Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 8, n. 1, 2021.

Trabalho apresentado no XL CNMAC, Evento Virtual - Co-organizado pela Universidade do Mato Grosso do Sul (UFMS).

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Geração de malhas com refinamento adaptativo usando tabelas de dispersão

Priscila C. Calegari[1]
DEC/UFSC, Araranguá, SC
Álvaro J. P. Franco[2]
INE/UFSC, Florianópolis, SC

Resumo. O presente trabalho apresenta uma ferramenta para a geração de malhas 2D com refinamento adaptativo e dinâmico. Uma tabela de dispersão é utilizada para o gerenciamento dos elementos da malha. Apresentamos resultados na geração de malhas adaptativas e dinâmicas para a simulação de uma instabilidade de Kelvin-Helmholtz.

Palavras-chave. Geração de malhas, refinamento adaptativo dinâmico, tabela de dispersão.

1 Introdução

O refinamento adaptativo de malhas é uma importante componente a ser considerada para a solução numérica eficiente de Equações Diferenciais Parciais (EDPs). A técnica AMR (*Adaptive Mesh Refinement*) é amplamente aplicada na simulação computacional de diversos problemas físicos. Dentre eles, problemas que envolvam combustão, separação de fases, deformação de gotas, transporte de partículas, propagação de ondas e interação fluido-estrutura [1, 3, 6, 12]. A ideia é concentrar poder computacional em regiões de interesse do domínio. A discretização do domínio computacional baseada na técnica AMR (*Adaptive Mesh Refinement*) foi proposta por Berger e Oliger [1] e trata problemas dinâmicos, com refinamento adaptativo em regiões de interesse. Com o refinamento localizado teremos menos pontos na discretização e assim, menor custo computacional. O refinamento da malha em regiões de interesse se adapta com base em critérios preestabelecidos.

Há diferentes propostas de malhas. Algumas particionam o domínio em triângulos utilizando, por exemplo, a triangulação de Delaunay ou ainda uma técnica chamada *advancing front* que triangulariza um domínio a partir da fronteira e avançando para o interior [11]. Outras particionam o domínio em malhas bloco-estruturadas compostas por quadriláteros (ou hexaedros, para o caso 3D) [3, 6, 10]. O gerenciamento do armazenamento das células de uma malha deve ser feito por alguma estrutura de dados. Podemos utilizar uma lista simples para isso. No entanto, recuperar uma célula, sem saber exatamente onde ela se encontra na lista, é uma operação cara. Se temos N células armazenadas na lista, então o tempo de execução é $O(N)$. Para este caso, inserir uma célula pode ser feita rapidamente desde que a inserção seja feita sempre no início da lista, por exemplo. Remover também pode ser feito rapidamente, se sabemos a localização da célula na lista. Outras estruturas de dados comumente utilizadas são baseadas em *árvores* tais como *quadtrees* (2D) e *octrees* (3D). Algumas ferramentas que utilizam árvores são, por exemplo, PARAMESH [10] e Afivo [14]. Outros trabalhos implementam florestas de quadtrees e octrees decompondo o domínio através

[1] priscila.calegari@ufsc.br.
[2] alvaro.junio@ufsc.br.

de coleções finitas de quadtrees (ou octrees) e de tal forma que seja possível utilizar algoritmos escaláveis para refinamento adaptativo e paralelo de malhas [2]. No entanto, vale ressaltar que as operações de inserção, remoção e recuperação dependem da altura da árvore que é $\Omega(\log_2 N)$ sendo N o número de células armazenadas na árvore. A nossa proposta é trabalhar com tabelas de dispersão, como estrutura organizadora das células da malha. Nesse caso, o *tempo esperado* para inserir, remover e recuperar células da malha é $O(1)$. Alguns trabalhos, como [7, 8], usam tabelas de dispersão implementadas por bibliotecas (STL e glib). Estes trabalhos geram malhas com balanceamento. Já a nossa proposta, implementa a própria tabela e função de dispersão, além de gerar uma malha refinada, com possibilidade de balanceamento.

Dessa forma, este trabalho apresenta uma ferramenta que gera malhas utilizando uma *tabela de dispersão* como uma estrutura de dados para gerenciar o armazenamento de suas células. Com essa estrutura, podemos *inserir*, *remover* e *recuperar* células de uma malha com refinamento adaptativo de forma eficiente. Além disso, podemos *refinar* uma célula, ou podemos *engrossar* células em uma só. Isso nos permite alterar uma malha no momento quando é determinado que tal célula deve ser refinada, ou que tais células devem ser combinadas. Com isso, uma fase realizada por algumas ferramentas de simulação numérica que remapeia uma malha do tempo t_i para uma malha no tempo t_{i+1} (veja por exemplo em [4]), não é mais necessária.

Este trabalho está organizado da seguinte forma. A Seção 2 apresenta a técnica de refinamento adaptativo de malhas. A Seção 3 apresenta as operações fundamentais definidas na malha com o uso da tabela de dispersão. A Seção 4 apresenta os experimentos numéricos. As considerações finais e trabalhos futuros são apresentados na Seção 5.

2 Geração da malha com refinamento adaptativo

Vamos considerar malhas que discretizam domínios 2D. No entanto, é possível generalizar os resultados para domínios 3D. Inicialmente, geramos uma malha uniforme, composta por células retangulares, que cobre um domínio $[a_1, b_1] \times [a_2, b_2]$, a qual chamaremos malha base. O ponto do centro de cada célula é dado por $(x_i, y_i) = (a_1 + (i + \frac{1}{2})\Delta x), a_2 + (j + \frac{1}{2})\Delta y))$, com $\Delta x = (b_1 - a_1)/N$, $\Delta y = (b_2 - a_2)/M$, e N e M são os números de células em cada uma das direções x e y.

A partir de uma malha uniforme, começamos o processo de refinamento em uma determinada região de interesse, onde cada célula selecionada é dividida e substituída por quatro novas células. O processo continua, até que seja atingido o nível de refinamento desejado, veja Figura 1. Um nível de refinamento l é composto por um conjunto de células que possuem as mesmas dimensões, Δx^l e Δy^l, sendo $\Delta x^l = 2\Delta x^{l+1}$ e $\Delta y^l = 2\Delta y^{l+1}$. A razão de refinamento é dada por $\Delta x^l / \Delta x^{l+1} = 2$ sendo l um número inteiro não negativo. Cada célula c de uma malha é identificada unicamente por três elementos $c = (i, j, l)$, onde i é o índice da coordenada x, j é o índice da coordenada y e l é o nível de refinamento de c.

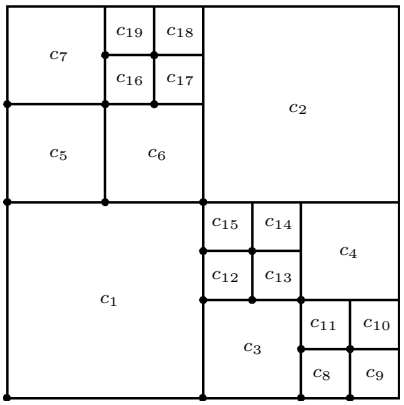

Figura 1: Malha com três níveis de refinamento.

No processo de construção da malha com refinamento adaptativo e dinâmico definimos duas operações: *refinar* e *engrossar*. Considere o seguinte cenário: em um determinado instante de tempo uma certa região do domínio pode ser considerada de menor importância para a aplicação. Dessa forma, a malha deverá ser mais grossa nessa região. Com o decorrer do tempo, tal região passa a ser importante e então deve ser refinada. Considere a Figura 1 para um exemplo, no qual

descreveremos estas duas operações. As células c_{16}, c_{17}, c_{18} e c_{19} (as quais chamaremos de células irmãs) vieram de um processo de refinamento de uma célula cuja área coincide com a área total destas quatro células. Portanto, dada uma célula $c = (i, j, l)$, sendo l menor que o nível mais fino, a operação *refinar* divide c em quatro novas células: $(2i, 2j, l+1)$, $(2i, 2j+1, l+1)$, $(2i+1, 2j+1, l+1)$ e $(2i+1, 2j, l+1)$. A célula c é removida da malha e as quatro células novas são inseridas. Como o número de inserções e remoção na tabela utilizadas por essa operação é constante, temos que o tempo de execução esperado dessa operação depende dos tempos de inserção e remoção na tabela de dispersão (que veremos ser ambos tempos esperados constantes).

A operação *engrossar* realiza o inverso da operação *refinar*, ou seja, transforma quatro células irmãs, todas pertencentes a um nível l, em uma célula no nível $l-1$, sendo $l > 0$ (0 é o nível da malha base). Essa transformação será efetivada somente se todas as células irmãs existirem na malha. Caso contrário, a operação não é realizada. Mais uma vez, temos um número constante de inserção e remoções de células na tabela para essa operação. Logo, o tempo de execução dela é esperado constante. Além disso, destacamos três pontos. Primeiro, se as coordenadas de c são inteiras então as coordenadas das células resultantes das operações refinar e engrossar também serão inteiras. A vantagem é que tal operação não está sujeita a erros numéricos. Segundo, a ferramenta permite gerar uma malha da forma como aparece na Figura 1. Para métodos onde um *balanceamento* $2:1$ é exigido, então uma operação que refina uma célula (ou que engrossa células irmãs) poderá ser propagada para células vizinhas de tal forma a garantir o balanceamento desejado. Ou seja, é possível utilizar as operações *refinar* e *engrossar* para implementar um balanceamento. O terceiro ponto está relacionado às paridades das coordenadas de células irmãs. Note, por exemplo, que uma célula inferior esquerda sempre terá índices pares; enquanto que uma célula superior direita sempre terá índices ímpares. A seguir descrevemos brevemente a estrutura de dados utilizada.

3 Tabela de dispersão

Vale ressaltar que o processo de refinamento adaptativo é dinâmico. Com isso, esperamos operações de remoções e inserções de células gerando assim novas malhas. Tais operações devem ser realizadas rapidamente. Em tabelas de dispersão, as operações de inserção, remoção e recuperação consomem tempo esperado constante. Para isso, é necessário uma função de dispersão que contribua com um número baixo de colisões de chaves. Lembrando que uma colisão ocorre quando duas chaves diferentes possuem um mesmo valor devolvido por uma função de dispersão. Em geral, há duas formas de tratar colisões. Uma utiliza o espaço da própria tabela de dispersão. A outra utiliza espaço extra à tabela, armazenando as chaves em listas encadeadas. Neste caso, cada posição p da tabela armazena uma lista de células cuja função de dispersão pode endereçar uma célula para a lista da posição p. Neste trabalho, as colisões na tabela de dispersão foram tratadas com listas encadeadas. O principal motivo desta escolha está em um resultado teórico que garante o seguinte (Teoremas 11.1 e 11.2 de [5]): *Se usamos uma função de dispersão que espalha uniformemente as células em uma tabela de dispersão que trata as colisões com listas, então uma recuperação (bem ou mal sucedida) de uma célula armazenada na tabela consome tempo* $\Theta(1+\alpha)$*, onde* α *é o fator de carga.* Portanto, se α é menor que uma constante, podemos esperar por operações eficientes.

Cada célula possui uma chave que é dada pelo seu nível e pelas coordenadas de um de seus vértices. Primeiro, uma enumeração E das células de uma malha uniforme do nível mais fino é considerada. A enumeração segue a ordem *de baixo para cima e da esquerda para a direita*, iniciando em 0. Agora vamos descrever como é calculada a chave de uma célula. Seja $c = (i, j, l)$ uma célula do nível de refinamento l e cujo par (i, j) é um índice inteiro para o seu vértice inferior esquerdo. Por exemplo, em uma malha uniforme com nível de refinamento l, os vértices inferiores esquerdos das células que tocam a fronteira inferior do domínio possuem índices $(0, 0), (1, 0), \ldots, (2^l - 1, 0)$.

DOI: 10.5540/03.2021.008.01.0421

As vizinhas acima possuem índices $(0,1),(1,1),\ldots,(2^l-1,1)$ e assim por diante. Considere o transporte da célula c para a célula c' pertencente ao nível mais fino $l^\star$ e cujos vértices inferiores esquerdos de ambas as células coincidem. A chave $K(c)$ será o valor dado para c' na enumeração E da malha uniforme do nível mais fino. Assim, uma célula $c=(i,j,l)$ onde $l \leq l^\star$, é trasportada para $c'=(2^{l^\star-l}i, 2^{l^\star-l}j, l^\star)$. Então, $K(c)=(i+jm)2^{l^\star-l}$, onde m é o número de células em cada linha da malha no nível $l^\star$.

Experimentos foram realizados utilizando três métodos de dispersão clássicos: *o método da divisão*; *o método da multiplicação*; e *o método do meio do quadrado*. O *método de Fibonacci* é um caso específico do método da multiplicação (veja [9] para detalhes de cada método). Resultados experimentais apontam para o uso do método de Fibonacci, por causar menos colisões em geral.

As operações fundamentais de tabelas de dispersão foram implementadas e estão disponíveis em `https://github.com/pccalegari/AMRHT2D`. Os detalhes dos algoritmos serão apresentados em uma versão estendida deste trabalho. Em linhas gerais, a operação de inserção de uma nova célula na tabela recebe uma nova célula c, calcula a sua chave K e armazena c na posição devolvida pela função de dispersão aplicada sobre K. A operação de remoção recebe uma célula c, calcula sua chave, utiliza uma função de dispersão para obter a posição de c na tabela e, se existir, remove c. A operação de recuperação recebe os índices e o nível de uma célula, calcula sua chave e a posição onde uma célula com tais características deveria estar e devolve tal célula, se existir.

4 Experimentos numéricos

No primeiro experimento geramos uma malha com refinamento adaptativo para a condição inicial do problema da deformação de uma gota [3]. O domínio é $\Omega=[0,1]^2$ e a condição inicial é dada por $\phi(x,y)=\tanh[75(r-d)]$, sendo $r=\frac{1}{4}$ o raio da circunferência, $d=\sqrt{(x-x_c)^2+(y-y_c)^2}$ a distância de cada ponto do domínio ao centro da circunferência $(x_c,y_c)=(\frac{1}{2},\frac{1}{2})$. A Figura 2 apresenta a função ϕ e uma região do domínio com quatro níveis de refinamento. Foram geradas malhas com nível base 32×32 células e diferentes níveis de refinamento. O critério de refinamento foi o mesmo para todos os casos. As células refinadas satisfazem a condição $-0,999+0,05\cdot l<\phi<0,999-0,05\cdot l$, sendo $l=0,\ldots,L-1$ e L o número de níveis.

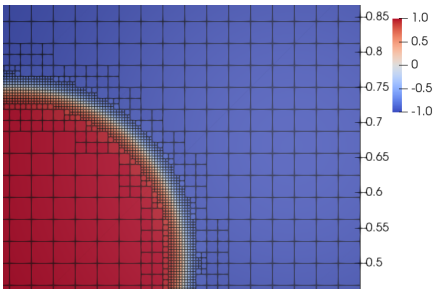

Figura 2: Função ϕ e malha com 4 níveis.

A Tabela 1 apresenta informações da tabela de dispersão: L número de níveis, N número de células da malha, P número de posições da tabela, $\alpha=N/P$ fator de carga aproximado, C e $CMAX$ são o número de colisões e o número máximo de células por posição da tabela, usando o método da divisão (MD) e o método de Fibonacci (MF), e $\lfloor \log_2(N) \rfloor$ um limitante inferior para a altura de qualquer árvore binária que armazena N células. O número de posições da tabela de dispersão é aproximadamente 10% o número de células de uma malha uniforme equivalente ao nível de refinamento mais fino. Por exemplo, com três níveis de refinamento e malha base 32×32 células, o número de células de uma malha equivalente ao nível de refinamento mais fino é $128\times 128=16.384$. Note que o fator de carga para $L3$ é maior que 1. Isso significa que existem mais células para armazenar do que posições da tabela. Podemos observar que o método de Fibonacci como função de dispersão espalhou melhor as células pois o número de colisões foi menor em todos os casos. Por último, note que o número máximo de células em uma posição da tabela sempre foi menor que a altura de qualquer árvore binária.

A Figura 3 apresenta a razão entre o número de posições vazias e o tamanho da tabela e

Tabela 1: Informações da tabela de dispersão.

L	N	P	α	C (MD)	$CMAX$	C (MF)	$CMAX$	$\lfloor \log_2(N) \rfloor$
3	2.488	1.638	1,52	1.691	7	1.081	5	11
4	5.500	6.553	0,84	1.717	4	1.499	4	12
5	15.772	26.214	0,60	6.882	5	3.763	5	13
6	51.808	104.857	0,49	16.928	4	13.022	4	15
7	179.512	419.430	0,43	63.238	5	33.751	5	17
8	635.632	1.677.721	0.38	132.876	4	84.765	4	19

a razão entre número de posições ocupadas e o tamanho da tabela para cada uma das malhas $L3, L4, \ldots, L8$. Neste experimento, a taxa de ocupação é menor que 40% a partir da malha $L6$. Além disso, a figura apresenta a razão entre a quantidade de posições da tabela com i células, para $i = 1, 2, 3, 4$, e o número de total de posições da tabela. Note que nesta razão consideramos todas as posições da tabela, inclusive as posições com listas vazias. Para o método de Fibonacci, dentre todas as malhas, o número máximo de células em uma mesma posição da tabela foi 5. A malha $L4$ possui a maior proporção de 1 célula por posição da tabela e o número de células por posição diminui rapidamente. Além disso, podemos concluir que a malha $L4$ apresentou o melhor custo/benefício com fator de carga de 84%, taxa de desocupação de 39% e de ocupação de 61%.

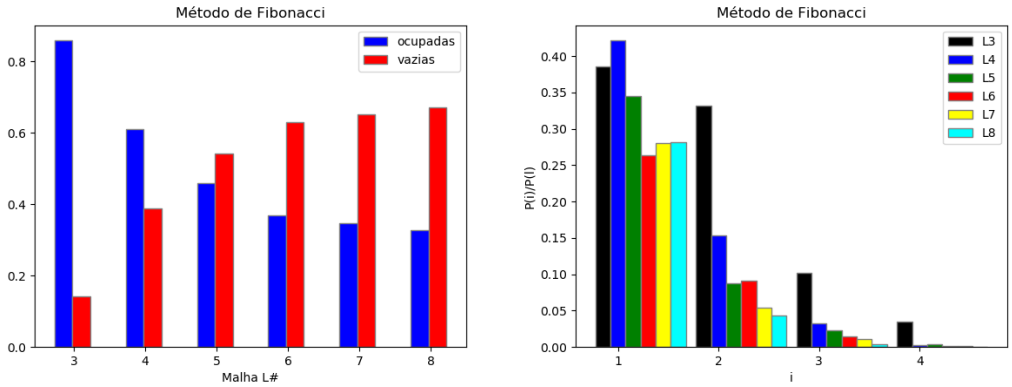

Figura 3: Taxa de ocupação e razão entre número de posições com i células e o tamanho da tabela.

No segundo experimento numérico apresentamos a geração da malha adaptativa e dinâmica em uma simulação de uma instabilidade de Kelvin-Helmholtz [6]. Para isso, incluímos o transporte de partículas lagrangianas, modelado pelo sistema de EDOs

$$\frac{d}{dt}x_p(t) = \frac{(y_p - y_c)vt(d,t)}{d}, \qquad \frac{d}{dt}y_p(t) = -\frac{(x_p - x_c)vt(d,t)}{d}, \tag{1}$$

sendo (x_p, y_p) a posição de uma partícula p, $vt(d,t) = (\beta/2\pi d)(1 - e^{-d^2/4\mu t})$, com $\beta = 500$ e $\mu = 10$ e $d = \sqrt{(x-x_c)^2 + (y-y_c)^2}$ e $(x_c, y_c) = (0,0)$. O domínio é $\Omega = [-\frac{1}{2}, \frac{1}{2}]^2$ e inicialmente 1.000 partículas estão distribuídas ao longo do eixo x (a menos de uma leve perturbação). O método de RK4 foi utilizado para obter a solução aproximada do sistema de EDOs (1). O critério de refinamento para a geração dinâmica das malhas foi a posição das partículas.

O objetivo deste experimento é apresentar as operações *engrossar* e *refinar*. Uma análise de erro não é apresentada pois a evolução das partículas não utiliza aproximações avaliadas na malha.

A Tabela 2 apresenta o tempo de CPU (em segundos) de diferentes simulações com o número de passos no tempo fixado em 300. O tempo de cada simulação que aparece na tabela é o tempo médio de 10 execuções. Comparamos os tempos das simulações que usam malhas adaptativas com malha base 16×16 com as simulações com malhas uniformes equivalentes ao nível de refinamento mais fino. Na tabela, L é o número de níveis, TR é o tempo da malha adaptativa, M é o número de células da malha uniforme e TU é o tempo da malha uniforme. Vale ressaltar que o tempo das malhas adaptativas é consideravelmente menor a medida que o número de níveis cresce. Por exemplo, o tempo de $L5$ é $26,6\%$ do tempo da respectiva malha uniforme e o de $L8$, apenas $1,11\%$.

Tabela 2: Tempo de simulação malha adaptativa × malha uniforme.

L	TR	M	TU	L	TR	M	TU
3	$0,787506$	64^2	$0,788561$	6	$1,2096589$	512^2	$12,836669$
4	$0,881960$	128^2	$1,385134$	7	$1,5908063$	1024^2	$49,205981$
5	$0,985843$	256^2	$3,703385$	8	$2,1916684$	2048^2	$196,72016$

A Figura 4 apresenta a malha, da simulação $L4$, no instante de tempo inicial e a malha gerada após 160 passos de tempo, no instante de tempo $t \approx 0.0016$[3].

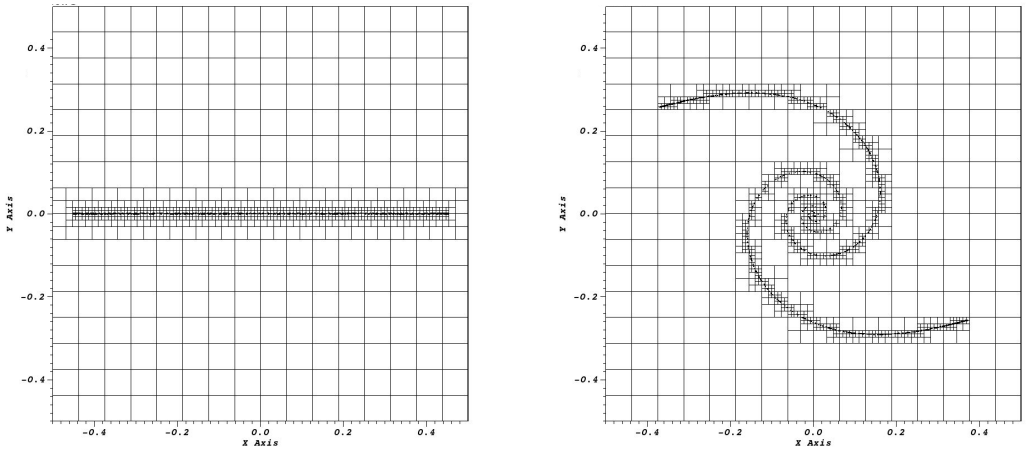

Figura 4: Malha adaptativa dinâmica em dois instantes de tempo.

5 Considerações finais

As funções de dispersão e a forma como tratamos colisões são clássicas mas possuem algumas vantagens e garantias teóricas. Vale lembrar que não houve mais que 7 células em uma posição de qualquer tabela em todos os nossos experimentos. Ainda vale dizer que a implementação desta proposta é simples e rápida. As operações *refinar* e *engrossar* permitem a modificação de uma malha de forma *online*, sem ser necessário passar por uma fase que remapeia uma malha em outra. Há funções de dispersão que são *perfeitas*, ou seja, garantem no máximo uma célula em uma posição da tabela, sem colisões. Como trabalhos futuros, vamos implementar funções de dispersão perfeitas, incluir as operações de balanceamento e uma proposta para obter células vizinhas de forma eficiente utilizando hipergrafos.

[3]As simulações foram realizadas em um computador Intel Celeron de 1.5GHz e 3.7Gb de RAM.

DOI: 10.5540/03.2021.008.01.0421

Agradecimentos

Os autores agradecem as sugestões de Alexandre M. Roma, Catalina M. R. Alvarez, Leomar M. Marschalk e dos revisores. Agradecem o apoio do CNPq (Proc. 423833/2018-9), FAPESC e VIIS-UDENAR.

Referências

[1] Berger, M. J. and Colella, P. Local adaptive mesh refinement for shock hydrodynamics, *Journal of Computational Physics*, 82(1):64–84, 1989. DOI:10.1016/0021-9991(89)90035-1.

[2] Burstedde, C., Wilcox, L. C., and Ghattas, O. (2011). p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3), 1103-1133, 2011. DOI:10.1137/100791634.

[3] Ceniceros, H. D., Nós, R. L. and Roma, A. M. Three-dimensional, fully adaptive simulations of phase-field fluid models, *Journal of Computational Physics*, 229(17):6135–6155, 2010. DOI:10.1016/j.jcp.2010.04.045.

[4] Collom, G., Redman, C. e Robey, W. Fast Mesh-to-Mesh Remaps Using Hash Algorithms, *Journal on Scientific Computing*, 40(4):450–476, 2018. DOI:10.1137/16M109140X.

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L. e Stein, C. *Introduction to Algorithms, 3a edição*. The MIT Press, 2009.

[6] Duarte, M., Descombes, S., Tenaud, C., Candel, S. and Massot, M. Time–space adaptive numerical methods for the simulation of combustion fronts, *Combustion and Flame*, 160(6):1083–1101, 2013. DOI:10.1016/j.combustflame.2013.01.013.

[7] Griebel, M. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing*, 61(2):151–179, 1998. DOI:10.1007/BF02684411.

[8] Ji, H., Lien, F. S., and Yee, E. A new adaptive mesh refinement data structure with an application to detonation, *Journal of Computational Physics*, 229: 8981–8993, 2010. DOI:10.1016/j.jcp.2010.08.023

[9] Knuth, D. E. *The art of computer programming, 3a. edição*. Pearson Education, 1997.

[10] MacNeice, P., Olson, K. M., Mobarry, C., Fainchtein, R., and Packer, C. PARAMESH: A parallel adaptive mesh refinement community toolkit, *Computer Physics Communications*, 126(3): 330–354, 2000. DOI:10.1016/S0010-4655(99)00501-9.

[11] Owen, S. J. A survey of unstructured mesh generation technology. *Proceedings of the 7th International Meshing Roundtable*. Sandia National Laboratories, 239:p.267, 1999.

[12] Roma, A. M., Peskin, C. S., and Berger, M. J. An adaptive version of the immersed boundary method, *Journal of Computational Physics*, 153(2): 509–534, 1999. DOI:10.1006/jcph.1999.6293.

[13] Szwarcfiter, J. L. e Markenzon, L. *Estruturas de Dados e seus Algoritmos, 3a edição*. Livros Técnicos e Científicos, 2010.

[14] Teunissen, J. and Ebert, U. Afivo: A framework for quadtree/octree AMR with shared-memory parallelization and geometric multigrid methods, *Computer Physics Communications*, 233:156–166, 2018. DOI:10.1016/j.cpc.2018.06.018

