UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

LABORATÓRIO DE SIMULAÇÃO NUMÉRICA EM MECÂNICA DOS FLUIDOS E TRANSFERÊNCIA DE CALOR - SINMEC

RELATÓRIO FINAL

DESENVOLVIMENTO DE UM MÉTODO DE BUSCA E INTERPOLAÇÃO DE DADOS ENTRE MALHAS

GUSTAVO TRENTINI HAMESTER

FLORIANÓPOLIS

JANEIRO DE 2016

Sumário

Lista de Figuras

Lista de Tabelas

Símbolos e Abreviações

N_p^B	Número de pontos da malha B
N_p^A	Número de pontos da malha B
p_0^A	Ponto de índice 0 da malha A
p^B	Um ponto qualquer da malha B
p^A	Um ponto qualquer da malha A
pc_i^B	Nós no vetor de conectividade de p^B
R_0	Raio de influência inicial
R	Raio de influência
$p_{closest}^B$	Ponto da malha B mais próximo de um ponto na malha A
$p_{closestNeighbor}^B$	Nó conectado com $p_{closest}^B$ mais próximo de p^A
$p_{closestNeighbor_{i-1}}^B$	Nó conectado com $p_{closest}^B$ mais próximo de p^A anterior a $p_{closestNeighbor}^B$
$d_{A->B_i}$	Distância entre p_0^A e $p_{closestNeighbor}^B$
n_v	Número de vizinhos
$n_{v,min}$	Número mínimo de vizinhos
$n_{v,max}$	Número máximo de vizinhos
$n_{v,min}$	Número mínimo de vizinhos
w_i	Peso de um vizinho sobre o nó em questão
u_i	Valor da propriedade em um ponto vizinho
u	Valor da propriedade já interpolado
d_i	Distância o nó em questão e um dos nós da vizinhança entre malhas

$n_{v,min}$	Número mínimo de vizinhos
f_i	Valor interpolado da propriedade
f	Valor analítico da propriedade
$RMSD$	Root Mean Square Deviation
IDW	Inverse Distance Weightning

1 Introdução

Encontrado em cavidades interconectadas de rochas chamadas de rocha-reservatório, o petróleo constitui a base da cadeia energética mundial. A sua crescente taxa consumo requer que o processo de produção extraia a maior quantidade possível do reservatório. Para realizar esse trabalho de otimização fatores como pressão, temperatura, geometria do reservatório e propriedades físicas tanto do fluido quanto da rocha na qual esse está aprisionado devem ser consideradas.

Devido ao avanço da velocidade dos computadores hoje é possível realizar simulações em complexas geometrias e com interações multifísicas como reservatórios de petróleo. Uma das interações que ganha destaque é entre rocha e fluido, pois a medida que extrai-se o óleo o reservatório sofre despressurização podendo acarretar em deformações à matriz porosa alterando o perfil de escoamento.

Para a simulação da deformação do reservatório juntamente com o escoamento do óleo podem ser utilizadas diferentes malhas para representar a região fluida e a sólida, pois em alguns casos a ordem de grandeza dos fenômenos torna o uso de uma determinada malha adequada para um tipo de problema inviável para o outro. Por exemplo, enquanto o reservatório deforma-se pouco a pouco ao longo de uma grande área, o escoamento pode apresentar variações em um área relativamente menor.

Algumas propriedades, como por exemplo a pressão, possuem grande influência em ambos fenômenos o que acarreta no problema de como acoplar e transferir os resultados da parte estrutural para a porção fluida do reservatório ou o contrário.

1.1 Objetivos

Esse trabalho tem como objetivo contribuir com a simulação acoplada da análise estrutural com o escoamento de reservatórios de petróleo fornecendo uma biblioteca capaz de interpolar valores de propriedades armazenadas na malha para a região rochosa para outra malha utilizada para representar a região fluida ou vice-versa.

Para realizar esse trabalho necessita-se padronizar os dados de geometria e conectividade das diferentes malhas, construir um algoritmo de análise de proximidade entre pontos que não estão na mesma malha e implementar um método de interpolação baseado na proximidade dos pontos.

O algoritmo de busca desenvolvido será comparado com o já estabelecido na literatura K-D Tree analisando tempo de computação. Além disso será analisado a dimensão do erro devido a interpolação para diferentes números de vizinhos e densidade de pontos numa mesma região.

1.2 Organização do Trabalho

A estrutura deste trabalho está dividida em 4 capítulos, além desta introdução e as conclusões. O capítulo 2 aborda a estrutura de dados das malhas utilizadas neste trabalho e como deve ser feita a padronização. No 3 são discutidos os dois tipos de algoritmos de busca testados. Em seguida, apresenta-se o método de interpolação baseado na distância entre os pontos. No capítulo 4 aborda-se como o código está estruturado e como utilizá-lo. Os resultados comparando os algoritmos de busca e como alguns parâmetros afetam a acuracidade da interpolação são apresentados no capítulo 5. Por último, apresenta-se as conclusões desse trabalho e as sugestões para trabalhos futuros.

2 Caracterizando a Malha

Uma malha pode ser interpretada como um conjunto finito de células que busca representar uma região do espaço, tanto em duas quanto em três dimensões. Em três dimensões a região é particionada em poliédros, e por polígonos em duas dimensões. Malhas cartesianas, malhas de voronoi, malhas corner-point, malhas de triângulos, malhas tetraédricas e muitas outras acabam por pertencer a uma das duas abrangentes classificações, poliédricas ou poligonais.

A existência de uma extensa gama de diferentes tipos de malhas e diferentes métodos de

armazenamento de seus dados tornam a transferência de informações uma tarefa complexa. A maneira de descrever a conectividade dos nós, o local de valoração das propriedades e a maneira de manusear os elementos da malha são exemplos de características que exigem um algoritmo para cada tipo de combinação de malhas. Portanto para evitar esse problema tomou-se a decisão de criar uma estrutura de malha padrão com o intuito de facilitar a execução das tarefas de busca e interpolação.

A malha padronizada deve conter informações básicas como número de nós, número de conexões que cada nó possui, índice dos nós adjacentes e a posição do ponto no espaço. Desse modo, para cada tipo de malha basta extrair esse dados da maneira específica de cada malha e posteriormente operá-las no modo padronizado. Em casos de conflito entre malhas cell-centered e vertex-centered, a primeira pode ser padronizada como se os centróides fossem nós e as células com a qual essa divide uma face formam sua matriz de conectividade.

A estrutura de malha utilizada nesse trabalho foi a que está disponível na biblioteca Element-based Finite Volume Library, EFVLib, desenvolvida no SINMEC. A EFVLib é uma coleção de diversas bibliotecas as quais realizam tarefas como leitura de arquivos, construção de malhas e sistema de equações, contando ainda com diversos métodos para a solução desses sistemas. A GridLib, uma subbibliotecas, foi utilizada para gerenciamento da topologia da malha e das formas geométricas dos elementos da malha. Além dessa, a IOLib facilitou a tarefa de leitura de leitura de dados a partir de um arquivo .txt. Esse deve conter a posição geométrica de cada nó e quais são os nós que formam os elementos. Além disso a generalidade do código permite trabalhar tanto em 2D quanto em 3D [2].

3 Algoritmos de Busca

Em computação científica, os algoritmos de busca são utilizado para encontrar um item com uma propriedade específica em meio a uma coleção de outros itens. Os itens podem estar organizados em alguma estrutura de dados ou ainda necessitam ser estruturados para tornar o processo mais eficiente. No caso de não estarem organizados é necessário aplicar

um algoritmo de sorting (organização).

A tarefa de encontrar os pontos mais próximos na malha B de um ponto P na malha A pode parecer trivial pois basta mensurar a distância de todos os pontos e selecionar os mais próximos. Entretanto essa tarefa passa a se tornar custosa computacionalmente à medida que a quantidade de pontos tanto na malha A ou na B aumentam. Sendo N_p^A o número de pontos na malha A e N_p^B o número de pontos na malha B, um método de busca simples para encontrar os pontos da malha B mais próximos à pontos da malha A precisaria mensurar a distância entre N_p^A x N_p^B pontos.

Neste trabalho foram testados dois método para a construção da matriz de vizinhança, *k-dimensional tree* (KDTree) e *straight walk*. O primeiro é uma estrutura de organização de dados semelhante a uma árvore que particiona o espaço até todos os nós atingirem o caso base, onde não podem mais ser divididos. O segundo utiliza a conectividade dos nós já presente na estrutura de dados da malha para buscar quais são os pontos que compõe a vizinhança.

A vizinhança descrita deve ser encontrada para cada ponto da malha B composta por pontos da malha A que estão dentro de um raio R distante do ponto da malha B. A grande dificuldade é encontrar qual a magnitude desse raio R. No caso de a malha A ser mais densa em pontos que a malha B, o raio de vizinhança tende a ser menor do que se a malha A fosse mais esparsa que a B. Sem mencionar que a quantidade de vizinhos é fixa e o raio R varia até satisfazer a quantidade de vizinhos escolhida. Um número ilimitado de vizinhos impossibilita a otimização do raio da vizinhança portanto é necessário encontrar o número ótimo nós vizinhos. Além disso, sem a limitação do número de vizinhos podem surgir casos em que há uma acentuada alteração do perfil da propriedade sendo interpolada resulte em erros grotescos de interpolação.

Para realizar a interpolação de propriedades de uma malha para outra assumiu-se que pontos relativamente próximos no espaço devam ter valores semelhantes. Assim a influência que um ponto tem sobre outro é totalmente baseada na distância entre seus pontos vizinhos. Mas quem são os pontos mais próximos? A priori não é possível afirmar pois as malhas são

criadas separadamente e não existe matriz de conectividade entre os pontos da malha A com a malha B. Por isso é necessário a execução de um algoritmo para a construção dessa matriz de conectividade, que nada mais é do que um problema de otimização. Na figura 1 pode-se observar uma representação da conectividade a ser construída entre as malhas.

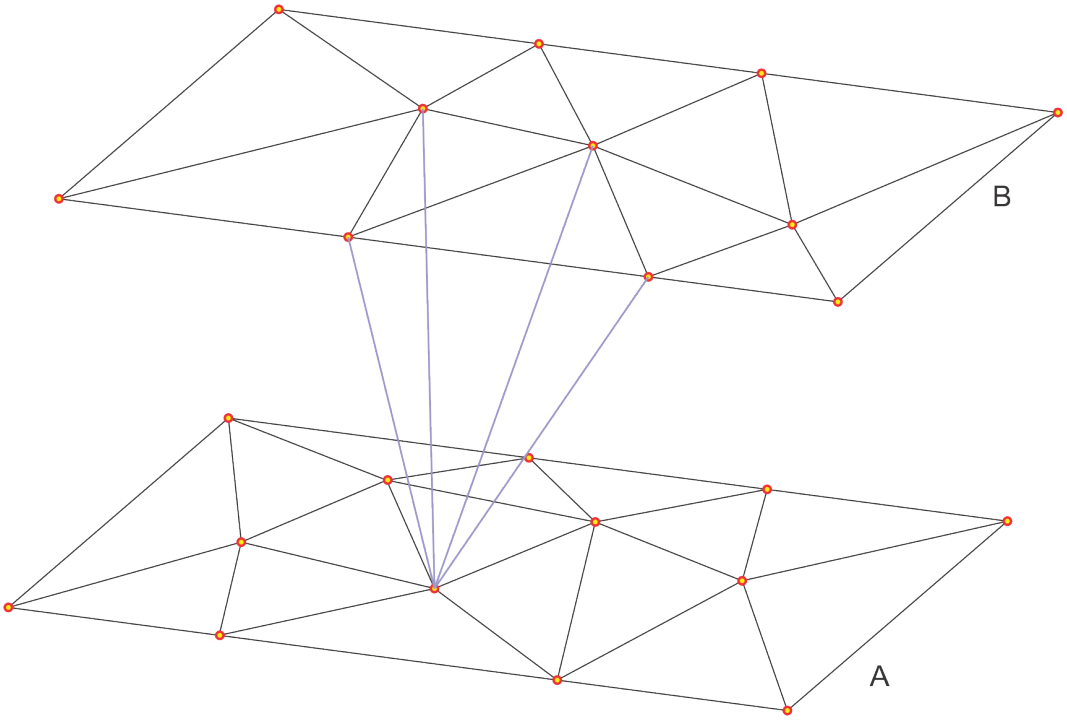

Figura 1: Representação da conectividade de um ponto P na malha A com seus vizinhos na malha B.

3.1 KDTree

O termo que dá nome a estrutura de dados é uma abreviação de árvore de k-dimensões. Assim que é contruída, a *KD Tree* faz com que operações como encontrar o vizinho mais próximo ou a coleção de vizinhos próximos tome menos tempo do que comparando um a um.

Como um dos objetivos do trabalho é comparar o método de busca desenvolvido com um outro já conhecido utilizou-se o método *KD Tree* implementado por Andrea Tagliasacchi. Esse código é hoje a base da *KD Tree* implementada no software *MATLAB* e pode ser encontrado em *Matlab Central File Exchange*. Ele é composto por três classes

MinHeap, *MaxHeap* e *KDTree* [3]. As duas primeiras são um tipo de algoritmo de *sorting* e mais detalhes podem ser encontrados em [2]. Já a última utiliza das atribuições das duas primeiras para construir a árvore, encontrar o nó mais próximo e realizar outros métodos de busca de vizinhança. Para facilitar o uso desse código pelo usuário final uma classe chamada *KD Tree Explorer* foi criada para servir como interface. A figura abaixo mostra a interação entre as classes.

O processo de construção da árvore parte da criação de uma caixa de tamanho suficiente para conter todos os pontos que serão mapeados. A lista de pontos é proveniente da lista de nós da malha. Em seguida inicia-se uma sequência de tomada de decisões que baseia-se nos seguintes princípios [1]:

1. As caixas são sucessivamente particionadas em duas caixas filhas
2. Cada partição é feita em apenas uma direção
3. As coordenadas são utilizadas alternadamente de maneira cíclica
4. Durante a partição escolhe-se a posição do corte que separe os pontos em duas partes iguais. No caso de número ímpar de pontos, a posição deve ser escolhida de maneira que fique apenas um ponto de diferença entre um lado e outro da árvore.

O próximo passo é dividir a caixa na média das coordenadas x dos pontos. O corte horizontal é feito analogamente. Em seguida, cada uma das caixas é dividida novamente até que a última caixa contenha um ou dois pontos, evitando a divisão de um caixa de dois pontos em duas caixas de um ponto o que reduz a quantidade de caixas armazenadas em algo próximo de 50%. Como a quantidade de nós é finita o código pode ser implementado recursivamente que é mais simples e sem o risco de *stack overflow*. Assim que a estrutura de dados está concluída ela pode ser usada por outras funções para encontrar vizinhanças de diferentes maneiras: número definido de vizinhos mais próximos, quais são os nós que estão dentro de um raio R distante do nó central ou quais são os nós dentro de um quadrado ao redor do ponto central.

Embora a classe *KDTree* possua muitas ferramentes ainda é necessário fazer alguns ajustes na procura pela vizinhança pois a função que faz a busca pelo raio, *ball_query*, não tem como saber a priori quantos nós vão compor a vizinhança. O ajuste está presente na classe *KDTreeExplorer*, a qual coleta os dados da malha padronizada e repassa para a *KDTree* construir a árvore. Em seguida a função *k_closest_points* é chamada para encontrar os nó da malha B mais próximos de cada nó da malha A. Sabendo quem é o nó mais próximo calcula-se a distância entre os pontos e a partir disso dá-se a primeira estimativa para o comprimento do raio da vizinhança que foi empiracamente escolhido como 50% maior que a distância ao nó mais próximo. Por último, basta executar a função *ball_query* com o raio estimado e verificar a quantidade de vizinhos dentro daquela região. Se a quantidade de vizinhos for maior que o máximo número de vizinhos o raio é reduzido por um fator de 10% e se a quantidade de vizinhos for menor que o número mínimo o raio é ampliado por um fator de 20%. A única excessão acontece para o caso dos nós possuírem exatamente a mesma posição geométrica, nesse caso só há um vizinho.

3.2 Straight Walk

A criação e implementação desse algoritmo surgiu da ideia de utilizar a estrutura de dados de malha da própria EFVLib sem a necessidade de outra estrutura como o que acontece com a KD Tree. Analisando-se as informações de malha disponíveis na EFVLib é plausível afirmar que através da matriz de conectividade dos nós pode-se encontrar toda a vizinhança tanto em malhas 2D quanto 3D. O que busca-se fazer aqui é construir um código capaz de percorrer todos os pontos da malha A e para cada ponto encontrar o correspondente mais próximo na malha B. Depois percorrer todas as conexões desse correspondente com o intuito de formar um rede de vizinhos para o ponto da malha A.

O algoritmo pode ser dividido nas seguintes etapas: montagem, encontrar o nó da malha B mais próximo de cada nó da malha A, percorrer os nós conectados com o nó da malha B e corrigir o raio de influência. Com exceção da montagem, todas as outras etapas

são realizadas em sequencia para cada nó, ou seja, a vizinhança do nó N da malha A só vai ser buscada após o nó N-1 estar com a vizinhança construída.

A etapa de montagem simplesmente inicializa as variáveis e dá um chute para um índice do nó da malha B mais próximo ao nó de índice 0 da malha A. Como a principio não se tem nenhuma informação da localização de cada nó, o chute é grosseiramente escolhido como o nó de índice igual metade do número de nós da malha.

Em seguida inicia-se a busca pela vizinhança do nó de índice 0 da malha A (p_0^A). Primeiramente, para cada nó conectado (pc_i^B) com o nó do chute inicial (p^B) verifica-se se ele já foi visitado através da *closestHashTable*. Se não foi visitado, calcula-se a distância entre p_0^A e pc_i^B. Se ela for menor que a distância entre p_0^A e p^B, pc_i^B se torna o novo p^B. Após todos os pc_i^B serem verificados dois casos podem ocorrer: nenhum pc_i^B está mais próximo de p_0^A do que p^B ou existe um pc_i^B mais próximo. No caso de existir, o algoritmo "caminha" para pc_i^B e realiza a mesma análise feita para p^B. Essa sequencia se repete até o caso de nenhum pc_i^B estar mais próximo. Vale ressaltar que devido ao uso da *hashTable* nenhum ponto é verificado mais do que uma vez mesmo que ele esteja na matriz de conectividade de pontos adjacentes garantindo uma caminhada direta até o ponto mais próximo de p_0^A apenas percorrendo a informação de conexão dos pontos.

Agora que já se conhece o ponto mais próximo de p_0^A deve-se construir a vizinhança ajustando o raio de influência a partir da quantidade de vizinhos encontrados. O raio de influência inicial, R_0, assume um valor 50% maior que a distância até o ponto mais próximo. Para poupar esforço computacional, antes de iniciar a busca da vizinhança verifica-se se a distância entre p_0^A e seu vizinho mais próximo na malha B é igual a zero, o que significa que estão na mesma posição. Para todos os outros casos o algoritmo inicia marcando que o ponto mais próximo, $p_{closest}^B$, como "já visitado". Em seguida para cada nó do vetor de conectividade de $p_{closest}^B$ confere-se se a distância entre $p_{closestNeighbor_i}^B$ e p_0^A ($d_{A->B_i}$) é menor ou igual que o raio de influência. Se estiver, duas novas checagens devem ser feitas. A primeira, se esse ainda não foi marcado no vetor de vizinhos influentes, marca-se na *closestHashTable* e adiciona-se ao vetor de vizinhos influentes. A segunda, se a distância

entre $p^B_{closestNeighbor_i}$ e p^A_0 ($d_{A->B_i}$) é menor que a distância entre $p^B_{closestNeighbor_{i-1}}$ e p^A_0 ($d_{A->B_i}$), salva-se a menor e marca-se que existe um nó mais próximo.

A condição de parada para esse *loop* está atrelada a duas exigências: não existir pontos dentro do raio de influência e o número de pontos visitados deve ser igual ao número de pontos que tiveram sua distância com relação a p^A_0 verificados (igual ao número de pontos da checados na *hashTable*). A não existência de pontos faz com que a rotina termine sem depender de outras condições.

Por último deve-se verificar se a escolher do raio de influência satisfaz as condições de número máximo e mínimo de vizinhos. Assim como para a KD Tree, se o n_v for maior que $n_{v,max}$, R é reduzido em 10% e se n_v for menor que $n_{v,min}$, R é ampliado 20%. Com o novo valor de R retornasse a etapa de busca pela vizinhança e executa-se esse processo até satisfazer as condições de número de vizinhos.

Após todas as etapas descritas nos últimos parágrafos serem executadas para cada ponto da malha A o resultado é um objeto que contem: um vetor com os vizinhos de A na malha B, um vetor de raio de influência para os pontos em A, um vetor com o número de vizinhos de cada ponto em A e uma matriz com as distâncias entre os pontos em A e seus respectivos vizinhos.

3.3 Interpolação

O método de interpolação escolhido baseia-se na hipótese de que coisas que estão mais próximas umas das outras tendem a ser mais parecidas do que coisas que estão longe e é conhecido como interpolação balanceada pelo inverso da distancia (em ingles IDW – *inverse distance weighted*). IDW utiliza de valores conhecidos dos arredores para predizer o valor em um ponto desconhecido, para isso ele assume que cada ponto conhecido exerce uma influência sobre o ponto desconhecido que diminuiu à medida que a distância aumenta. A equação 1 expressa matematicamente como o peso (w_i) é inversamente proporcional à distância elevada no expoente p [4].

$$u(x) = \begin{cases} \frac{\sum_N^{i=1} w_i u_i}{\sum_N^{i=1} w_i}, & se\ d_i \neq 0 \\ u_i, & se\ d_i = 0 \end{cases} \tag{1}$$

where

$$w_i = \frac{1}{{d_i}^p} \tag{2}$$

Analisando o gráfico da figura 2 que mostra a influência da escolha do valor de p pode-se inferir que a medida que p cresce o peso para pontos mais distantes decai rapidamente. Se p = 0, todos os pontos possuem o mesmo peso e a interpolação será um média de todos os pontos do domínio. Entretanto se p for muito grande apenas os pontos imediatamente vizinhos exercem influência.

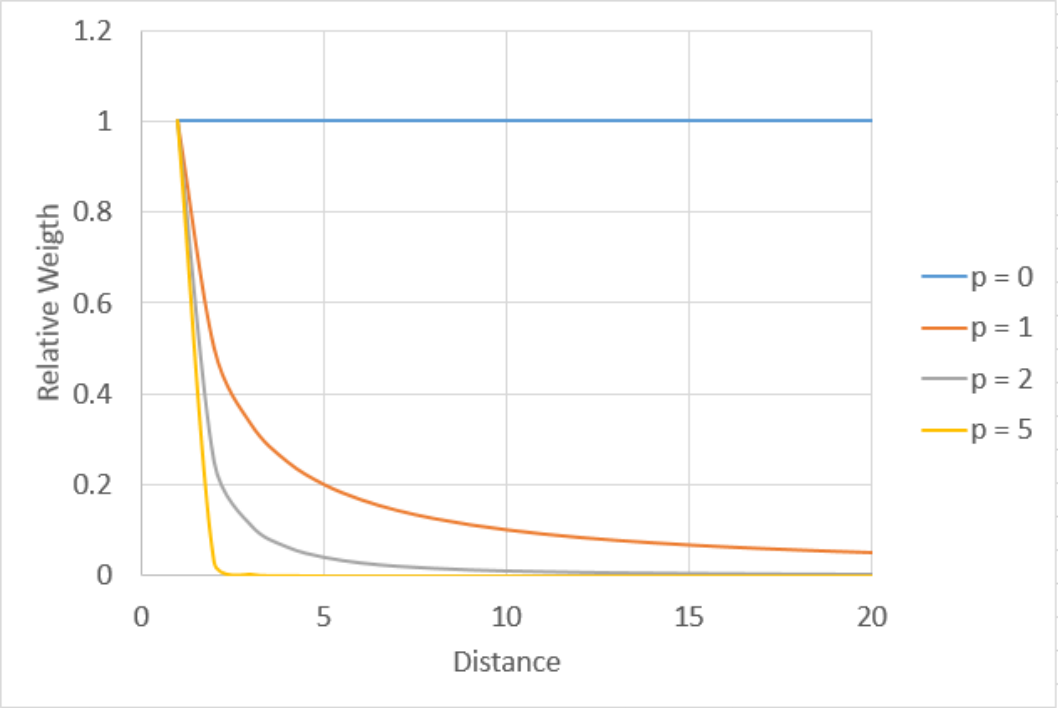

Figura 2: Influência do expoente no peso de acordo com a distância entre os nós

4 Estrutura do Código

A biblioteca *InterGridInterpolatorLib* foi implementada dentro da EFVLib 2.0 com o intuito de utilizar suas ferramentas e também estar disponível no mesmo ambiente onde é

feita a simulação. A *InterGridInterpolatorLib* pode ser divida em três núcleos: padronização da malha, procura pela vizinhança e construção da matriz de interpolação.

A padronização da malha tem como classe base a *StandardGrid* que funciona como um tipo de struct já que suas funções apenas retornam número de conexões por nó, índice dos nós conectados por nó, posição geométrica do nó e o número total de nós, tudo isso dentro da mesma malha. No estado estágio atual do código, essa classe é usada pela EbFVMToStd para montar um objeto da classe *StandardGrid* a partir de um objeto da classe *Grid*. Construir um objeto com as mesmas funcionalidades de um que já existe pode ser considerado um desperdício de memória, entretanto neste caso a padronização evita que o código de procura pela vizinhança precise ser alterado para da combinação de estrutura de malha diferente. Por exemplo, se um usuário tem interesse em interpolar dados de uma malha com estrutura diferente da EFVLib basta criar um classe que colete os dados exigidos pela *StandardGrid* e o núcleo de interpolação seguirá sem alteração.

KDTreeExplorer e *StraightWalkExplorer* são duas classes que performam a mesma tarefa de maneira diferente. As duas são responsáveis por construir um objeto da classe Neighborhood através dos dados disponíveis em um objeto da classe *StandardGrid*. A *KDTreeExplorer* utiliza das classes *KDTree* e *MyHeaps* para realizar a busca, enquanto que a *StraightWalkExplorer* possui todas suas funcionalidades consigo. As duas classes foram implementadas com valores padrões de número mínimo e máximo de vizinhos mas que podem ser alterados pelo usuário.

A Neighborhood armazena informações como: matriz de peso para interpolação, matriz de distância dos vizinhos relativa a um determinado nó, número de vizinhos influentes, índice dos vizinhos influentes e tamanho do raio de influência para cada nó. Ao contrário da *Standardgrid*, a *Neighborhood* guarda a relação de vizinhança entre nós que estão em malhas diferentes.

Todas essas funcionalidades já citadas trabalham juntas na classe *InterGridInterpolator*. Ela recebe dois objetos do tipo *StandardGrid*, uma malha com campo de valores conhecidos e outra para a qual se quer interpolar os valores. Para isso ela requer também o campo de

valores, o método de busca, o número máximo e mínimo de vizinhos e o valor do expoente p da IDW. Além de ser responsável por chamar um método de busca para construir um objeto *Neighborhood*, a *InterGridInterpolator* calcula a matriz de interpolação e armazena no mesmo objeto *Neighborhood*. Essa matriz de interpolação é a essência de tudo que foi discutido até aqui. Assim que ela é construída sabe-se exatamente qual o peso que os nós vizinhos na malha com campo conhecido exercem sobre o nó da malha com campo desconhecido. Ainda nessa mesma classe foi implementado um método para comparar o erro quando interpola-se os dados e ter uma dimensão de quão aceitável é o resultado. Para realizar a interpolação o usuário deve seguir os seguintes passos:

1. Construir dois objetos de malha. Um para o campo conhecido e outro para o desconhecido;

2. Padronizar os objetos de malha;

3. Construir um objeto ScalarField ou VectorField; Criar um objeto da InterGridInterpolator e passar pelo menos dois objetos de malhas padronizadas, um Field e o método de busca. Todas os outros parâmetros possuem valores default.

5 Resultados

Nesse capítulo serão discutidos alguns parâmetros do processo de encontrar a vizinhança e interpolar, também abrangendo como eles podem alterar o tempo computacional e a dimensão do erro. Primeiramente, *KDTree* e *StraightWalk* são comparados nas mesmas condições com o objetivo de escolher o método mais eficiente para seguir para a comparação de outros aspectos. Em seguida, verificou-se a influência do número mínimo e máximo de vizinhos. E por último analisou-se como o expoente de peso da *IDW* afeta a dimensão do erro para um determinado conjunto de malhas.

Para dimensionar o erro da interpolação considerou-se a função da equação 3 pois ela possui uma significante variação em todas as três dimensões e também possui um ponto

de cela. A diferença entre o campo interpolado e o campo analítico fornecido por essa função é definido como erro. A diferença desses dois campos também é campo escalar o que torna complicada a comparação da dimensão dos erros. Portanto, utilizou-se o *Root-Mean-Square Deviation* (RMSD) que representa o desvio padrão como parâmetro de comparação da dimensão do erro. O RMSD é definido como a raiz do somatório das diferenças ponto a ponto ao quadrado dividido pelo número de pontos e pode ser escrita matematicamente como na equação 4.

$$f = x^2 - x - y^2 + y - z^2 + z \tag{3}$$

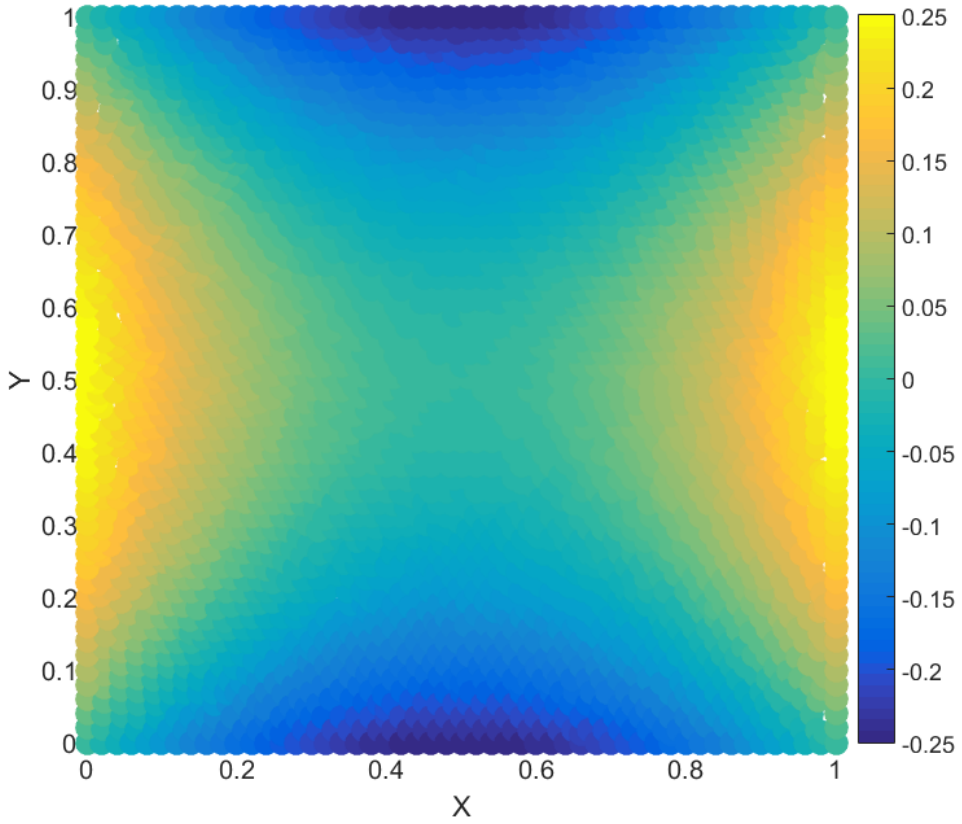

Figura 3: Mapeamento da função de referência.

$$RMSD = \sqrt{\frac{\sum_{N}^{i=1}(f_i - f)^2}{N}} \tag{4}$$

Foram geradas quatro malhas com número de nós diferentes para passar dados de uma para a outra e vice-versa. As malhas foram nomeadas de A à D de acordo com o tabela 1, onde também está descrito parâmetros da malha relevantes para esse trabalho.

Nome	Número de Nós	Número de Elementos	Tipo de Elemento
A	2943	5684	Triangular
B	16	9	Quadrangular
C	207234	206433	Quadrangular
D	1267844	2495686	Triangular
E	23248	42494	Triangular

Tabela 1: Especificações das Malhas

Como já mencionado o primeiro passo é descobrir qual método de busca é mais eficiente. Comparou-se o tempo de busca que o StraightWalk gasta para montar o objeto Neighborhood com o tempo que a KDTree leva para ser construída mais o tempo que ela gasta para montar o objeto. Embora pareça ser uma comparação injusta, foi escolhido dessa maneira pois essa é a experiência do usuário. A tabela 2 mostra os experimentos realizados para as combinações de malha A-B, A-D e C-D. Pode-se perceber o grande aumento no tempo computacional que ocorre quando o método KDTree é utilizado para uma malha mais densa em pontos como a malha D. Fica evidente que a etapa de construir uma nova estrutura de dados como a árvore é custosa pois interpolando no sentido A-D leva 11.9 s enquanto no sentido D-A o processo é 62x mais lento. Outra evidência é a pequena diferença no tempo para interpolar da malha D para a A e para interpolar da malha D para malha C. Embora a quantidade de nós entre A e C difere em duas ordens de grandeza o tempo difere em apenas 4%. Além disso, a KDTree apresenta nenhuma vantagem com relação ao desvio padrão da solução analítica quando comparado com o StraightWalk nas mesmas condições, comportamento o qual se repete para todos os outros casos apresentados na tabela 2.

Ainda na tabela 2, pode-se perceber ao observar os resultados de RMSD que quando

interpola-se de uma malha mais refinada para mais grosseira o desvio padrão é muito menor do que quando faz-se o contrário. Esse comportamento era esperado afinal a malha mais refinada possui mais informações sobre o formato do campo de escalares, e quando interpola-se da mais grosseira para mais refinada é preciso criar informações pois o campo não está tão detalhado.

Sentido		Tempo [s]		RMSD	
De	Para	KDTree	Straight Walk	KDTree	Straigth Walk
A	B	9.00E-03	1.00E-10	2.03E-03	2.03E-03
B	A	1.30E-02	7.00E-03	5.87E-02	5.87E-02
D	A	736	0.445	8.65E-05	8.65E-05
A	D	11.9	4.11	2.29E-03	2.29E-03
D	C	765	17.2	9.37E-05	9.37E-05
C	D	82.0	31.5	3.58E-04	3.58E-04

Tabela 2: Comparação entre os métodos de busca

Para exemplificar o que foi discutido acima, a figura 4 mostra um mapeamento bidimensional do resíduo quando interpola-se da malha E para malha C usando a função 3 como o campo escalar. As cores indicam quanto o resultado interpolado difere do analítico. Os nós da fronteira por estarem na mesma posição possuem valores exatos. Pelo fato da região central possuir um ponto de cela, os vizinhos tem valores muito próximos o que faz com que nessa região do mapa tenha-se uma boa interpolação. Nas demais partes onde a função apresenta mais variações percebe-se que a interpolação chega a atingir um desvio relativo em torno de 0.4% da solução analítica, o que ainda pode ser considerado aceitável.

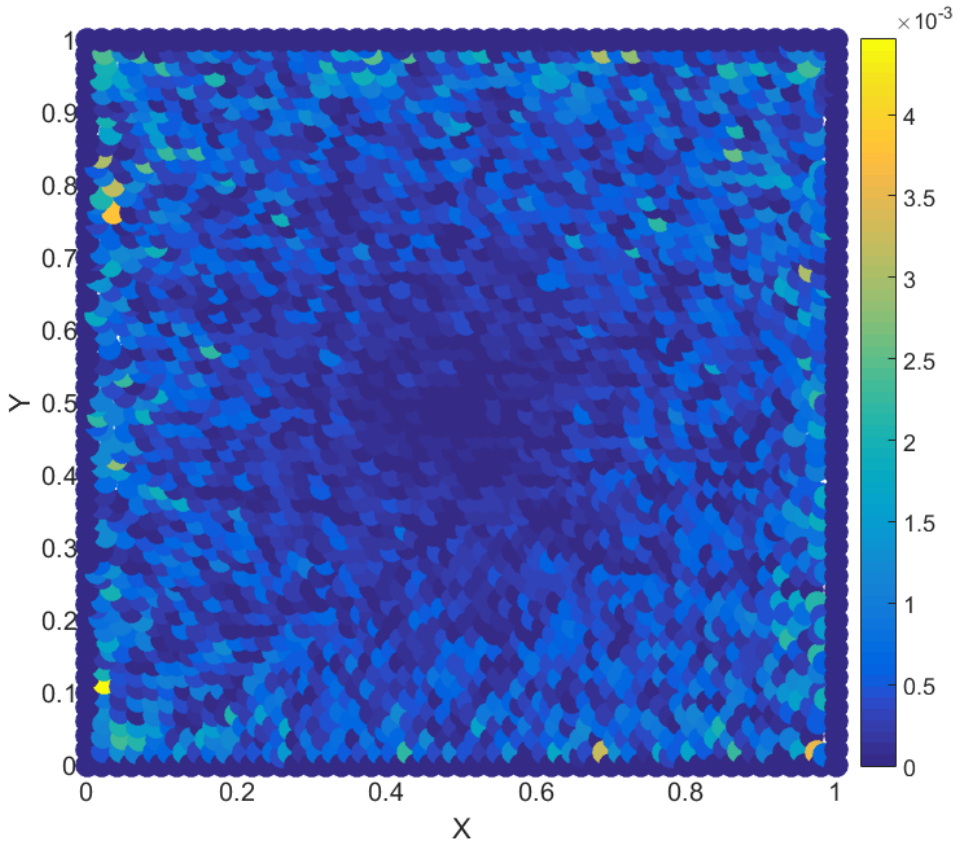

Figura 4: Mapeamento do resíduo quando o campo é interpolado da E para C.

A tabela 3 apresenta os resultados dos experimentos variando o número máximo e mínimo de vizinhos quando as interpolações foram feitas da malha D para C pelo método StraightWalk com o expoente p da IDW igual a 2.0. Percebe-se que não há um padrão definido, como por exemplo restringindo o número máximo de vizinhos o código poderia ser mais lento, por outro lado não há grande variação no tempo computacional. Já o RMSD sofre uma relativa variação, mas que não chega a uma ordem de grandeza, quando altera-se o número mínimo de vizinhos. À medida que exige-se do método de busca que ele relacione mais nós entre as malhas a qualidade da interpolação tende a decair. Realizou-se o mesmo experimento interpolando no sentido de C para D, porém os resultados foram muito semelhantes e portanto suprimidos deste trabalho. Baseado nesses resultados deduz-se que no mínimo três vizinhos e no máximo 6-10 vizinhos são escolhas plausíveis para interpolar.

N Máximo	Número Mínimo de Vizinhos					
	RMSD			Tempo Computacional [s]		
	3	4	5	3	4	5
4	9.35E-05			19.18		
5	9.65E-05	1.03E-04		17.34	16.81	
6	9.36E-05	1.02E-04	1.29E-04	16.88	16.88	15.90
7	9.37E-05	1.03E-04	1.40E-04	16.57	16.47	15.81
8	9.37E-05	1.03E-04	1.40E-04	16.51	17.19	17.49
9	9.44E-05	1.03E-04	1.40E-04	16.94	17.15	17.42
10	9.37E-05	1.03E-04	1.40E-04	16.72	15.66	16.67

Tabela 3: Experimentos variando o número máximo e mínimo de vizinhos

O expoente da IDW é um parâmetro essencial pois ele que define o quanto a distância entre os pontos influência na interpolação. A escolha de um P ideal para todas as malhas e campos escalares não é trivial e quando busca-se otimizá-lo através do desvio padrão (RMSD) a tarefa apresenta infinitas soluções. Para cada combinação de malha e campo escalar escolhido o resultado da interpolação será diferente, entretanto é possível otimizar o valor P quando trata-se uma interpolação de cada vez.

Assim que a vizinhança está construída e a matriz de distância dos vizinhos relativas a nó é conhecida, pode-se calcular a matriz de pesos para uma série de valores de P e calcular o desvio padrão para cada um dos casos. Na figura 5 apresenta-se as curvas de otimização seguindo o processo descrito acima para três diferentes combinações de malhas. Com exceção da interpolação de C para B, fica evidente a existência de um expoente P ótimo para cada situação. Além disso, o valor ótimo de P para todas as combinações acontece no intervalo de 1 a 3. Na curva de interpolação de C para B não fica claro graficamente o ponto de mínimo desvio, porém após aproximadamente 3 o desvio padrão sofre apenas variações dentro da mesma ordem de grandeza. O que explica esse comportamento é o fato de C ser uma malha mais densa que B, portanto os nós de B possuem um número

suficiente de vizinhos próximos para fornecer uma boa interpolação.

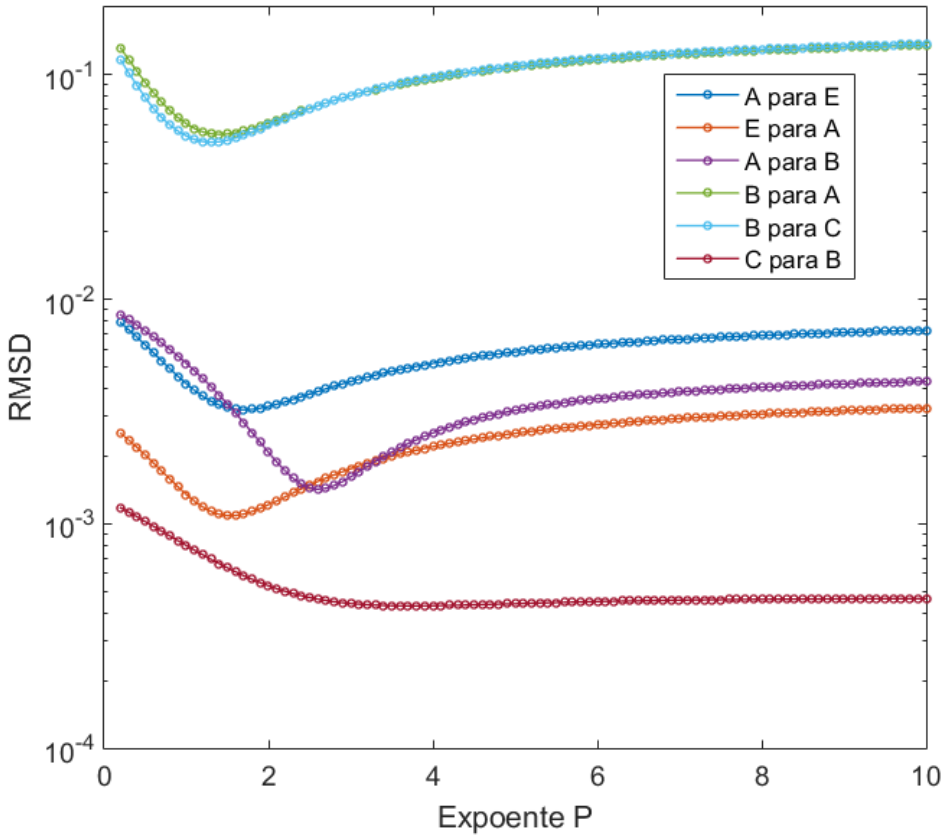

Figura 5: Experimentos variando o valor de P para diferentes combinações de malhas

6 Considerações Finais

Nesse trabalho mostrou-se como realizar a interpolação de um campo escalar entre malhas com diferentes números de nós e elementos utilizando o *Inverse Distance Weighting Method* para pesar a influência de cada nó da vizinhança sobre um ponto de interesse. A principal contribuição foi o desenvolvimento e implementação de um método para realizar a busca e construir a vizinhança entre malhas apenas utilizando os dados de conectividade disponível e sem necessitar a criação de outra estrutura de dados como o que acontece com a KDTree. O StraightWalk mostrou-se promissor com relação a sua eficiência para montar as coleções de vizinhos até mesmo para malhas com número de nós na casa dos milhões.

O código implementado está disponível na biblioteca EFVLib e funcional até a presente

data. No momento ele suporta malhas no formato EFVLib tanto em 2D quanto em 3D. Outros tipos de malhas podem ser facilmente padronizadas.

De acordo com os resultados apresentados pode-se afirmar que os parâmetros adotados como padrão fornecem resultados próximos do ideal mesmo não executando uma rotina de otimização. Os números máximo e mínimo de vizinhos definido como padrão foram dez e três, respectivamente. Já o expoente P adotou-se 2.0. A interpolação ideal utilizando todas as ferramentas apresentadas neste trabalho pode ser obtida se realizada uma rotina de otimização buscando a minimização do RMSD variando o expoente P e os parâmetros de vizinhança.

Sugestões para Trabalhos Futuros

Em trabalhos futuros pode ser explorada a possibilidade de implementar uma rotina de otimização e analisar o quão significante é a melhora da interpolação e se o custo computacional adicional compensam seu uso. Também pode-se pesquisar outros métodos de verificar a qualidade da interpolação além do RMSD.

O StraightWalk no momento não considera a direção para a qual o vetor posição entre o nó em questão e seu vizinho aponta. A escolha dos vizinhos é feita apenas pela proximidade, fazendo com que pontos em uma linha reta possam estar todos relacionados com o nó central. Se adicionada essa informação ao algoritmo pode-se buscar construir a vizinhança circundando o nó central.

Além disso, novos tipos de malhas podem ser adicionados e também pode-se construir um módulo para interpolação de campos vetoriais.

Referências

[1] D.E. Knuth. *The Art of Computer Programming.* Newly Available Sections of the Classic Work. Addison-Wesley, 2005.

[2] Silva A. F. C. Hurtado F. S. V. Pescador A. V. B. e Donatti C. N. Maliska, C. R. *Simulação de reservatórios de petróleo pelo método EbFVM com solver multigrid (SIMREP).* 2009.

[3] W.H. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, 2007.

[4] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.