GERADOR DE MALHA PARA MODELOS DE MULTI-ESCALA E MULTI-FÍSICA APLICADOS NA SIMULAÇÃO EM RESERVATÓRIOS EM MEIOS CÁRSTICOS

Guilherme Praciano Karst Caminha

gpkc@cin.ufpe.br

CIn – UFPE

Av. Jornalista Anibal Fernandes, s/n – Cidade Universitária, 50740-560, Recife, Pernambuco, Brasil

Ramiro Brito Willmersdorf

Paulo Roberto Maciel Lyra

ramiro@willmersdorf.net

prmlyra@padmec.org

CTG - UFPE

Av. Prof. Moraes Rego, 1235 – Cidade Universitária, 50740-560, Recife, Pernambuco, Brasil

Abstract. *A modelagem de reservatórios em sistemas cársticos é um problema desafiador devido à presença de estruturas chamadas vugs em seu domínio, que podem ser conectados entre si através de finas fraturas. A geração de uma malha que descreve bem este domínio requer o uso de ferramentas de modelagem de geometria e geração de malhas avançadas. Este trabalho propõe o uso das ferramentas de código aberto OpenCascade e GMSH como tecnologias facilitadoras para a modelagem da geometria e geração da malha voltada para a simulação em multi-escala e multi-física de reservatórios em meios cársticos. Na macro-escala (escala de Darcy) gera-se uma malha estruturada quadrilateral, normalmente utilizada nos simuladores comerciais de reservatórios. Na micro-escala (escala de Stokes-Brinkman) o modelo é gerado considerando-se uma distribuição aleatória de vugs modelados de forma simplificada com formato de elipses, com ou sem superposição, e que podem ainda estar conectadas por fraturas, modeladas como retângulos. Nesta escala utiliza-se uma malha não-estruturada triangular. O programa computacional foi construído utilizando a linguagem orientada a objetos C++. Alguns exemplos acadêmicos são apresentados para ilustrar a robustez e funcionalidade do sistema na geração de modelos e malhas em duas dimensões.*

Keywords: *Multi-escala, Multi-física, Geração de Malhas, Meio Cársticos*

1 INTRODUÇÃO

A simulação numérica de fluxo e transporte em reservatórios em sistemas cársticos é um problema desafiante devido à presença de cavernas, ou vugs, em seu interior. Estes vugs podem estar interconectados através de fraturas, formando uma rede, e estas fraturas ocorrem em múltiplas escalas. Sistemas cársticos são compostos de material poroso, o que nos leva a ter a co-existência de duas físicas diferentes num mesmo meio: dentro dos vugs e fraturas, possuímos domínios de fluxo livre, que são modeladas pela equação de Stokes, e nas regiões ao redor, possuímos regiões porosas, que são modeladas pela equação de Darcy (Gulbransen, 2010).

Vugs e fraturas podem modificar drasticamente as características físicas de um meio e, portanto, devem ser levados em conta de forma precisa no modelo de fluxo e transporte. Para a modelagem e simulação deste tipo de problema, utiliza-se comumente a formulação de Stokes-Brinkman, dada por (Lingaarden, 2010),

$$\nabla p = -\mu \boldsymbol{K}^{-1} v + \nabla . \mu^* (\nabla v + \nabla v^T), \qquad \nabla . v = 0 \qquad (1)$$

onde p é a pressão do fluido, $\boldsymbol{v}$ é a velocidade do fluido, $\boldsymbol{K}$ é o tensor de permeabilidade e μ e μ^* são a viscosidade, e a viscosidade efetiva do fluido, respectivamente. A viscosidade efetiva é dependente da geometria do meio, e é diferente da viscosidade, mas é uma prática comum adotar $\mu^* = \mu$ em casos altamente porosos (Nield, 1992). Nesta formulação, se assumirmos permeabilidade infinita obtemos a lei de Stokes, e se assumirmos $\mu^* = 0$ obtemos a lei de Darcy.

Este trabalho possui foco na geração de geometrias e malhas em duas dimensões que modelam problemas em meios cársticos. Vugs são modelados como elipses, que podem ou não possuir interseções, e fraturas são modeladas como quadrilaterais que conectam dois vugs. Desta forma, uma rede não necessariamente totalmente conexa de vugs de formato genérico é criada.

Recursos de multi-escala também são levados em conta. Para isto, uma malha estruturada quadrilateral de larga escala é gerada, dividindo o domínio original em diversas "regiões". Vugs podem ou não estar contidos em várias destas "regiões", e os parâmetros que definem a quantidade de "regiões" e o tamanho dos vugs podem ser ajustados a partir de informações geoestatísticas, de forma a modelar o problema de forma realista, além de permitir também a geração de malhas simplificadas para testes locais de formulação.

Neste trabalho, a seção 2 descreve as estruturas de dados utilizadas para a estruturação do nosso software, e as bibliotecas de código aberto que implementam estas estruturas de dados. Na seção 3, os algoritmos desenvolvidos para a geração das geometrias são apresentados. Na seção 4, alguns modelos e malhas são gerados para casos hipotéticos, procurando demonstrar a flexibilidade e robustez do sistema.

2 ESTRUTURAS DE DADOS

Este trabalho depende fortemente das estruturas de dados e algoritmos associados às bibliotecas utilizadas. Estas bibliotecas serão apresentadas em mais detalhe a seguir.

Uma estrutura de dados consiste em uma forma abstrata de se representar a memória do computador, de forma que entre a memória e o programador exista uma camada de interface que permite o acesso à memória sem existir a necessidade de se programar diretamente nela. Apesar desta abstração existir, é geralmente requerido que não haja perda significativa de desempenho ou generalidade. Desta forma, busca-se geralmente a implementação de estruturas de dados inteligentes, que modelem bem um dado problema computacional, mas que não afetem o desempenho e nem a capacidade do programa em questão de resolver problemas específicos.

Um algoritmo consiste então de uma série finita de passos, que podem ou não operar sob ou utilizar uma estrutura de dados, com o objetivo de se resolver um dado problema.

2.1 Conceitos de C++

C++ é uma linguagem multi-paradigma criada por Bjarne Stroustrup (Stroustrup, 2013). Neste trabalho, o foco é na linguagem C++ como ferramenta genérica e de alto desempenho.

A linguagem oferece uma gama de estruturas de dados e algoritmos através de sua biblioteca padrão, a STL (Standard Template Library). Esta é uma biblioteca genérica, programada utilizando os conceitos de classe e templates, que são oferecidos pela linguagem.

Classes são estruturas utilizadas para abstração de entidades de programação com características similares. São formadas originalmente por um conjunto de variáveis internas e um conjunto de funções que operam sobre estas variáveis. Templates são métodos poderosos de fornecer polimorfismo (a capacidade de uma classe de se comportar como outra) em tempo de compilação. Um exemplo clássico do uso dos templates seria na classe *list*, que existe na STL. Esta classe permite a criação de uma lista de qualquer tipo ou classe instanciável, bastando passar este tipo ou classe como argumento de template, na forma *list<class>*. Desta forma, basta escrever o código da classe *list* uma vez para poder usá-la em diversos contextos diferentes, com tipos ou classes diferentes.

2.2 OpenCascade

O OpenCascade (OpenCascade, 2014) é uma ferramenta de desenvolvimento de software voltada para CAD (C*omputer Aided Design,* ou Desenho Assistido por Computador). Esta biblioteca inclui uma série de algoritmos numéricos para a operação em formas abstratas de geometria, permitindo a geração de geometrias complexas.

No presente trabalho, as estruturas internas do OpenCascade foram utilizadas como representações da geometria que está sendo gerada, e suas funções internas foram utilizadas para operar nas partes desta geometria.

As principais estruturas utilizadas foram as que compõem o pacote TopoDS, que trata das estruturas de dados de descrição das formas geométricas. Dentre estas estruturas de dados, as mais usadas foram a *TopoDS_Shape*, *TopoDS_Face*, *TopoDS_Edge*, *TopoDS_Wire* e *TopoDS_Vertex*, que foram utilizadas desde a geração do domínio inicial, até a geração das cavernas, *vugs* e fraturas. Estas estruturas de dados são ilustradas na figura 1 e na figura 2, adaptadas do manual de usuário Modeling Data, do OpenCascade. Na figura 1, ilustramos uma geometria do tipo *TopoDS_Shape*, que por sua vez é composta por duas

TopoDS_Face e assim por diante. Esta hierarquia é a estrutura de dados final, ilustrada na figura 2.

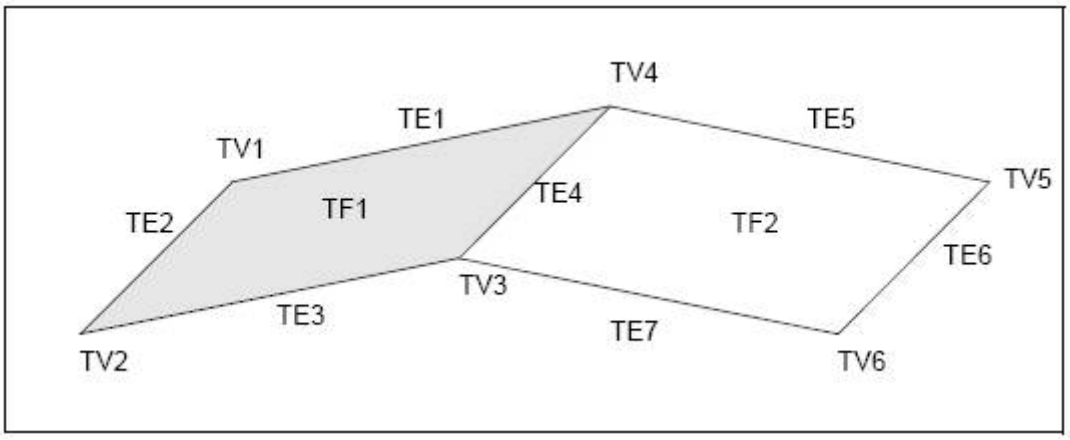

Figura 1. Estrutura de uma TopoDS_Shape

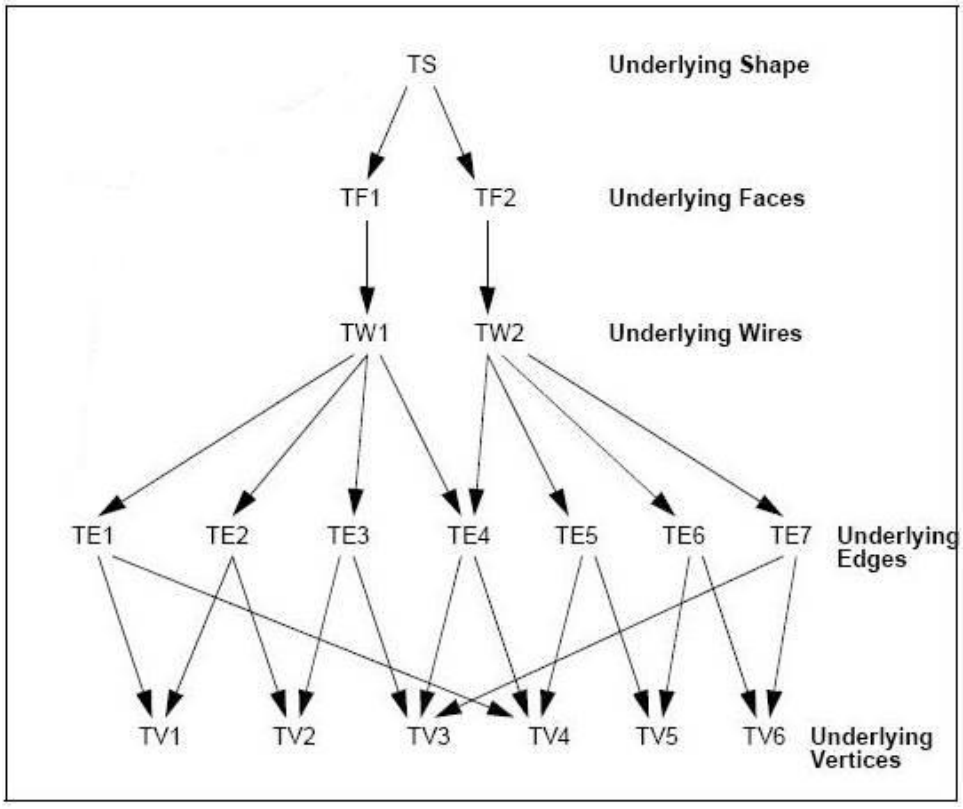

Figura 2 - Estrutura de dados resultante.

Os principais algoritmos utilizados são as Operações Booleanas, especificamente Cut e Fuse, que são mostradas na figura 3. Esta figura ilustra a geração de uma nova forma C utilizando as operações Cut e Fuse dadas duas formas arbitrárias A e B. Outros algoritmos muito utilizados são do pacote BRepBuilder. Este pacote é fundamental na criação de formas, permitindo a criação de formas Wire a partir de diversas formas Edge, ou a utilização do algoritmo de Sewing, que permite a junção de diversas formas diferentes mas que possuem sobreposto um Wire (2D) ou um Shell (3D) entre si.

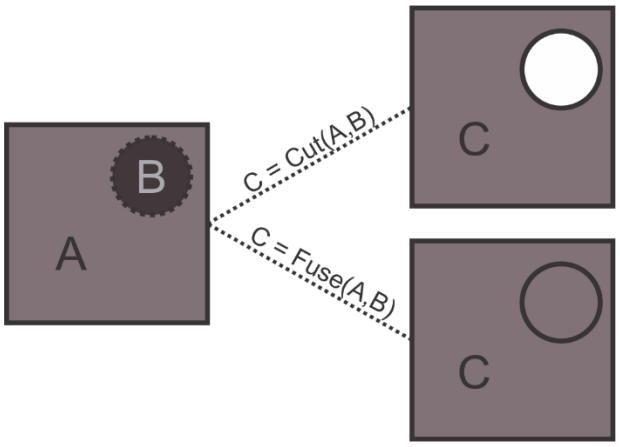

Figura 3 – Ilustração dos algoritmos Cut e Fuse.

2.3 Gmsh

O Gmsh é um gerador de malhas para elementos finitos (Geuzaine, 2009). Esta ferramenta permite a geração de uma malha a partir de uma geometria, que no caso deste trabalho é gerada com o OpenCascade. O Gmsh permite também a definição da geometria utilizando suas próprias funcionalidades CAD, porém este conjunto de funcionalidades foi considerado impróprio para os objetivos deste trabalho.

É possível utilizar uma interface do Gmsh diretamente no C++, e com isto foi possível fazer a geração de malha *in loco* neste trabalho, sem a necessidade de comunicação através de arquivos.

A utilização do Gmsh neste trabalho é bastante simples, e resume-se basicamente a:

1. Inicializar o Gmsh;
2. Importar a forma final da geometria gerada pelo OpenCascade, passando-a como referência para a função importOCCShape;
3. Iterar por todos os pontos da geometria, atribuindo a estes um tamanho característico referente ao tamanho das arestas da malha que serão geradas nas proximidades daquele ponto;
4. Exportar a malha para o formato de saída, nesse caso em arquivo.

3 ALGORITMOS DESENVOLVIDOS

Esta ferramenta de geração foi feita com o objetivo de obedecer alguns padrões de geometria encontrados em meios cársticos. O mais importante destes padrões, é o da existência de cavernas, ou *vugs* no interior dos meios. Estes são genericamente representados por elipses, em 2-D, uma vez que não se dispõe da geometria real dos mesmos. Alguns *vugs* são ilustrados na figura 4. Outro padrão é o da existência de fraturas entre estes *vugs*. Estas fraturas, neste trabalho, são aproximadas por espécies de canaletas retangulares, também para o caso 2-D, que conectam dois *vugs*, ilustradas na figura 5.

Além destes padrões, esta ferramenta visa a simulação também em multi-escala. Desta forma, tem-se uma malha estruturada quadrilateral na macro-escala, onde cada elemento (ou subregião) da mesma pode ou não conter *vugs*. Neste trabalho, *vugs* e

fraturas podem atravessar elementos da malha grossa (multi-escala). Uma malha contendo suas subdivisões de multi-escala e alguns *vugs* com fraturas é mostrada na figura 6.

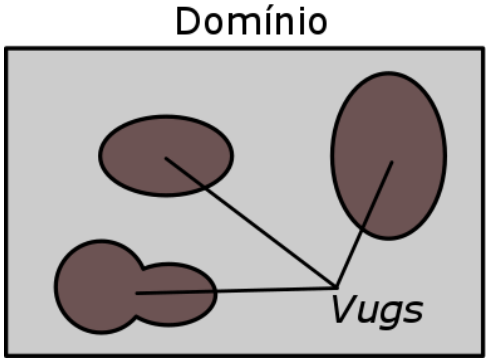

Figura 4 – Ilustração de possíveis formas de *vugs*.

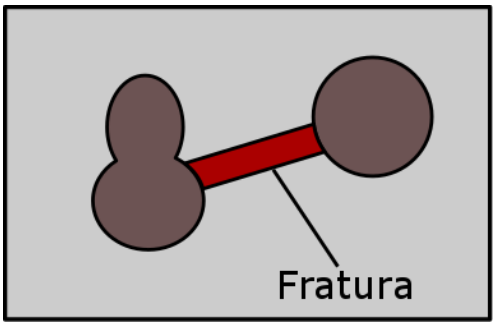

Figura 5 – Ilustração de uma fratura.

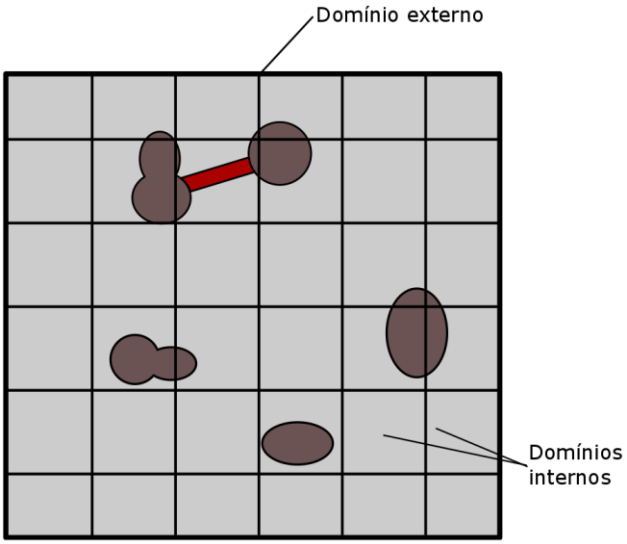

Figura 6 – Representação da multi-escala.

Dados estes requerimentos, a estruturação para a ferramenta se dá através de uma única classe de interface com o OpenCascade e com o Gmsh. Esta classe deve ser capaz de, dados parâmetros de entrada, gerar um domínio, criar *vugs* e fraturas neste domínio, separar este domínio em subdomínios para a multi-escala e exportar para o formato de

malha adotado, Gmsh. Este procedimento está descrito na figura 7, no algoritmo em pseudocódigo Procedimento Geral.

```
Procedimento Geral:
1. D ← Domínio
2. S, S' ← Cria_Vugs()
3. S ← Cria_Fraturas(D, S)
4. Insere_Shapes(S, S', D)
5. Divide_Domínio(D)
6. Exporta_Para_Gmsh(D)
```

Figura 7 - Macro-algoritmo que ilustra o funcionamento geral do sistema de geração de malhas e geometrias de multi-física e multi-escala.

No passo 1 do algoritmo ilustrado na figura 7, a geometria do domínio é criada. É nesta etapa que o programa lê um arquivo de entrada que refere-se aos parâmetros de dimensionamento do domínio, tolerâncias, número de divisões do domínio externo em subregiões (referente à malha em macro-escala), tamanho dos elementos da malha fina (i.e. na micro-escala), e distribuições estatísticas de tamanho e posição de *vugs* e fraturas.

Nota-se que nos passos 2, 3 e 4 utiliza-se duas variáveis introduzidas S e S', cujos conteúdos são primeiro obtidos no passo 2, sendo S em seguida modificado no passo 3. Estas variáveis deverão armazenar cada uma, uma lista de *shapes. Shapes* neste contexto podem ser entendidas como abstrações das estruturas de dados do pacote TopoDS. O passo 4 trata de inserir todas estas *shapes* no domínio D, através da operação booleana Fuse, afim de manter os contornos originais de todas as *shapes*. A lista de S' irá conter resultantes de dois ou mais elementos de S que foram sobrepostos, ou seja, que possuem interseção. Portanto, deve-se tomar o devido cuidado de verificar se um elemento de S não foi anteriormente mesclado com outros e inserido em S', antes de inserí-lo no domínio, para evitar duplicações.

Após a criação e inserção de *vugs* e fraturas, o particionamento do domínio em elementos da macro-escala é feito no passo 5. Este particionamento obedece os parâmetros de entrada, que informam a quantidade de partições que serão feitas.

No passo 6 utiliza-se o método importOCCShape, já presente na *API* (*Application Programming Interface*) do Gmsh. Este método recebe como parâmetro um TopoDS_Shape do OpenCascade, e o converte para a estrutura de dados do Gmsh. O domínio resultante, após todas as inserções de faces e as subdivisões, será do tipo TopoDS_Shape.

Os passos 2 e 3 necessitam de um detalhamento mais profundo. A figura 8 apresenta o algoritmo para a geração de *vugs* isolados ou com formas resultantes da mesclagem de dois ou mais *vugs*.

Cria_Vugs(D):

1. N ← Quantidade esperada de vugs
2. Enquanto a quantidade de vugs a serem inseridos na geometria for menor que N:
    1. V ← Elipse ou círculo conforme com os parâmetros de entrada
    2. Insere V na lista S de *shapes*
    3. S_i ← Lista com todos os shapes que possuem interseção com V, incluindo V
    4. Se S_i não possuir apenas V:
        1. Para cada *shape* V_j em S_i:
            1. Verifique se V_j possui interseções que não estão em S_i
            2. Caso existam, adicione-as recursivamente em S_i, verificando também suas interseções, e assim por diante
        2. Faça a mesclagem do conteúdo de S_i e insira este resultado na lista S'
        3. Verifique em S' se existe alguma outra mesclagem resultante que inclua um ou mais elementos de S_i
        4. Caso exista, remova estas outras mesclagens da lista S'
3. Retorne S e S'

Figura 8 - Algoritmo para a criação de vugs

No passo 1 do algoritmo ilustrado na figura 8, a quantidade estatisticamente esperada de vugs no domínio inteiro é obtida. Em seguida, no passo 2, os vugs são aleatoriamente distribuídos no domínio, obedecendo esta quantidade esperada.

É importante ressaltar que o resultado gerado pela mesclagem de dois *shapes* no passo 2.4.2 não se trata de uma simples operação booleana do tipo Fuse. Além desta operação, todas as arestas internas que aparecem após esta operação são removidas. A figura 9a ilustra que o resultado de uma operação Fuse com duas faces seria um composto de 3 faces, caso elas não sejam tangentes ou totalmente contidas entre si. Ao invés disto, obtemos uma única face, como mostrado na figura 9b.

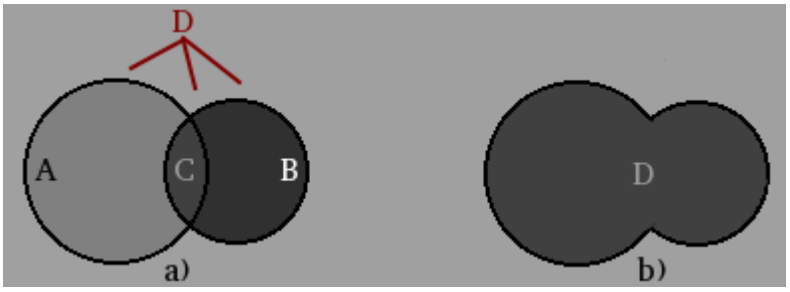

Figura 9 - (a) Resultado de uma operação Fuse. (b) Resultado de uma operação de mesclagem.

Cria_Fraturas(D, S):

1. Repita até a quantidade de fraturas inseridas respeitar os parâmetros de entrada:
    1. S_1, S_2 ← Selecione da lista S dois *shapes* que representem dois vugs distintos, sem interseção, e que não já foram escolhidos juntos antes
    2. P_1, P_2 ← Pontos centrais de S_1 e S_2
    3. L ← $\sqrt{(\boldsymbol{P_{1x}} - \boldsymbol{P_{2x}})^2 + (\boldsymbol{P_{1y}} - \boldsymbol{P_{2y}})^2}$
    4. H ← $\frac{\boldsymbol{0.1}}{\boldsymbol{L}}$
    5. S_q ← $\sqrt{\frac{(\boldsymbol{P_{1y}} - \boldsymbol{P_{2y}})^2}{(\boldsymbol{P_{1x}} - \boldsymbol{P_{2x}})^2} + \boldsymbol{1}}$
    6. dx ← $\frac{\boldsymbol{H}(\boldsymbol{P_{1y}} - \boldsymbol{P_{2y}})}{\boldsymbol{S_q}(\boldsymbol{P_{1x}} - \boldsymbol{P_{2x}})}$
    7. dy ← $\frac{-\boldsymbol{H}}{\boldsymbol{S_q}}$
    8. P_{01} ← Ponto com cordenadas X = P_{1x}-dx e Y = P_{1y}-dy
    9. P_{02} ← Ponto com cordenadas X = P_{2x}-dx e Y = P_{2x}-dy
    10. P_{03} ← Ponto com cordenadas X = P_{1x}+dx e Y = P_{1y}+dy
    11. P_{04} ← Ponto com cordenadas X = P_{2x}+dx e Y = P_{2x}+dy
    12. F ← Face formada pelas arestas P_{01}-P_{02}, P_{02}-P_{04}, P_{04}-P_{03}, e P_{03}-P_{01}
    13. Se F não possuir interseção com outros elementos da lista S:
        1. F_r ← Resultado da operação booleana Cut, utilizando S_1 e S_2 para cortar F
        2. Insira F_r em S
2. Retorne S

Figura 10 - Algoritmo para a criação de fraturas entre *vugs*.

O algoritmo Cria_Fraturas, ilustrado na Figura 10 é responsável pelo cálculo da posição, comprimento e direção das fraturas. Utilizando a simplificação de que cada ponto central das menores arestas do retângulo que modela a fratura situa-se cada um no ponto central de S_1 e S_2, ou seja, que a direção do retângulo será de um centro a outro de S_1 e S_2, temos que o comprimento L deste retângulo será facilmente calculado no passo 3 como a distância dos centros de S_1 e S_2. Adotamos empiricamente uma altura H para este retângulo como sendo 0.1/L, calculada no passo 4.

Os passos 5 a 7 calculam o incremento em x e y a partir dos pontos centrais de S_1 e S_2 que devemos utilizar para calcular os quatro pontos que irão compor a fratura, o que é feito nos passos 8 a 11. A face F referente ao retângulo que modela a fratura é criada no passo 12.

No passo 13, verifica-se se F não possui interseção com outros elementos da lista S, o que poderia ocorrer por generalização, e não há nada na ferramenta de modelagem utilizada que impeça isto de ocorrer, mas por simplicidade preferiu-se que tais casos não existam. Caso não haja interseção, as áreas de F que intersectavam S_1 e S_2 são removidas utilizando a operação *Cut* como ilustrado na figura 11, e inserimos a resultante em S.

Figura 11 - Dois *vugs* unidos por uma fratura (a) antes das operações Cut, (b) após as operações Cut.

4 RESULTADOS E CONCLUSÕES

Abaixo apresentamos resultados da geração de geometrias e malhas utilizando a ferramenta desenvolvida neste trabalho. Todas as geometrias são geradas utilizando parâmetros de distribuições aleatórias uniformes, para fins de testes. Estas distribuições podem facilmente ser substituídas, caso se deseje utilizar esta ferramenta em conjunto com programas para simulação multi-escala e multi-física de reservatórios cársticos. Neste caso, as distribuições virão da geoestatística.

O resultado gerado na Figura 12 utiliza uma geometria de altura e largura 10, com 5 subdivisões horizontais e verticais para a macro-escala. A figura 12a apresenta apenas a geometria e a malha grossa, enquanto a figura 12b mostra a geometria em conjunto com a malha final gerada, com tamanho prescrito de arestas igual a 0.1. Neste exemplo, a distribuição de vugs prevê que a quantidade média de vugs por domínio seja igual a 1.

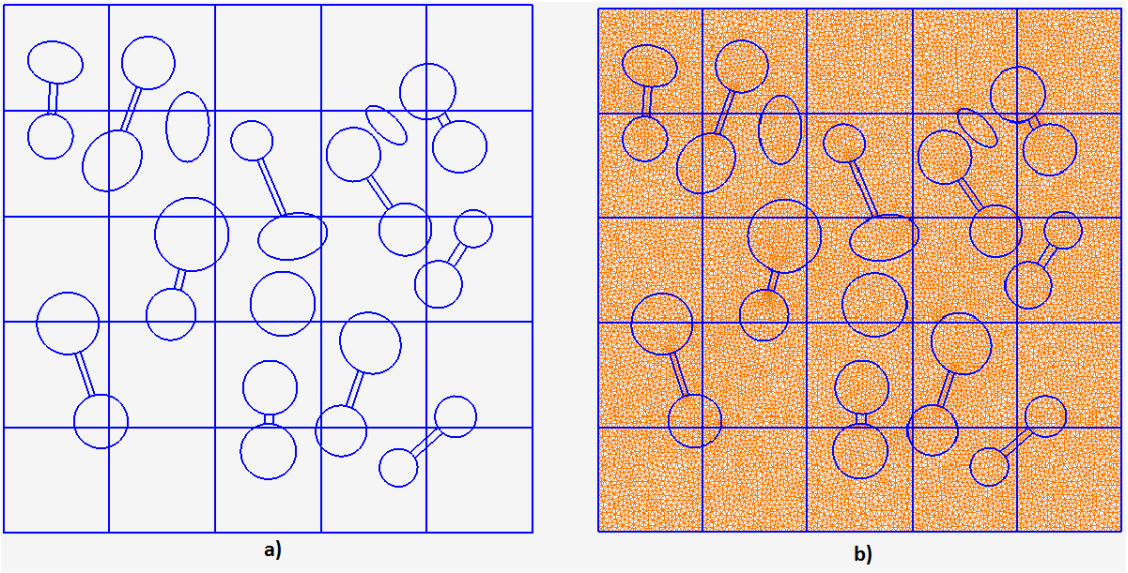

Figura 12 - (a) Geometria e malha grossa gerada. (b) Geometria, malha grossa e malha fina.

Dada a comprovação da capacidade da ferramenta de gerar malhas e geometrias, parte-e então para os testes de robustez. Para isso, parâmetros são variados, buscando-se chegar no limite da ferramenta. Dois testes de robustez são mostrados na figura 13, onde os parâmetros de altura do domínio, e a quantidade de subdomínios, foram modificados. Além disto, a probabilidade de dois *vugs* se interseccionares foi também aumentada.

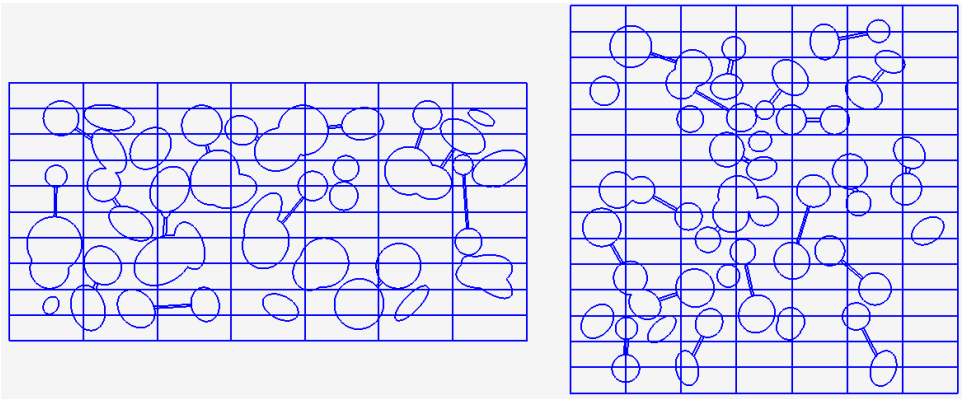

Figura 13 - Geometrias geradas variando parâmetros de altura e quantidade de subdomínios.

A figura 14 mostra um teste de *stress*, onde o tamanho do domínio, a quantidade de subdivisões, o desvio padrão do tamanho dos *vugs* e a chance de um *vug* ser uma elipse são aumentados. Na figura 14a, mostra-se a visão geral da geometria, onde a região indicada pelo retângulo vermelho é ampliada na figura 14b.

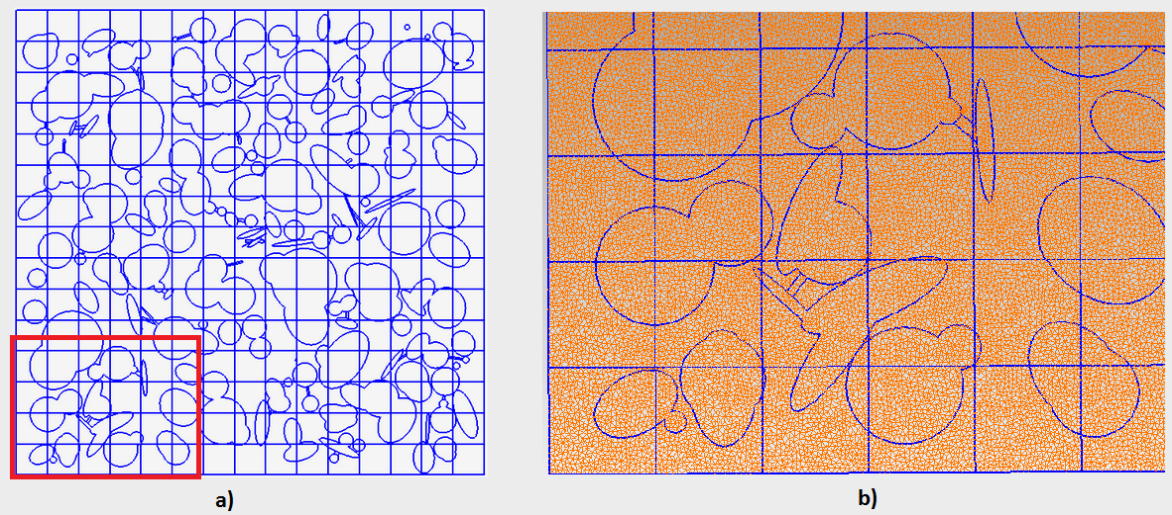

Figura 14 - (a) visão geral da geometria, (b) zoom da malha.

Mesmo o teste de *stress* foi executado com êxito, o que comprova que a ferramenta possui robustez. Além disto, constata-se que a sua flexibilidade permite que seja usada em grande escala para a modelagem e simulação multi-escala e multi-física bidimensional de reservatórios cársticos.

Como trabalhos futuros, esta ferramenta será extendida para três dimensões, e será utilizada diretamente em simuladores que utilizam o método dos elementos finitos.

AGRADECIMENTOS

Os autores gostariam de agradecer ao CNPq e CENPES-PETROBRAS pelo apoio financeiro para o desenvolvimento da presente pesquisa.

REFERÊNCIAS

Geuzaine, C., Remacle, J.-F.: *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.* International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

Gulbransen, A. F., Hauge, V. L., Lie, K.-A.: *A Multiscale Mixed Finite-Element Method for Vuggy and Naturally Fractured Reservoirs*, Society of Petroleum Engineers, 2010.

Lingaarden, I. S., Krotkiewski, M., Lie, K.-A., Pal, M., Schmid, D. W.: *On the Stokes-Brinkmann Equations for Modeling Flow in Carbonate Reservoirs*, ECMOR XII - 12th European Conference on the Mathematics of Oil Recovery, 2010.

Nield, D. A., Bejan, A.: *Convection in Porous Media*. Springer, 1992.

OpenCascade, Open-Source Toolkit for 3D modeling. *URL: http://www.opencascade.com*, visitado em 2014.

Popov, P., Qin, G., Bi, L., Efendiev, Y., Kang, Z., Li, J.: *Multi-Physics and Multi-Scale Methods for Modeling Fluid Flow Through Naturally Fractured Carbonate Karst Reservoirs*. Society of Petroleum Engineers, 2007.

Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley, 4th ed., 2013.