Carbonate (Karst) Petroleum Reservoirs

Filipe A. C. S. Alves[1], Artur C. R. de Souza[2], Darlan K. E. de Carvalho[3], Paulo R. M. Lyra[3]

[1]*Center for Computer Science, Universidade Federal de Pernambuco*
Av. da Arquitetura, S/N, CEP 50740-550 Recife/PE, Brazil
facsa@cin.ufpe.br
[2]*Dept. of Civil Engineering, Universidade Federal de Pernambuco*
Av. da Arquitetura, S/N, CEP 50740-550 Recife/PE, Brazil
artur.castiel@ufpe.br
[3]*Dept. of Mechanical Engineering, Universidade Federal de Pernambuco*
Av. da Arquitetura, S/N, CEP 50740-550 Recife/PE, Brazil
darlan.ecarvalho@ufpe.br, paulo.lyra@ufpe.br

Abstract. One of the main reasons why the modeling of carbonate (karst) reservoirs is a challenging task is the geometric representation required to capture the complex geological structures in these reservoirs, such as fractures, vugs, cavities and caves. Mesh generation from a geometrical model is thus affected by the choices made for vugs and fractures representations. An alternative approach is to assign geological features to the volumes in a fine scale mesh as physical properties. These discrete models are required when, for instance, Stokes-Brinkman's equations are adopted to model multiphase flow simulation in carbonate karstic oil reservoirs, found often in Brazilian Pre-Salt. In this paper, we propose a computational tool for generation of karst reservoir scenarios by creating a geometrical model of the geological features and then computing the mesh volumes inside vugs, cavities, caves and fractures. Vugs are represented by randomly distributed and possibly overlapping ellipsoids in a three dimension space. They might be connected by fractures, modeled as cylinders, ellipsoids or boxes. Face connectivity between mesh volumes in a fracture is assured. The code was written in Python using the NumPy library and the IMPRESS (Intuitive Multilevel Pre-processor for Smart Simulation), an "in-house" pre-processor used for mesh management. Example scenarios were generated to evaluate the correctness and robustness of the developed computational tool.

Keywords: Carbonate (karst) reservoir, fractures, mesh generation, computational geometry

1 Introduction

The simulation of carbonate (karst) reservoirs is a challenging task due to the presence of large void spaces such as fractures, cavities, and caves, also known as "vugs" in the geology literature. In addition, as outlined by Gulbransen et al. [1], those features are not well separated from the porous rock matrix. For this reason, the Stokes-Brinkman equations are usually adopted for the flow simulation in the context of naturally fractured karst reservoirs.

The representation of fractures and vugs in those reservoirs can be complex and time consuming. The model can be generated by in situ observations to gather data and model specific scenarios. Such representations can be very accurate and provide a depiction directly related to the geology and physics of the medium. Nevertheless, the data is hard to obtain and still very specific to a single scenario, as pointed by Gringarten [2]. Alternatively, it is possible to characterize fractures and vugs as simple geometrical entities randomly distributed in the space domain. Despite not having a physical value, those shapes can be generated inexpensively and might be sufficient for preliminary simulations.

In turn, fractures can be represented in the same dimension as the rock matrix mesh or in inferior dimensions. The latter is the most common approach when a coupled Darcy-Stokes model is adopted, as exemplified by Alghalandis [3]. It requires an individual discretization that might or not be conforming to the rock matrix mesh. However, for the Stokes-Brinkman model, it is convenient to use a same dimension depiction, since the flow in the fracture is indistinguishable from the porous rock. This allows the flow in the meshed fractures and vugs to be simulated alongside the flow in the rock matrix.

In this work, we present a computational tool for 3D geometric modelling of naturally fractured karst reservoirs in the context of flow simulation modeled by Stokes-Brinkman equations in a single domain approach. Based on the work of Caminha et al. [4], the geometrical model is randomly generated and then inserted into the matrix mesh. Fractures are represented in the same dimension as the rock matrix.

The outline of this paper is as follows. In section 2 we present the implemented algorithms and data structures that enable the tool. Afterwards, the generation of some scenarios is shown and discussed. Finally, conclusions are drawn and the future works are summarized.

2 Computational Approach

The algorithms were developed using the Python language following the object-oriented programming (OOP) paradigm. For the mesh management, the Intuitive Multilevel Preprocessor for Smart Simulation (IMPRESS) [5] was used. It is a Python module based on MOAB (Mesh Oriented Database) [6, 7] that allows for easy access to mesh elements and mesh properties. Moreover, functions and operators required to generate the geometrical features were implemented using Numpy [8].

Figure 1 shows the main steps of the procedure for generation of a discrete reservoir scenario. The inputs to the program are a mesh of the medium obtained with the mesh generation tool of choice, and a set of parameters containing the range of possible values for the dimensions of vugs, the number of ellipsoids for vugs and the number of fractures to be generated. The number of fractures must be less than the number of possible vug ellipsoids pairs, i.e., less than $\binom{n}{2}$ where n is the number of vug ellipsoids.

The first step of the procedure is the generation of geometrical entities representing fractures and vugs. Then, the intersections between those entities and the mesh volumes are computed, that is, for each mesh volume we calculate if it is contained within a geometrical feature and if so, a physical property is assigned. The tool is designed to find intersections for any mesh element geometry (tetrahedral, hexahedral, etc.). The output of the program is a mesh with assigned physical properties.

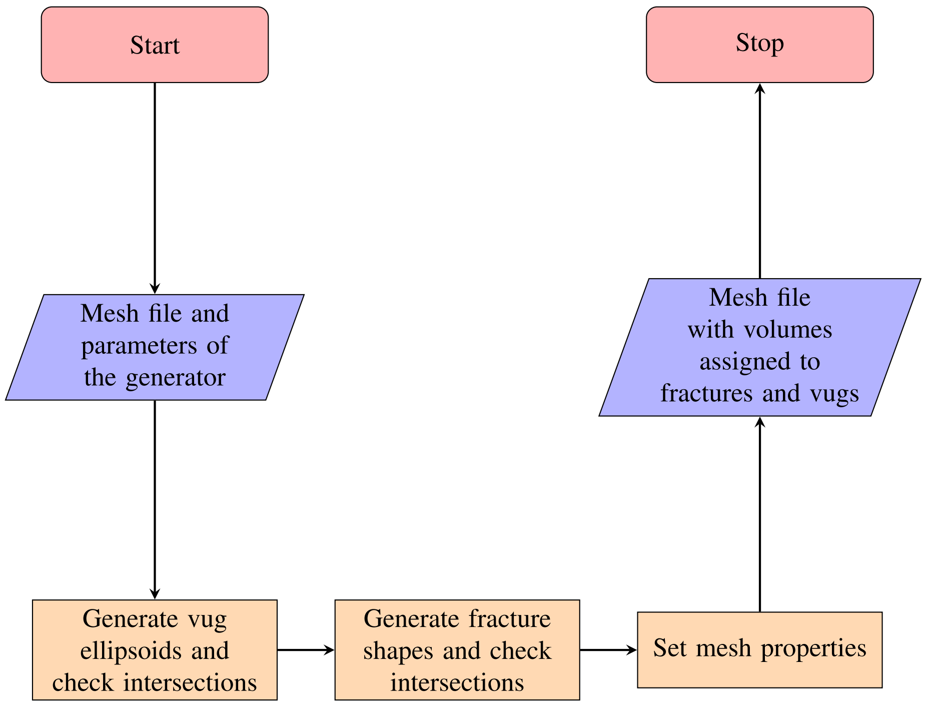

Start
Stop
Mesh file and parameters of the generator
Mesh file with volumes assigned to fractures and vugs
Generate vug ellipsoids and check intersections
Generate fracture shapes and check intersections
Set mesh properties

Figure 1. Execution flowchart of the fractures and vugs generator.

The tool was implemented for 3D model meshes. Fracture shapes can be chosen from cylinders, ellipsoids or boxes, while vugs are represented by ellipsoids. The shapes are randomly generated and rotated following an uniform distribution. At the current stage of the work, we do not take into account any physical property of the media to determine the spatial distribution of fractures and vugs nor the chosen shapes for fractures and vugs intend to reflect the real geology of the reservoir. They are a rough approximation of how fractures and vugs are shaped in real scenarios. Furthermore, we assume as a simplifying hypothesis that fractures always connect two vugs. Those assumptions are temporary limitations and shall be addressed in future works.

The size of the ellipsoids representing vugs are also determined randomly, picked from a range of parameters set by the user. These ellipsoids are allowed to overlap to form a single vug. For fractures, the dimensions are a function of the distance between the vugs it connects and the mesh volumes dimensions. The specificities vary with the chosen fracture shape and were determined empirically.

Given the distribution of the geometrical shapes, the intersections with mesh elements are computed. The general procedure for vugs is described in algorithm 1. It consists in determining if the centroid of a mesh volume is contained inside of the rotated ellipsoid, i.e, if the coordinates $\mathbf{x} = (x', y', z')$ of the centroid in the rotated axis are such that

$$\frac{(x'-x_0)^2}{a^2} + \frac{(y'-y_0)^2}{b^2} + \frac{(z'-z_0)^2}{c^2} \leq 1, \tag{1}$$

where x_0, y_0 and z_0 are the coordinates of the center of the ellipsoid and a, b and c are the ellipsoid's parameters. If equation 1 is true, a mesh property is assigned to the element indicating that it belongs to a vug. The procedure was implemented using vector operations optimized by Numpy's [8] data structures. In addition, the information about mesh entities, centroids in this case, is obtained through IMPRESS' interface, which in turn uses MOAB's tree based search algorithms.

```
Algorithm 1: Generate vugs and compute intersections
Input: $I$: interval of possible values for the ellipsoids' parameters, $N_v$: number of ellipsoids to be generated
Output: list of volumes marked as vug or not
1 Set $V$ to be an empty list to store the mesh volumes in a vug;
2 while the number of ellipsoids is less than $N_v$ do
3     Pick ellipsoid's parameters from $I$;
4     Apply a random rotation and translation to the ellipsoid;
5     Check which volumes are inside the ellipsoid;
6     Append these volumes to $V$;
7 end
```

For the fractures, a similar procedure is followed, as described by algorithm 2. First, the mesh volumes which centroids are inside the shape representing the fracture are found. This computation, however, can lead to undesired "gaps" between volumes in the fracture, that is, portions of the fracture shape that are not connected between themselves. To handle those cases, we search the intersection between the line segment determined by the two ellipsoids' centers connected by the fracture and the faces of the mesh volumes. This way, the connection between the two selected ellipsoids is assured at least along the line segment. We must point out that the input mesh is not modified to conform to the intersections. As with the vugs procedure, all operations are performed using vector operations optimized by Numpy [8].

```
Algorithm 2: Generate fractures and compute intersections
Input: $E$: list of vug ellipsoids, $N_f$: number of fractures to be generated
Output: Mesh volumes assigned to fractures
1 Find the minimal edge length $l_{min}$;
2 while the number of fractures is less than $N_f$ do
3     Let $E_1$ and $E_2$ be two ellipsoids not yet connected by a fracture;
4     Let $d$ be the distance between the centers of $E_1$ and $E_2$;
5     Choose the dimensions of the shape using $d$ and $l_{min}$;
6     Search the volumes which centroids are in the fracture shape;
7     Find the volumes which faces are intersected by the line segment defined by the centers of $E_1$ and $E_2$;
8     Assign the fracture property to the volumes that are not already assigned to vugs;
9 end
```

3 Results and Discussion

Figure 2 shows an accomplished scenario using a hexahedral mesh. All three possible fracture shapes are presented. As it can be seen, the vugs were generated as ellipsoids and the fractures connect ellipsoids forming vugs correctly, i.e., the endpoints of fractures were indeed placed in an ellipsoid forming a vug. The mesh properties were rightly assigned as well, that is, all volumes were either assigned to a fracture or a vug. The two sets of volumes are disjoint. Withal, a volume designated to one of the features always neighbors another volume in the same feature. Such property avoids the generation of incoherent scenarios as a portion of a fracture entirely surrounded by a vug and vice-versa.

Figure 3 presents in detail one of the fractures in Figure 2a comparing its generation when the intersections along the main axis are verified (3a) and when they are not (3b). It is noticeable how the addition of this step helped to prevent the generation of a fracture with disconnected parts. Additionally, Figure 3a also displays how the volumes added to the fracture by the extra step delimit the line segment connecting the centers of the ellipsoids constituting vugs and are consistent with the proposed algorithm.

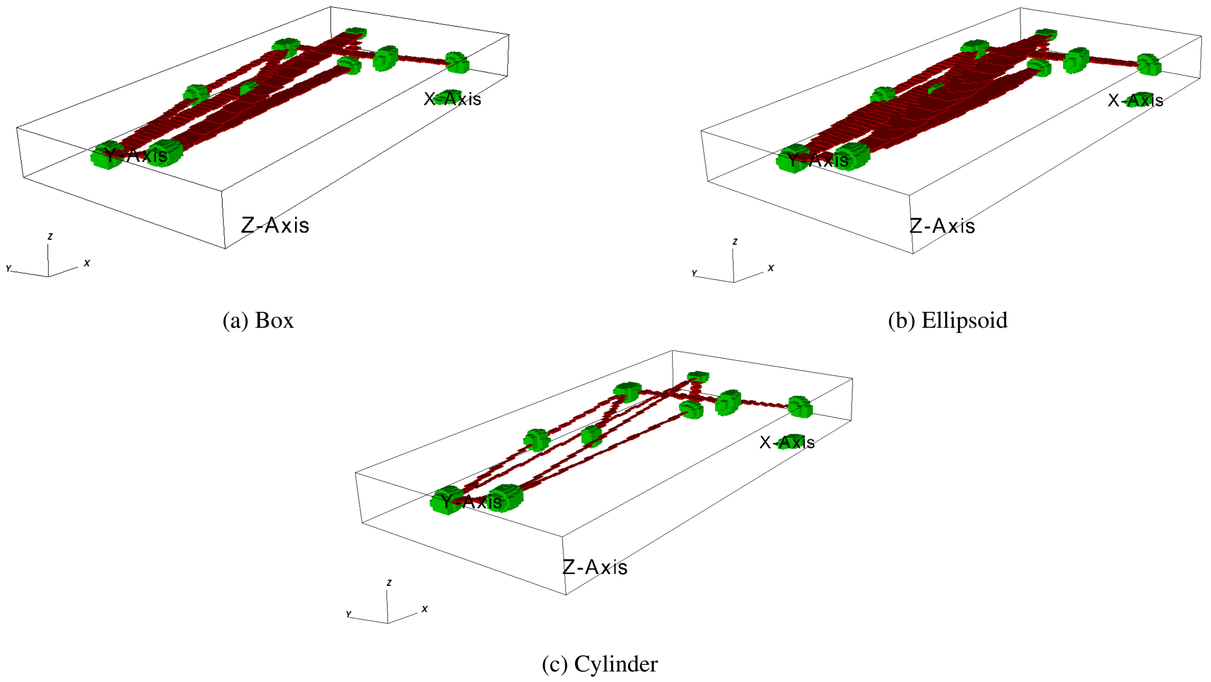


Figure 2. Scenarios obtained with a hexahedral mesh for each possible fracture shape.

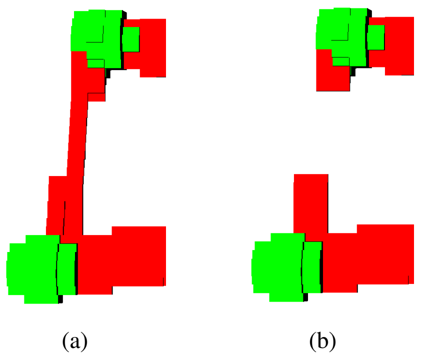
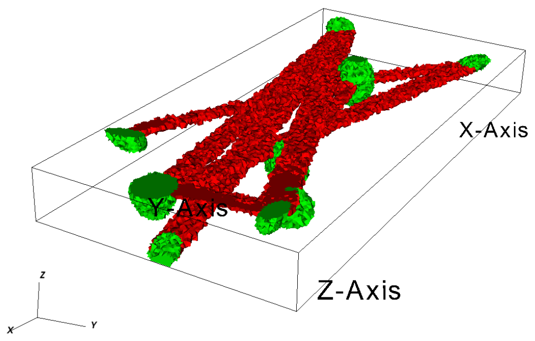
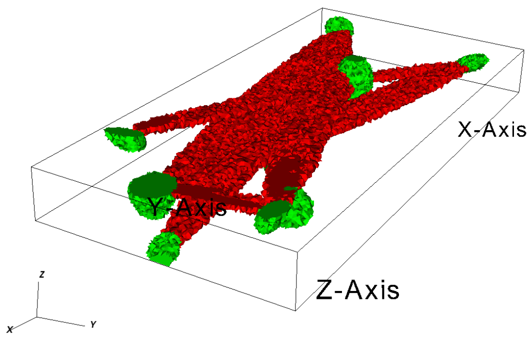


Figure 3. Comparison of a fracture generated with (a) and without (b) checking for intersections along the main axis.

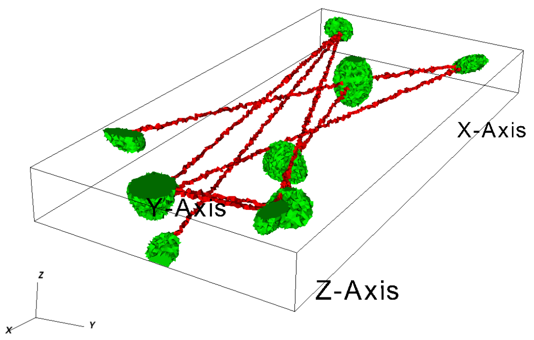
Figure 4 presents a scenario generated using a tetrahedral mesh. As with the previous example, vugs and fractures were created as expected, set according to specifications and, in the case of fractures, properly connecting two ellipsoids that compound a vug. For box and ellipsoid shaped fractures, it is notable that those shapes are slightly "bulkier" if compared with their hexahedral counterpart. That was caused by the adjustment of the shape dimensions to avoid "holes" in the fracture.



(a) Box



(b) Ellipsoid



(c) Cylinder

Figure 4. Scenarios obtained with a tetrahedral mesh for each possible fracture shape.

4 Conclusions

In this work, we introduced a tool for geometric modelling of naturally fractured karst petroleum reservoirs. As shown by the results, it is capable of generating fairly reasonable scenarios for the simulation of reservoirs using the Stokes-Brinkman model in a single domain approach. The obtained geometric models can be used as toy problems for testing simulation methods.

As an example of application, we aim to use the scenarios obtained with the presented tool alongside the 3D extended version of a cell-centered Multipoint Flux Approximation method based on harmonic points (MPFA-H) as proposed by Melo et al. [9]. This method was developed by our group to approximate the Stokes-Brinkman equations on heterogeneous porous media using unstructured meshes with a monolithic and a segregated (SIMPLEC) approaches.

In future works, we intend to improve the detection of intersections of the geometry with the mesh elements, and use physical properties from the rock matrix mesh and also derived from geostatistics to determine the distribution of fractures and vugs. It is also part of future efforts to generate representations of fractures and vugs that are closer to their real geology or even allow the user to provide a geometrical model of his own. Finally, we also plan to make the spawning of fractures more flexible, allowing to have fractures that do not connect two vugs.

Acknowledgements. We gratefully thank CNPq, FACEPE and CAPES, for providing financial aid to this project. We also would like to acknowledge the members of PADMEC for contributing with a great environment for the development of this paper and for contributing directly or indirectly with it.

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] A. F. Gulbransen, V. L. Hauge, and K.-A. Lie. A Multiscale Mixed Finite-Element Method for Vuggy and Naturally Fractured Reservoirs. *SPE Journal*, vol. 15, n. 02, pp. 395–403, 2009.

[2] E. Gringarten. 3-d geometric description of fractured reservoirs. *Mathematical Geology*, vol. 28, pp. 881–893, 1996.

[3] Y. Fadakar Alghalandis. Adfne: Open source software for discrete fracture network engineering, two and three dimensional applications. *Computers & Geosciences*, vol. 102, pp. 1–11, 2017.

[4] G. P. K. Caminha, R. B. Willmersdorf, and P. R. M. Lyra. Gerador de malha para modelos de multi-escala e multi-física aplicados na simulação em reservatórios em meios cársticos. In N. A. Dumont, ed, *Proceedings of the XXXVI Iberian Latin-American Congress on Computational Methods in Engineering*. In portuguese, 2015.

[5] A. C. R. de Souza. Impress: Intuitive multilevel preprocessor for smart simulation. Available at `https://github.com/padmec-reservoir/impress`, 2020.

[6] T. J. Tautges, R. Meyers, K. Merkley, C. Stimpson, and C. Ernst. MOAB: a mesh-oriented database. SAND2004-1592, Sandia National Laboratories. Report, 2004.

[7] V. Mahadevan, I. Grindeanu, R. Jain, P. Shriwise, and P. Wilson. MOAB v5.2.1. Available at: `https://doi.org/10.5281/zenodo.2584862`, 2020.

[8] C. R. Harris, K. J. Millman, van der S. J. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, van M. H. Kerkwijk, M. Brett, A. Haldane, del J. F. Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, vol. 585, n. 7825, pp. 357–362, 2020.

[9] P. H. M. Melo, F. R. L. Contreras, D. K. E. Carvalho, and P. R. M. Lyra. A multipoint flux approximation method based on harmonic points (mpfa-h) for the numerical solution of the stokes-brinkman equations in carbonate petroleum reservoirs. In *Proceedings of the XLI Iberian Latin-American Congress on Computational Methods in Engineering*, 2020.