



**Universidade Federal do Ceará**  
**Departamento de Computação**  
**Mestrado e Doutorado em Ciência da Computação**  
**CRAb - Computação Gráfica, Realidade Virtual e Animação**

**Markos Oliveira Freitas**

**Geração em Paralelo de Malhas Tetraédricas com Fraturas por Decomposição Espacial Binária**

**Fortaleza, Ceará**  
**2015**



Markos Oliveira Freitas

Geração em Paralelo de Malhas Tetraédricas com Fraturas por Decomposição Espacial Binária

Tese submetida à Coordenação do Curso de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Joaquim Bento Cavalcante Neto  
Universidade Federal do Ceará  
Orientador

Creto Augusto Vidal  
Universidade Federal do Ceará  
Coorientador

Francisco Heron de Carvalho Junior  
Universidade Federal do Ceará

André Maués Brabo Pereira  
Universidade Federal Fluminense

Luiz Fernando Campos Ramos Martha  
Pontifícia Universidade Católica do Rio de Janeiro

Fortaleza, Ceará  
2015

Dados Internacionais de Catalogação na Publicação  
Universidade Federal do Ceará  
Biblioteca de Ciências e Tecnologia

- 
- F937g Freitas, Markos Oliveira.  
Geração em paralelo de malhas tetraédricas com fraturas por decomposição espacial binária /  
Markos Oliveira Freitas. – 2015.  
118 f. : il. color.
- Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de  
Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2015.  
Área de Concentração: Ciência da Computação.  
Orientação: Prof. Dr. Joaquim Bento Cavalcante Neto.  
Coorientação: Prof. Dr. Creto Augusto Vidal.
1. Computação de alto desempenho. 2. Algoritmos. 3. Computação gráfica. I. Título.

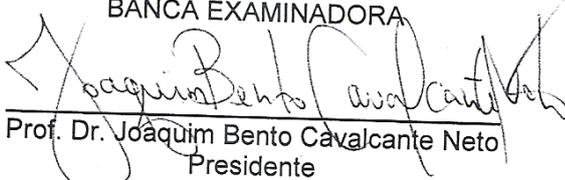
Markos Oliveira Freitas

**Geração em Paralelo de Malhas Tetraédricas com Fraturas por Decomposição Espacial Binária**

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em 08 de maio de 2015

BANCA EXAMINADORA

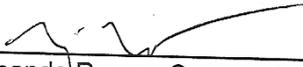
  
Prof. Dr. Joaquim Bento Cavalcante Neto  
Presidente

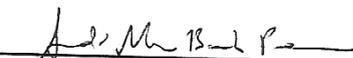
Universidade Federal do Ceará – UFC



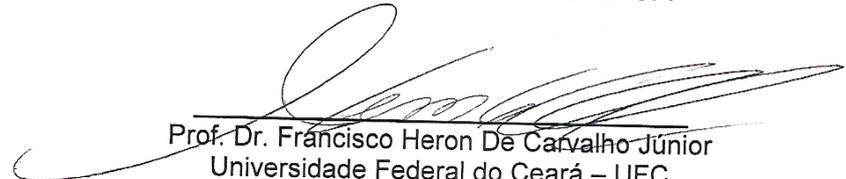
Prof. Dr. Creto Augusto Vidal  
Coorientador

Universidade Federal do Ceará – UFC

  
Prof. Dr. Luiz Fernando Ramos Campos Martha  
Pontifícia Universidade Católica do Rio de Janeiro-PUC



Prof. Dr. André Maués Brabo Pereira  
Universidade Federal Fluminense-UFF

  
Prof. Dr. Francisco Heron De Carvalho Júnior  
Universidade Federal do Ceará – UFC

Fortaleza, 08 de maio de 2015



# AGRADECIMENTOS

Agradeço a Deus, por ter-me dado a vida e a motivação necessária para seguir o meu caminho.

Aos meus pais, Maury e Elena, pelo incentivo contínuo e incondicional para a realização dos meus sonhos. Ao meu irmão, Lukas, pelos momentos de alegria proporcionados. À minha noiva, Karol, pelo apoio e carinho durante esses anos.

Agradeço aos meus professores orientadores, Bento e Creto, por terem acreditado no meu trabalho e por terem guiado minha jornada acadêmica. Aos professores e colaboradores Luiz Fernando, Bruce Carter, Wash Wawrzynek e Anthony Ingraffea, e aos membros da banca Francisco Heron e André Maués, pelas indispensáveis contribuições que aprimoraram e engrandeceram este trabalho.

Aos colegas de labuta Yuri, Ricardo, Teófilo, Martha, Rafael Ivo, Rafael Vieira, Suzana, Danilo, Caio, Laise, Lílian, Arnaldo, Tiago, Artur, Paulo Bruno, Mingal, Pablo, Henrique Bustamante e aos outros nomes que tenha esquecido, pelos momentos divertidos durante as intermináveis horas enfiadas no laboratório de pesquisa. Em especial, agradeço a Daniel Siqueira e Daniel Nascimento, pelas discussões e contribuições dadas para a realização deste trabalho. Agradeço aos amigos Rômulo, Natália, Yuri, Daniela, Paulo Jorge, Elainy, Julli, Diógenes, Karol, Bel e Cleidson pelas amizades que se formaram e que nunca acabarão. Aos amigos da Universidade de Cornell, Al, Brett, Ashley e Niko, por terem me recebido durante o meu doutorado sanduíche. Agradeço também aos amigos que compareceram à minha defesa e que ainda não foram mencionados, Antônio, Flôrencio, Ivens e Elias.

Agradeço ao Programa de Mestrado e Doutorado em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) e ao Grupo de Pesquisa em Computação Gráfica, Realidade Virtual e Animação (CRAb), pela oportunidade dada. Aos funcionários da administração do Laboratório de Pesquisa em Computação (LIA), Daniel e Vlademiro, e da secretaria do MDCC, Orley, Marcos e Leiliane, que, mesmo durante as férias, mostraram-se dispostos a ajudar sempre que preciso.

Agradeço ao Texas Advanced Computing Center (TACC), da Universidade do Texas, por ter cedido o *cluster* Stampede para a execução dos casos de teste presentes neste trabalho. À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Instituto Tecgraf de Desenvolvimento de Software Técnico-Científico (TecGraf) da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), pelo apoio dado financeiro para a realização deste trabalho.

Finalmente, agradeço a todos os anônimos que me acompanharam e me fizeram ser uma pessoa melhor.



# RESUMO

Este trabalho descreve uma técnica para gerar malhas tridimensionais tetraédricas utilizando computação paralela, com processadores de memória compartilhada, memória distribuída ou memória híbrida. A entrada para o algoritmo é uma malha triangular que modela a superfície de um ou vários objetos, que podem conter buracos no interior ou fraturas internas ou na borda. Uma estrutura em forma de árvore binária de partição espacial é proposta neste trabalho para, recursivamente, decompor o domínio em tantos subdomínios quantos forem os processos ou *threads* no sistema paralelo, em que cada subdomínio tem a geometria de um paralelepípedo retangular. Esta decomposição tenta equilibrar a quantidade de trabalho em todos os subdomínios. A quantidade de trabalho, conhecida como carga, de qualquer gerador de malha é geralmente dada em função do tamanho da saída do algoritmo, ou seja, do tamanho da malha gerada. Assim, faz-se necessária uma técnica para estimar previamente o tamanho dessa malha, que é carga total do domínio. Este trabalho faz uso de uma *octree* refinada, gerada a partir da malha de superfície dada como entrada, para estimar esta carga, e a decomposição é feita a partir dessa *octree*. Uma vez decomposto o domínio, cada processo/*thread* gera a malha em seu subdomínio por uma técnica de avanço de fronteira, de forma que ela não ultrapasse os limites definidos pelo seu subdomínio, e aplica um melhoramento nela. Alguns dos processos/*threads* ficam responsáveis por gerar as malhas conectando os subdomínios, ou seja, as malhas de interface, até que toda a malha tenha sido gerada. Esta técnica apresentou bons resultados de *speed-up*, mantendo a qualidade da malha comparável à qualidade da malha gerada sequencialmente.

**Geração em paralelo de malhas, Tetraedralização, Computação de alto desempenho.**



# ABSTRACT

This work describes a technique for generating three-dimensional tetrahedral meshes using parallel computing, with shared, distributed, or hybrid memory processors. The input for the algorithm is a triangular mesh that models the surface of one of several objects, that might have holes in its interior or internal or boundary cracks. A binary tree structure for spatial partitioning is proposed in this work to recursively decompose the domain in as many subdomains as processes or threads in the parallel system, in which every subdomain has the geometry of a rectangular parallelepiped. This decomposition attempts to balance the amount of work in all the subdomains. The amount of work, known as load, of any mesh generator is usually given as a function of its output size, i.e., the size of the generated mesh. Therefore, a technique to estimate the size of this mesh, the total load of the domain, is needed beforehand. This work uses a refined octree, generated from the surface mesh, to estimate this load, and the decomposition is performed on top of this octree. Once the domain is decomposed, each process/thread generates the mesh in its subdomain by means of an advancing front technique, in such a way that it does not overpass the limits defined by its subdomain, and applies an improvement on it. Some of the processes/threads are responsible for generating the meshes connecting the subdomains, i.e., the interface meshes, in order to generate the whole mesh. This technique presented good speed-up results, keeping the quality of the mesh comparable to the quality of the serially generated mesh.

**Parallel mesh generation, Tetrahedralization, High performance computing.**



# LISTA DE FIGURAS

Figura 1.1 – Exemplos de uso de malhas. . . . .	22
Figura 2.1 – Avanço de fronteira. . . . .	27
Figura 2.2 – Diagrama de Voronoi (em tracejado). . . . .	28
Figura 2.3 – Critério de Delaunay. . . . .	28
Figura 2.4 – Triangulação por inserção de vértices. . . . .	28
Figura 2.5 – Decomposição discreta do domínio sem agrupamento. . . . .	29
Figura 2.6 – Decomposição discreta do domínio com agrupamento. . . . .	30
Figura 2.7 – Decomposição contínua do domínio com interfaces <i>a priori</i> . . . . .	30
Figura 2.8 – Decomposição contínua do domínio com interfaces <i>a posteriori</i> . . . . .	31
Figura 2.9 – Iterações de avanço de fronteira e malha final particionada em 4 processos (DE COUGNY et al., 1996). . . . .	33
Figura 2.10–Migração de elementos do processador A para o processador B (OKUSANYA; PE-RAIRE, 1996). . . . .	34
Figura 2.11–Malha gerada pela técnica de Hodgson e Jimack (1996), já particionada. . . . .	35
Figura 2.12–Técnica de De Cougny e Shephard (1999). . . . .	35
Figura 2.13–Técnica em Topping e Cheng (1999). . . . .	36
Figura 2.14–Malha gerada pela técnica de Lämmer e Burghardt (2000). . . . .	36
Figura 2.15–Inserção paralela de um novo ponto (CHRISOCHOIDES; NAVE, 2000). . . . .	37
Figura 2.16–Entrada, malha inicial não-conforme, malha intermediária não-conforme e malha final conforme da técnica em Coupez et al. (2000). . . . .	37
Figura 2.17–Técnica de Löhner (2001). . . . .	38
Figura 2.18–Subdivisão recursiva do domínio em Larwood et al. (2003). . . . .	39
Figura 2.19–Processador 1 “rouba” um triângulo do processador 2, quando as dependências, indicadas pelas setas, causam um <i>deadlock</i> (KOHOUT et al., 2005). . . . .	39
Figura 2.20–Zona de <i>buffer</i> da célula $S_i$ , na técnica descrita em Chernikov e Chrisochoides (2005). . . . .	40
Figura 2.21–Técnica de Linardakis e Chrisochoides (2006). . . . .	41
Figura 2.22–Refinamento das arestas de interface (LINARDAKIS; CHRISOCHOIDES, 2006). . . . .	41
Figura 2.23–Caminho de propagação de arestas de $t_0$ e o refinamento empregado (RIVARA et al., 2006) . . . . .	42
Figura 2.24–Contorno e particionamento de sua malha grosseira refinada (ITO et al., 2007). . . . .	42
Figura 2.25–Formas de decomposição do domínio em Glut e Jurczyk (2008). . . . .	43
Figura 2.26–Malhas geradas particionando o domínio (ZAGARIS et al., 2009). . . . .	43
Figura 2.27–Geração de uma malha tetraédrica pelo primeiro método (YILMAZ et al., 2010). . . . .	44
Figura 2.28– <i>Kd-tree</i> para particionar os blocos (WU et al., 2011). . . . .	45
Figura 2.29–Linhas divisórias dos triângulos que são descarregados para o disco (WU et al., 2011). . . . .	46
Figura 2.30–Modificações para garantir o ângulo mínimo de 60 graus. Os dois primeiros casos (a) são resolvidos em (b), e o terceiro caso (c) é resolvido em (d) (PANITANARAK; SHONTZ, 2011). . . . .	46
Figura 2.31–Geração da malha pela técnica de Lo (2012a). . . . .	47
Figura 2.32–Decomposição do domínio e geração de malha em Löhner (2012). . . . .	48
Figura 2.33–Plano de partição, arestas da interface, malha de interface e subdomínios da técnica de Chen et al. (2012). . . . .	48

Figura 2.34–Decomposição por <i>quadtree</i> e deslocamento da <i>quadtree</i> na técnica de Freitas et al. (2013). . . . .	49
Figura 2.35–Quantidade de trabalhos por ano. . . . .	49
Figura 3.1 – Exemplo de trinca. A distância entre vértices e faces mostrada é apenas esquemática e não existe na realidade. . . . .	52
Figura 3.2 – Contorno e <i>octree</i> gerada. . . . .	53
Figura 3.2 – Contorno e <i>octree</i> gerada (continuação). . . . .	54
Figura 3.3 – Avanço de uma face da fronteira. . . . .	55
Figura 3.4 – Geração de malhas por retrocesso. . . . .	56
Figura 3.5 – Troca de faces. . . . .	57
Figura 3.6 – Troca de arestas, com três tetraedros adjacentes (caso inverso à troca de faces). . . . .	57
Figura 3.7 – Exemplo de árvore BSP criada para um domínio (em preto). Cada célula (em vermelho), interna ou folha, tem a geometria de um retângulo. . . . .	58
Figura 3.8 – Geração das malhas (em cinza) respeitando a geometria das células da BSP. . . . .	59
Figura 3.9 – Comparação entre malhas refinada e não refinada. Perceba que a mesma fronteira foi utilizada para as duas malhas. . . . .	60
Figura 3.10–Fronteira com discretização uniforme (azul - área maior, vermelha - área menor). . . . .	60
Figura 3.11–Regiões de transição. . . . .	61
Figura 3.12– <i>Octree</i> de estimativa de carga classificada. . . . .	61
Figura 3.13–Classificação das células da <i>octree</i> de estimativa de carga (amarelo - sobre a borda, verde - interna, vermelho - externa, azul - classificação desconhecida). . . . .	63
Figura 3.14–Células cheias em cada subdomínio (amarelo - sobre a borda, verde - interna). . . . .	63
Figura 3.15–Células cheias em cada subdomínio após a remoção de camadas internas (amarelo - sobre a borda, verde - interna). . . . .	64
Figura 3.16–Posicionamento do plano de partição (em preto) com relação às células de uma <i>quadtree</i> . . . . .	65
Figura 3.17–Posicionamento do plano de partição para um conjunto de células de uma <i>quadtree</i> . . . . .	65
Figura 3.18–Decomposição do domínio em cada eixo. . . . .	66
Figura 3.18–Decomposição do domínio em cada eixo (continuação). . . . .	67
Figura 3.18–Decomposição do domínio em cada eixo (continuação). . . . .	68
Figura 3.19–Decomposição no modelo de uma esfera em 16 subdomínios. . . . .	71
Figura 3.20–Exemplo de árvore BSP criada para 5 subdomínios. . . . .	71
Figura 3.21–Exemplo de árvore BSP criada com fator de proporcionalidade para 5 subdomínios. Os fatores estão indicados em cada nó interno da BSP. . . . .	72
Figura 3.22–Visão global da BSP criada para um domínio dado em preto. Os subdomínios dos processos ou <i>threads</i> são mostrados em vermelho. . . . .	73
Figura 3.23–BSPs locais criadas em cada processo. . . . .	73
Figura 3.24–Decomposição da fronteira de uma esfera. . . . .	74
Figura 3.25–Exemplo de fronteira inicial em forma de trinca. A distância entre vértices e faces mostrada é apenas esquemática e não existe na realidade. . . . .	74
Figura 3.26–Restrições no avanço de fronteira. . . . .	75
Figura 3.27–Fronteiras original e atualizada de diversos subdomínios do modelo da esfera. . . . .	75
Figura 3.28–Malhas geradas nos subdomínios do modelo da esfera. . . . .	76
Figura 3.29–Geração das malhas nas <i>threads</i> e nos processos. A malha gerada está representada em cinza. . . . .	77
Figura 4.1 – Topologia da rede do <i>cluster</i> Stampede. Fonte: < <a href="http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide">www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide</a> >. . . . .	78
Figura 4.2 – Exemplos. . . . .	79

Figura 4.3 – Modelo da Viga. . . . .	80
Figura 4.4 – Decomposições do modelo da Viga com 16 processos. . . . .	80
Figura 4.4 – Decomposições do modelo da Viga com 16 processos (continuação). . . . .	81
Figura 4.5 – Tempo de execução e <i>speed-up</i> da Viga. . . . .	82
Figura 4.6 – Detalhamento do tempo de execução. . . . .	84
Figura 4.7 – Detalhamento do tempo de execução em porcentagem. . . . .	85
Figura 4.8 – Detalhamento do tempo de execução de geração de malha. . . . .	86
Figura 4.9 – Detalhamento do tempo de execução de geração de malha em porcentagem. . . . .	87
Figura 4.10 – Tempo de execução por processo. . . . .	88
Figura 4.11 – Número de elementos por subdomínio por processo. . . . .	89
Figura 4.12 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces. . . . .	91
Figura 4.13 – Erro na estimativa do número de elementos em cada tarefa, subdomínios ou interfaces. . . . .	92
Figura 4.14 – Erro médio e desvio padrão da estimativa do número de elementos para o modelo da Viga. . . . .	93
Figura 4.15 – Viga, 1 camada, limiar 50%. . . . .	94
Figura 4.16 – Diferença entre a porcentagem da qualidade das malhas geradas em paralelo e a da malha gerada sequencialmente, para o modelo da Viga, 1 camada, limiar 50%. . . . .	95
Figura 4.17 – Diferença total na qualidade das malhas geradas, para o modelo da Viga. . . . .	96
Figura 4.18 – Malha para o modelo da Viga. . . . .	96
Figura 4.19 – Modelo da Engrenagem (Fratura). . . . .	97
Figura 4.20 – Modelo do Rotor. . . . .	97
Figura 4.20 – Modelo do Rotor (continuação). . . . .	98
Figura 4.21 – Modelo da Engrenagem. . . . .	98
Figura 4.22 – Decomposições dos modelos práticos com 16 processos. . . . .	98
Figura 4.22 – Decomposições dos modelos práticos com 16 processos (continuação). . . . .	99
Figura 4.23 – Decomposição da fratura do modelo da Engrenagem (Fratura) com 16 processos. . . . .	99
Figura 4.24 – Tamanhos das malhas dos modelos práticos. . . . .	100
Figura 4.25 – Tempo de execução e <i>speed-up</i> dos modelos práticos. . . . .	101
Figura 4.26 – Detalhamento do tempo de execução absoluto (coluna da esquerda) e em porcentagem (coluna da direita). . . . .	102
Figura 4.27 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita). . . . .	103
Figura 4.27 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita) (continuação). . . . .	104
Figura 4.28 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, com 16 processos. . . . .	104
Figura 4.28 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, com 16 processos (continuação). . . . .	105
Figura 4.29 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces (coluna da esquerda) e erro na estimativa do número de elementos de cada tarefa (coluna da direita), com 16 processos. . . . .	105
Figura 4.29 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces (coluna da esquerda) e erro na estimativa do número de elementos de cada tarefa (coluna da direita), com 16 processos (continuação). . . . .	106
Figura 4.30 – Erro médio e desvio padrão da estimativa do número de elementos para os modelos práticos. . . . .	107

Figura 4.31–Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença na porcentagem da qualidade das malhas geradas (coluna da direita). . . . .	107
Figura 4.31–Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença na porcentagem da qualidade das malhas geradas (coluna da direita) (continuação). . . . .	108
Figura 4.32–Diferença na porcentagem da qualidade das malhas geradas. . . . .	109
Figura 4.33–Malhas para os modelos práticos. . . . .	109

# LISTA DE TABELAS

Tabela 2.1 – Decomposição do domínio: discreta x contínua. . . . . 31  
Tabela 2.2 – Classificação dos trabalhos. . . . . 50



# LISTA DE ALGORITMOS

3.1 Seleção de eixo(s) . . . . .	70
----------------------------------	----



# SUMÁRIO

	<b>Lista de figuras</b> . . . . .	<b>12</b>
	<b>Lista de tabelas</b> . . . . .	<b>16</b>
	<b>Sumário</b> . . . . .	<b>20</b>
<b>1</b>	<b>INTRODUÇÃO</b> . . . . .	<b>22</b>
<b>1.1</b>	<b>Motivação</b> . . . . .	<b>22</b>
<b>1.2</b>	<b>Objetivos</b> . . . . .	<b>23</b>
<b>1.3</b>	<b>Organização</b> . . . . .	<b>24</b>
<b>2</b>	<b>TRABALHOS RELACIONADOS</b> . . . . .	<b>26</b>
<b>2.1</b>	<b>Introdução</b> . . . . .	<b>26</b>
<b>2.2</b>	<b>Conceitos e classificações</b> . . . . .	<b>26</b>
2.2.1	Técnica de geração da malha . . . . .	26
2.2.1.1	Avanço de fronteira . . . . .	26
2.2.1.2	Delaunay . . . . .	27
2.2.1.3	Arbitrária . . . . .	28
2.2.2	Decomposição do domínio . . . . .	29
2.2.2.1	Decomposição discreta do domínio . . . . .	29
2.2.2.2	Decomposição contínua do domínio . . . . .	30
2.2.2.3	Comparação das técnicas de decomposição . . . . .	31
<b>2.3</b>	<b>Geração em paralelo de malhas</b> . . . . .	<b>32</b>
<b>2.4</b>	<b>Classificação dos trabalhos relacionados</b> . . . . .	<b>49</b>
<b>2.5</b>	<b>Considerações finais</b> . . . . .	<b>50</b>
<b>3</b>	<b>TÉCNICA PROPOSTA</b> . . . . .	<b>52</b>
<b>3.1</b>	<b>Introdução</b> . . . . .	<b>52</b>
<b>3.2</b>	<b>Técnica sequencial</b> . . . . .	<b>53</b>
3.2.1	Construção da <i>octree</i> . . . . .	53
3.2.2	Geração da malha . . . . .	54
3.2.3	Melhoria da malha . . . . .	57
3.2.4	Resultados . . . . .	58
<b>3.3</b>	<b>Técnica paralela</b> . . . . .	<b>58</b>
3.3.1	Estimativa da carga . . . . .	59
3.3.1.1	Classificação das células . . . . .	61
3.3.1.2	Estimativa de carga em cada subdomínio . . . . .	62
3.3.1.3	Estimativa de carga entre subdomínios . . . . .	63
3.3.2	Decomposição do domínio . . . . .	64
3.3.2.1	Posicionamento do plano de partição . . . . .	64
3.3.2.2	Decomposição em cada eixo . . . . .	66
3.3.2.3	Melhor decomposição dentre os eixos . . . . .	68
3.3.2.4	Proporcionalidade nas estimativas de carga . . . . .	70
3.3.2.5	Decomposição em paralelo do domínio . . . . .	72

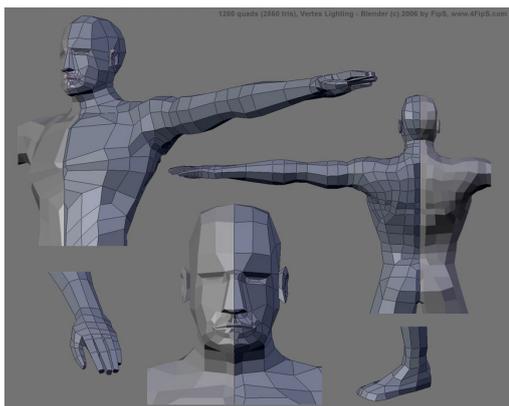
3.3.3	Geração da malha . . . . .	73
3.3.3.1	Fronteira inicial de um subdomínio . . . . .	73
3.3.3.2	Geração da malha em um subdomínio . . . . .	74
3.3.3.3	Geração da malha entre subdomínios . . . . .	76
<b>3.4</b>	<b>Considerações finais . . . . .</b>	<b>77</b>
<b>4</b>	<b>EXEMPLOS E RESULTADOS . . . . .</b>	<b>78</b>
<b>4.1</b>	<b>Introdução . . . . .</b>	<b>78</b>
<b>4.2</b>	<b>Exemplos . . . . .</b>	<b>79</b>
<b>4.3</b>	<b>Viga . . . . .</b>	<b>80</b>
4.3.1	Tempo de execução e <i>speed-up</i> . . . . .	80
4.3.2	Detalhamento do tempo de execução . . . . .	83
4.3.3	Detalhamento do tempo de execução de geração de malha . . . . .	83
4.3.4	Balanceamento da carga . . . . .	88
4.3.5	Estimativa da carga . . . . .	89
4.3.6	Qualidade da malha . . . . .	92
4.3.7	Malha . . . . .	96
<b>4.4</b>	<b>Modelos práticos . . . . .</b>	<b>96</b>
4.4.1	Tempo de execução e <i>speed-up</i> . . . . .	100
4.4.2	Detalhamento do tempo de execução . . . . .	102
4.4.3	Detalhamento do tempo de execução de geração de malha . . . . .	103
4.4.4	Balanceamento da carga . . . . .	103
4.4.5	Estimativa da carga . . . . .	105
4.4.6	Qualidade das malhas . . . . .	106
4.4.7	Malhas . . . . .	108
<b>4.5</b>	<b>Considerações finais . . . . .</b>	<b>108</b>
<b>5</b>	<b>CONCLUSÃO . . . . .</b>	<b>112</b>
<b>5.1</b>	<b>Principais contribuições . . . . .</b>	<b>112</b>
<b>5.2</b>	<b>Trabalhos futuros . . . . .</b>	<b>112</b>
	 <b>APÊNDICE . . . . .</b>	 <b>116</b>
	 <b>APÊNDICE A – ANÁLISE DA TÉCNICA PARALELA . . . . .</b>	 <b>118</b>
<b>A.1</b>	<b>Introdução . . . . .</b>	<b>118</b>
<b>A.2</b>	<b>Complexidade . . . . .</b>	<b>118</b>
A.2.1	Caso bidimensional . . . . .	119
A.2.2	Caso tridimensional . . . . .	122
<b>A.3</b>	<b>Super-linearidade . . . . .</b>	<b>122</b>
<b>A.4</b>	<b>Escalabilidade . . . . .</b>	<b>123</b>
<b>A.5</b>	<b>Considerações finais . . . . .</b>	<b>125</b>
	 <b>Referências . . . . .</b>	 <b>126</b>

# 1 INTRODUÇÃO

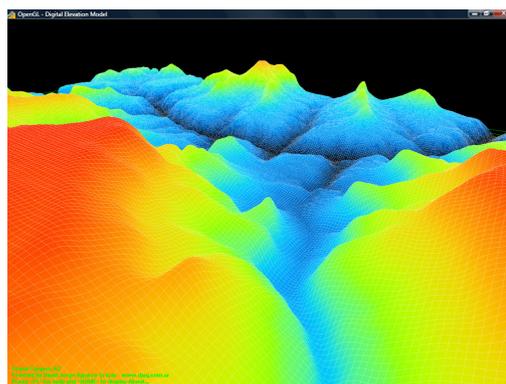
## 1.1 Motivação

Malhas e, em particular, triangulações e tetraedralizações são amplamente utilizadas como uma base para representar geometrias e outras informações que aparecem em uma larga variedade de aplicações (HJELLE; DÆHLEN, 2006). Podem ser utilizadas em:

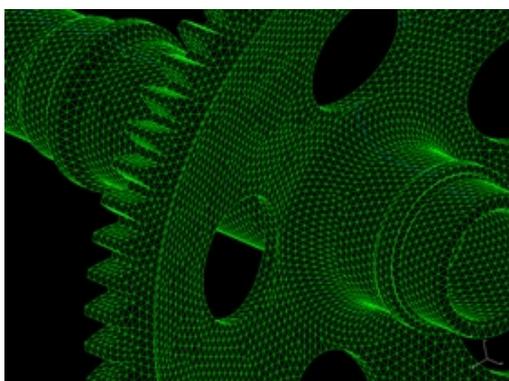
- Computação Gráfica (CG) e visualização, modelando avatares, objetos e ambientes tridimensionais para aplicações de realidade virtual (Figura 1.1a);
- Sistemas de Informações Geográficas (SIG, ou GIS, do inglês *Geographical Information Systems*), representando terrenos e relações entre objetos geográficos (Figura 1.1b);
- Projetos Assistidos por Computadores (ou *Computer Aided Design*, CAD), para a modelagem de objetos industriais a serem construídos (Figura 1.1c);
- Engenharia, ajudando na análise e simulação de fenômenos físicos através de métodos numéricos (Figura 1.1d), como o Método dos Elementos Finitos (MEF, ou *Finite Element Method*, FEM).



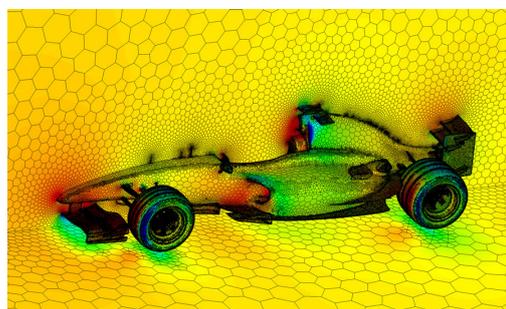
(a) Malha de computação gráfica (modelo humano).  
Fonte: <[www.blenderartists.org](http://www.blenderartists.org)>.



(b) Malha de sistemas de informação geográfica (*Grand Canyon*, EUA). Fonte: <[www.mcrenox.com.ar](http://www.mcrenox.com.ar)>.



(c) Malha de projetos assistidos por computadores (engrenagem). Fonte: <[www.stephensondesign.com](http://www.stephensondesign.com)>.



(d) Malha de engenharia (carro de corrida). Fonte: <[www.ansys.com](http://www.ansys.com)>.

Figura 1.1 – Exemplos de uso de malhas.

A fim de representar mais fielmente os objetos reais, é essencial a utilização de malhas mais refinadas, ou seja, com mais polígonos ou poliedros, permitindo que mais detalhes sejam vistos ou analisados. Com isso, a geração manual dessas malhas pode ser bastante demorada, podendo exigir uma certa capacidade

artística de quem estiver modelando em algumas situações. Portanto, a geração automática dessas malhas tornou-se uma necessidade.

Assim, com o uso de malhas geradas automaticamente, tornou-se possível, por exemplo, simular computacionalmente fenômenos físicos com maior precisão. Entretanto, a utilização de uma única unidade de processamento pode fazer com que essas simulações demorem muito. Logo, com o propósito de diminuir o tempo de execução das simulações, passou-se a utilizar técnicas de Computação de Alto Desempenho, ou seja, a utilização de diversos processadores trabalhando conjuntamente para resolver uma instância de um problema. A Computação de Alto Desempenho também viabiliza simulações em situações onde a memória disponível para uma única unidade de processamento não é suficiente para comportar a malha total.

Além disso, em algumas simulações computacionais, a física ou a geometria do problema mudam com o tempo (BERN et al., 1999), e é necessário gerar uma malha completamente nova a cada passo, por exemplo, em simulações de corpos que se movimentam ou de topologias que se modificam (LÖHNER, 2001). Com isso, o tempo de geração da malha também torna-se crucial para que a simulação aconteça rapidamente. Portanto, também é natural desenvolver técnicas paralelas de geração de malha, uma vez que existem processadores disponíveis e alocados para as simulações computacionais.

## 1.2 Objetivos

Este trabalho tem como objetivo principal propor uma técnica de geração em paralelo de malhas tetraédricas baseada em uma técnica sequencial existente descrita em Miranda et al. (1999), Cavalcante-Neto et al. (2001), cuja versão bidimensional foi inicialmente paralelizada em Freitas (2010), Freitas et al. (2011), Freitas et al. (2013). Essa técnica deve satisfazer os seguintes requisitos:

1. Respeitar a fronteira (discretizada em triângulos) dada como entrada;
2. Garantir a boa qualidade da malha gerada, a fim de prover resultados precisos em simulações de fenômenos físicos;
3. Garantir uma boa transição de malha entre regiões muito refinadas, com muitos tetraedros pequenos, e regiões pouco refinadas, com poucos tetraedros grandes;
4. Ser capaz de tratar fronteiras com trincas, gerando malhas que as respeitem, para que a malha seja utilizada em simulações de propagação de trincas;
5. Ser independente do tipo de arquitetura de memória (compartilhada ou distribuída);
6. Ser capaz de estimar com precisão o número de elementos gerados em cada subdomínio;
7. Apresentar um bom balanceamento de carga, distribuindo uniformemente os subdomínios entre as unidades de processamento;
8. Apresentar um bom *speed-up*.

Os quatro primeiros requisitos pertencem à técnica sequencial e, conseqüentemente, a técnica paralela também deve satisfazê-los. O primeiro requisito é muito importante em diversos problemas, tais como os encontrados em simulações cujos domínios contenham regiões com diferentes materiais e/ou buracos. Nesses problemas, deseja-se, geralmente, que a malha seja conforme com uma discretização existente do contorno dessas regiões.

Com relação ao segundo requisito, embora a técnica sequencial não garanta limites na razão de aspecto dos elementos, a cada passo, toma-se cuidado para gerar elementos com a melhor razão de aspecto possível. Mostrando que a técnica paralela gera malhas tão boas quanto a técnica sequencial, o segundo requisito é satisfeito.

Com respeito ao terceiro requisito, em muitas aplicações, a diferença entre o tamanho de elementos nas regiões refinadas e o de elementos nas regiões grosseiras é maior que duas ordens de magnitude. Assim, prover bons elementos em regiões de transição é um requisito importante.

O quarto requisito é importante em simulações de propagação de trincas porque permite que a geração da malha ocorra localmente, em regiões próximas à ponta da trinca. Nas malhas deste trabalho, a superfície da trinca é representada como uma região de volume nulo, formada por faces da entrada geometricamente coincidentes, mas cujas normais têm sentidos opostos. É importante mencionar que, embora a técnica foi idealizada para funcionar em modelos com trincas, ela funciona igualmente em modelos sem trincas, sem nenhuma modificação. Outros tipos de descontinuidades, como arames ou curvas soltas, podem ser facilmente incorporados tanto à implementação sequencial quanto à paralela.

Os quatro últimos requisitos dizem respeito somente à técnica paralela. Deseja-se que a implementação execute em diversos tipos de plataformas paralelas, desde computadores pessoais com vários núcleos a computadores do tipo *cluster*, com vários processadores. Portanto, o algoritmo deve ser genérico o suficiente para abranger arquiteturas de memória compartilhada e distribuída, e a implementação deve refletir essa característica. Embora os resultados mostrados neste trabalho sejam somente para uma máquina de memória distribuída, a técnica em si não apresenta nenhuma restrição quanto ao tipo de memória. Isso é uma vantagem desta técnica, uma vez que muitas das técnicas paralelas apresentadas na literatura funcionam somente para um dos tipos de memória. Além disso, a técnica funciona tanto para duas quanto para três dimensões, isto é, ela pode ser utilizada para gerar malhas triangulares bem como para gerar malhas tetraédricas, apesar de a versão 3D ser o foco deste trabalho.

Os últimos três requisitos estão relacionados uns aos outros. Dado que a quantidade de trabalho, chamada de carga em Computação de Alto Desempenho, é bem estimada, e dado que a decomposição é apropriada para particionar o domínio em subdomínios com aproximadamente a mesma carga, o *speed-up* atingido pela técnica deve ser bom o suficiente para fins práticos.

As principais diferenças do presente trabalho para os existentes são:

- Preservação da fronteira, pois diversos trabalhos relacionados criam sua própria fronteira ou refinam a entrada;
- Preservação da qualidade da malha gerada, que não deve ser muito diferente da qualidade da malha gerada sequencialmente;
- Tratamento transparente de trincas, cujas faces são tratadas como qualquer outra face da fronteira;
- Estimativa da carga de trabalho, algo que muitos trabalhos relacionados omitem ou negligenciam;
- Decomposição do domínio baseada na estimativa da carga de trabalho, pois, nos trabalhos relacionados que apresentam alguma estimativa de carga, esta é feita após a decomposição;
- Análise da estimativa de carga, que deve ser corresponder à quantidade real de trabalho de cada subdomínio.

### 1.3 Organização

Este trabalho está organizado em cinco capítulos. O Capítulo 2 relata as principais técnicas existentes de geração em paralelo de malhas. O Capítulo 3 apresenta a técnica proposta neste trabalho e o Capítulo 4 traz os resultados obtidos com a sua implementação. No Capítulo 5, são apresentadas as conclusões acerca do trabalho e algumas sugestões de trabalhos futuros.



## 2 TRABALHOS RELACIONADOS

### 2.1 Introdução

Este capítulo apresenta alguns trabalhos que, de alguma forma, estão relacionados com esta pesquisa. Inicialmente, são mostradas duas formas de classificação das técnicas paralelas de geração de malhas, a primeira, de acordo com a técnica sequencial utilizada para gerar a malha, e a segunda, com maneira como o domínio é decomposto. Posteriormente, os diversos trabalhos relacionados são apresentados em ordem cronológica. Estes trabalhos são, então, classificados e associados à presente pesquisa.

### 2.2 Conceitos e classificações

#### 2.2.1 Técnica de geração da malha

Existem várias classificações possíveis para as técnicas de geração de malhas. Neste trabalho, elas são classificadas em três categorias: Avanço de Fronteira, Delaunay e Arbitrárias. Nos algoritmos de avanço de fronteira, a malha deve ser criada a partir de restrições que formem o contorno da região. Nos algoritmos de Delaunay, maximiza-se o menor ângulo dos triângulos gerados. Todas as outras técnicas são chamadas de arbitrárias. Por razões didáticas, as explicações no restante do texto serão ilustradas com modelos bidimensionais, embora as mesmas ideias se apliquem em casos tridimensionais.

##### 2.2.1.1 Avanço de fronteira

Os algoritmos de avanço de fronteira partem de um contorno especificado da região a ser preenchida (Figura 2.1a). Este contorno é chamado de fronteira inicial, e um elemento é gerado por vez a partir dessa fronteira. À medida que os polígonos (ou poliedros, em três dimensões) são gerados, a fronteira é atualizada, sempre removendo e/ou adicionando elementos de fronteira. O algoritmo termina quando não há mais fronteira, indicando que a região foi totalmente preenchida, ou quando não é possível gerar mais nenhum elemento mesmo com fronteira, indicando que o algoritmo falhou.

Uma fronteira bidimensional é formada por arestas, enquanto que uma fronteira 3D é uma superfície poligonal, geralmente formada por triângulos. Essa técnica geralmente faz uso de pontos internos, ou seja, insere novos vértices, que não pertencem à entrada. Um algoritmo nessa categoria procede da seguinte forma, de uma maneira geral, no caso 2D triangular (Figura 2.1), enquanto houver arestas na fronteira:

1. Escolha e remova uma aresta da fronteira, a aresta base (Figura 2.1b);
2. Encontre um ponto ideal para a formação de um novo triângulo com a aresta base (Figura 2.1c);
3. Crie uma região de busca em torno desse ponto ideal (Figura 2.1d);
4. Selecione o vértice dentro dessa região de busca cujo triângulo (entre esse vértice e a aresta base) seja válido e seja o de melhor qualidade:
  - a) Caso não exista nenhum vértice nessa região, verifique se o triângulo formado com o ponto ideal é válido;
  - b) Se for válido, adicione esse ponto à lista de vértices. Se não for válido, mova essa aresta para o final da fronteira, e vá para o passo 1, ou termine o algoritmo indicando falha;
5. Forme o novo triângulo com o vértice selecionado e adicione-o à malha (Figura 2.1e);
6. Atualize a fronteira, inserindo as arestas que foram criadas, e removendo as arestas que já existiam;
7. Se existir aresta na fronteira, volte para o passo 1.

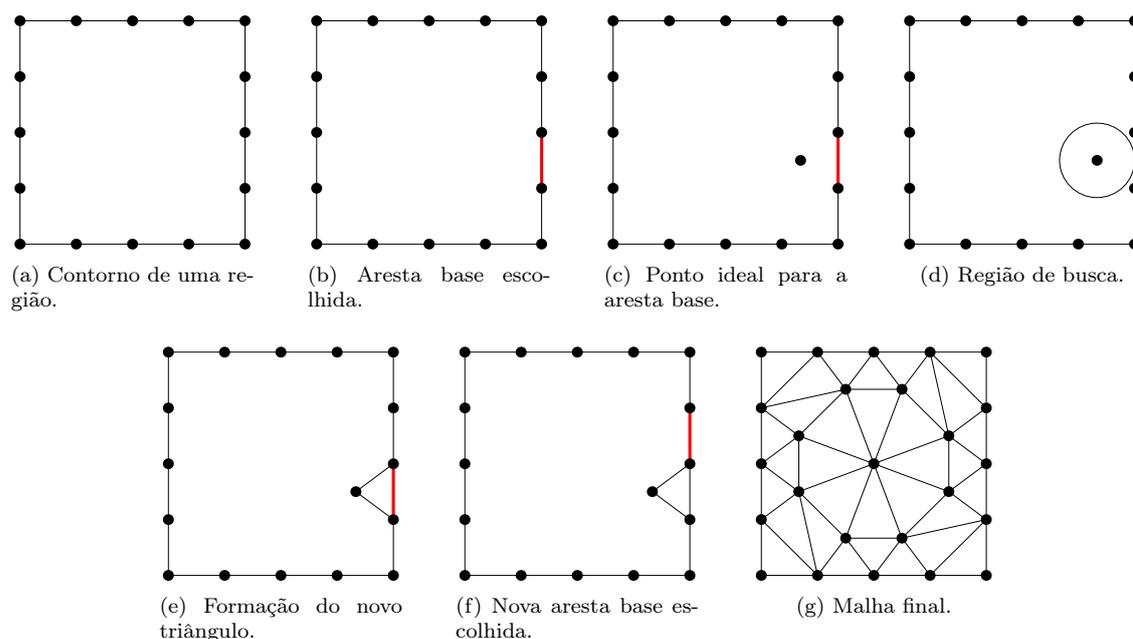


Figura 2.1 – Avanço de fronteira.

Os algoritmos de avanço de fronteira têm facilidade em tratar regiões descontínuas, ou por conterem buracos, ou por serem regiões separadas, uma vez que a fronteira é sempre respeitada. Além disso, os elementos próximos à fronteira são, geralmente, de boa qualidade, o que muitas vezes é interessante na aplicação de métodos numéricos (como os MEFs) sobre a malha gerada, já que, muitas vezes, essas regiões apresentam altos gradientes.

Entretanto, em algumas vezes, os elementos mais internos à malha nem sempre são de boa qualidade, pois a região torna-se menor à medida que a fronteira avança. Para tratar esses casos, geralmente uma técnica de suavização ou otimização é aplicada na malha resultante do algoritmo. Existem, ainda, alguns casos onde o algoritmo falha em gerar a malha, geralmente casos tridimensionais.

É muito difícil estimar-se a complexidade de algoritmos de avanço de fronteira, uma vez que ela depende da quantidade de elementos de fronteira existentes na malha final em vez do tamanho da entrada, ou seja, a quantidade de arestas/polígonos da fronteira. Assim sendo, a complexidade é geralmente estimada a partir de execuções exaustivas do algoritmo, resultando em uma complexidade empírica de  $O(N^p \log N)$ , onde  $N$  é o número de elementos da saída e  $p$  é ligeiramente maior que 1 (JIN; TANNER, 1993; MÖLLER; HANSBO, 1995).

### 2.2.1.2 Delaunay

Na técnica de Delaunay (nome dado em homenagem ao matemático russo Boris Delaunay), a região a ser preenchida é, em geral, o fecho convexo do conjunto de vértices dados como entrada e, muitas vezes, são necessários refinamentos para a inserção de pontos internos. Essa técnica é bem estudada e tem uma teoria bem formalizada (HJELLE; DÆHLEN, 2006). A triangulação gerada por um algoritmo de Delaunay é um grafo dual ao diagrama de Voronoi (nome dado por conta do também matemático russo Georgy Voronoi) (Figura 2.2). Dessa forma, um vértice qualquer da malha é sempre adjacente ao vértice mais próximo dele.

Para um triângulo ser chamado de triângulo de Delaunay, e, portanto, pertencer à malha final, é necessário que não exista nenhum outro vértice dentro do círculo que passa pelos três vértices desse triângulo (o circuncírculo desse triângulo, Figura 2.3), chamado de critério do círculo ou critério de

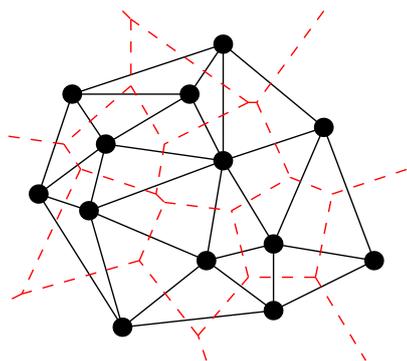
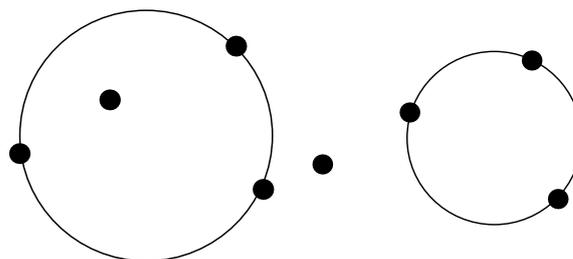


Figura 2.2 – Diagrama de Voronoi (em tracejado).

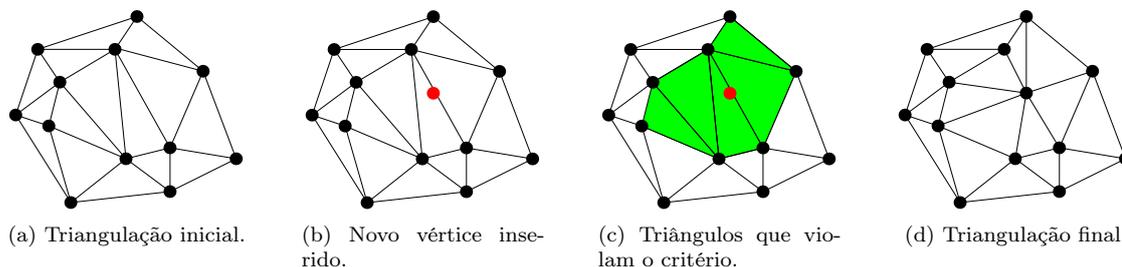


(a) Critério não satisfeito. (b) Critério satisfeito.

Figura 2.3 – Critério de Delaunay.

Delaunay. Na tetraedralização de Delaunay, é considerada uma esfera circunscrevendo um tetraedro. A triangulação de Delaunay é, ainda, aquela que maximiza os ângulos internos dos triângulos da malha (HJELLE; DÆHLEN, 2006).

Existem algumas variações de algoritmos de Delaunay. Em uma delas, encontra-se uma aresta que faz parte da triangulação e, a partir dela, acha-se um triângulo de Delaunay. Assim, com as novas arestas, encontram-se novos triângulos, em um algoritmo parecido com o de avanço de fronteira. Em outra variação, insere-se um vértice por vez, criando-se um triângulo inicial ou modificando-se uma triangulação de Delaunay (de apenas um subconjunto de vértices da entrada) pré-existente (Figura 2.4).



(a) Triangulação inicial. (b) Novo vértice inserido. (c) Triângulos que violam o critério. (d) Triangulação final.

Figura 2.4 – Triangulação por inserção de vértices.

A complexidade do problema de gerar uma triangulação de Delaunay é  $O(n \log n)$ . Existem variações da técnica de Delaunay com restrições, ou seja, com arestas dadas na entrada, que podem ser tanto internas quanto de fronteira. Nesses casos, a malha gerada é a mais próxima de uma Delaunay possível, respeitando as restrições (HJELLE; DÆHLEN, 2006).

A desvantagem da triangulação de Delaunay é que os elementos mais próximos do fecho convexo da região (ou do contorno, utilizando restrições) nem sempre têm boa qualidade, o que nem sempre é adequado. Para contornar essa situação, refinamentos e otimizações, que fazem uso de pontos internos, são utilizados para se melhorar a qualidade da malha (RUPPERT, 1999).

### 2.2.1.3 Arbitrária

Neste trabalho, são chamadas técnicas arbitrárias aquelas que não se enquadram nem como avanço de fronteira nem como de Delaunay. Alguns exemplos de técnicas aqui agrupadas são aquelas que geram malhas de *quadtrees/octrees*, por varredura, por extrusão, por bissecção de arestas, por aplicação de padrões, de Voronoi, dentre outras.

Por não terem tantos requisitos, as malhas arbitrárias podem ser utilizadas para demonstrações de teoremas. Por exemplo, um teorema importante afirma que o problema de geração de malhas bidimen-

sionais triangulares tem complexidade  $\Omega(n \log n)$ , com  $n$  sendo a quantidade de pontos da entrada, e pode ser demonstrado a partir de uma redução do problema de ordenação (CARVALHO; FIGUEIREDO, 1991). Isso acontece por que o número de elementos da malha final é função linear no número de pontos da entrada, o que não é necessariamente verdade em três dimensões.

## 2.2.2 Decomposição do domínio

Nas técnicas paralelas de geração de malha, costuma-se decompor (ou particionar) o domínio de entrada em diversos subdomínios, que são, então, distribuídos entre os processos ou entre as *threads*. Estes são responsáveis pela geração simultânea das malhas nos subdomínios. As regiões da malha entre subdomínios são chamadas de interfaces e, em algumas técnicas, é necessário haver sincronização para as malhas de interface serem geradas.

Para decompor o domínio, Löhner (2001) apresenta o argumento da distância, que afirma que a adição de um novo elemento à malha afeta somente os elementos próximos a ele. Ou seja, dado que elementos a serem gerados estão suficientemente distantes uns dos outros, eles podem ser adicionados à malha concorrentemente.

Este argumento é aplicado diretamente para as técnicas de avanço de fronteira pois, na adição de um elemento, é necessário fazer testes de interseção com os elementos vizinhos e/ou próximos. Elementos distantes, por exemplo, que estão do outro lado da malha, não afetam a adição desse elemento. Em técnicas de Delaunay, alguns testes a mais são necessários, mas o mesmo argumento vale, dessa vez, para a inserção de novos vértices da entrada. Técnicas arbitrárias também seguem a mesma lógica.

Chrisochoides (2005) apresenta um levantamento de diversas técnicas de geração de malha e as classifica de acordo com a maneira como o domínio é decomposto, para garantir que os subdomínios estejam suficientemente distantes uns dos outros. Os tipos de decomposição são discreta ou contínua.

### 2.2.2.1 Decomposição discreta do domínio

As técnicas de decomposição discreta do domínio geralmente usam uma malha grosseira para a região, a partir da entrada dada, gerada por algum algoritmo sequencial. Tem-se, então, duas abordagens comumente usadas. Na primeira delas (Figura 2.5), cada elemento é considerado como uma sub-região (WILSON; TOPPING, 1998). Na segunda (Figura 2.6), cada sub-região é formada pelo agrupamento de elementos próximos (ITO et al., 2007). Uma vez definidas as sub-regiões, cada uma delas forma um subdomínio, e suas bordas são refinadas (tanto as interfaces entre dois subdomínios quanto o contorno dado como entrada). Então, um algoritmo sequencial é aplicado em cada subdomínio em um processo diferente.

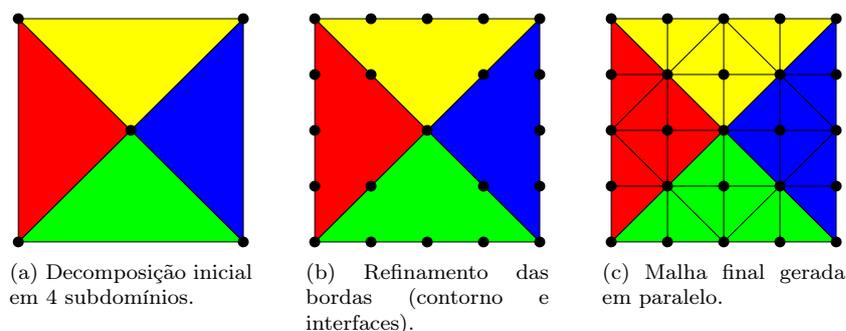


Figura 2.5 – Decomposição discreta do domínio sem agrupamento.

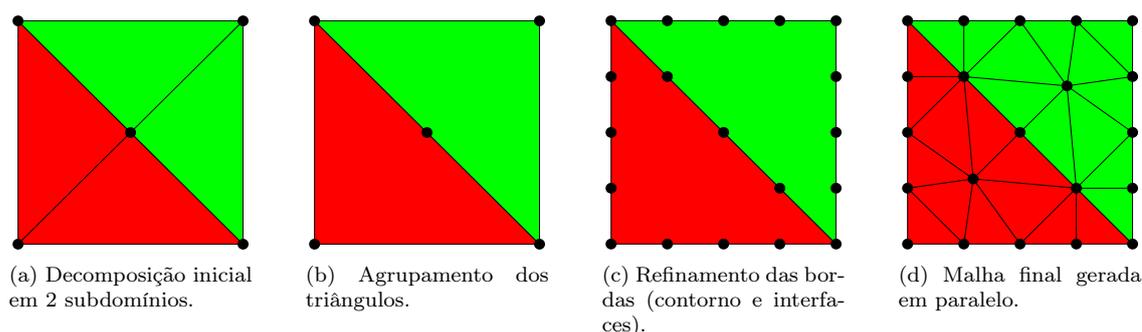


Figura 2.6 – Decomposição discreta do domínio com agrupamento.

Existem técnicas que utilizam decomposição discreta do domínio sem refinar o contorno dado como entrada (MORETTI, 2001). Entretanto, tais técnicas geralmente apresentam um *speed-up* limitado. Esse tipo de algoritmo tem a vantagem de não necessitar de nenhuma modificação nos algoritmos sequenciais utilizados, podendo-se utilizar algoritmos estáveis e bem testados. Além do mais, não é necessária muita comunicação entre os subdomínios. Nas técnicas com agrupamento, os subdomínios são geralmente definidos pelo particionamento da malha (ou do grafo representado pela malha) (TOPPING; KHAN, 1996; KARYPIS; KUMAR, 1998; KARYPIS et al., 2011), problema este que é NP-Completo (BISWAS et al., 2000; LÄMMER; BURGHARDT, 2000). Se a decomposição não for bem feita, as sub-regiões podem não proporcionar a geração de uma boa malha interna nem uma transição suave entre duas sub-malhas adjacentes. Além disso, é possível que nem a malha grosseira seja gerada, inviabilizando esse tipo de método. Apesar de exemplificada com a técnica de avanço de fronteira, técnicas de Delaunay e técnicas arbitrárias também podem utilizar esta maneira de decompor o domínio.

### 2.2.2.2 Decomposição contínua do domínio

Nas técnicas de decomposição contínua do domínio, os subdomínios são encontrados de qualquer forma que não seja pelo particionamento de uma malha grosseira. Assim, um subdomínio é definido como uma parte do contorno juntamente com uma parte da região interna a ele. Então, malhas são geradas, uma para cada subdomínio, concorrentemente. Esse tipo de decomposição pode, ainda, ser subdividido em duas categorias, dependendo da forma como é gerada a malha entre os subdomínios, chamada de malha de interface. Se for gerada antes da malha interna ao subdomínio, essa abordagem é chamada de *a priori*. Caso seja gerada depois, é chamada de *a posteriori*. A geração da malha de interface *a posteriori* geralmente requer sincronização entre processos (Figuras 2.7 e 2.8).

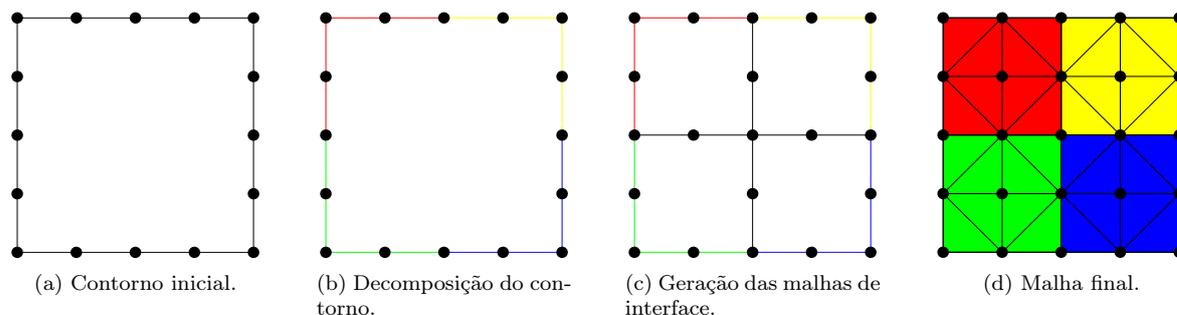


Figura 2.7 – Decomposição contínua do domínio com interfaces *a priori*.

A decomposição contínua tem a vantagem de não precisar particionar uma malha. Na abordagem *a posteriori*, não é necessário definir malhas de interface, que podem interferir na qualidade da malha

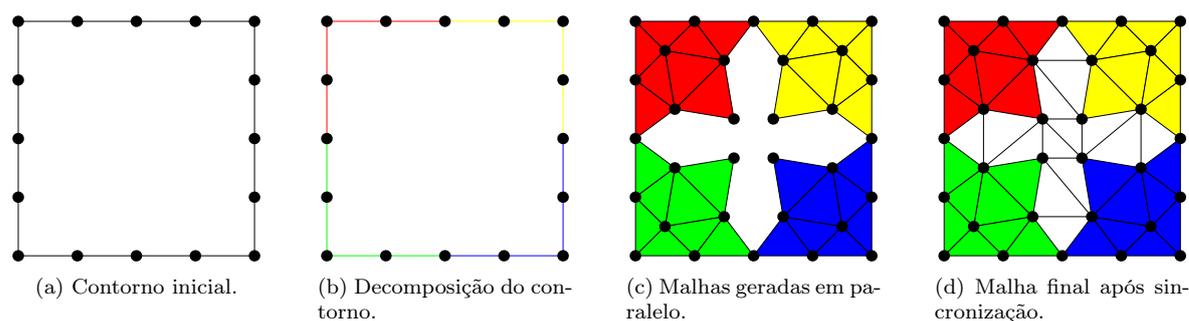


Figura 2.8 – Decomposição contínua do domínio com interfaces *a posteriori*.

final. Outra vantagem desse algoritmo é a de preservar o contorno dado como entrada, sem modificá-lo através de refinamentos. Entretanto, na abordagem *a posteriori*, o algoritmo sequencial deve ser levemente modificado para decidir se a criação de um ponto interno respeita os limites de uma sub-região. Além disso, nessa abordagem, é necessário haver sincronização para a geração completa da malha, o que não acontece com a abordagem *a priori*. Aqui também foi utilizada a técnica de avanço de fronteira nos exemplos, mas técnicas de Delaunay e técnicas arbitrárias podem utilizar esta maneira de decompor o domínio, com as devidas adaptações.

As técnicas de decomposição contínua proporcionam mais liberdade de escolha, e envolvem subdivisões regulares do domínio, subdivisões recursivas (*quadtree* ou *octree*) (LÖHNER, 2001), criação de linhas de contorno (GLOBISH, 1995) ou planos separadores apropriados para a decomposição (DE COUGNY; SHEPHARD, 1999; ZAGARIS et al., 2009), dentre outras. A forma de decomposição deve ser facilmente encontrada, para não degradar a performance do algoritmo como um todo.

### 2.2.2.3 Comparação das técnicas de decomposição

Por utilizar técnicas de particionamento de malhas, um método de decomposição discreta utiliza-se principalmente das informações topológicas (de vizinhança) na malha. Um método de decomposição contínua, por outro lado, utiliza-se essencialmente de informações geométricas presentes no domínio. A Tabela 2.1 indica as principais vantagens e desvantagens dos métodos de avanço de fronteira por decomposição discreta e contínua.

Tabela 2.1 – Decomposição do domínio: discreta x contínua.

Característica	Discreta	Contínua	
		<i>A priori</i>	<i>A posteriori</i>
Particionamento	Malha grosseira	Qualquer ☹	Qualquer ☹
Enfoque	Topologia	Geometria ☹	Geometria ☹
Refinamento da borda	Geralmente necessário	Não necessário ☹	Não necessário ☹
Contornos internos	Necessários	Necessários	Não necessários ☹
Algoritmos conhecidos	Sem modificações ☹	Sem modificações ☹	Com modificações
Transição	Pode ser ruim	Pode ser ruim	Geralmente boa ☹
Sincronização	Inexistente/pouca ☹	Inexistente/pouca ☹	Razoável/muita

A existência dos contornos internos, presentes na técnica de decomposição discreta e na de decomposição contínua *a priori*, facilita a geração da malha, uma vez que técnicas sequenciais existentes podem ser utilizadas sem modificações e que há pouca ou nenhuma sincronização. Entretanto, os contornos internos podem levar à geração de elementos ruins próximos a eles, bem como a uma pobre transição de tamanhos de elementos entre regiões mais refinadas e regiões mais grosseiras da malha.

Quando os contornos internos não existem, na técnica de decomposição contínua *a posteriori*, geralmente, a malha final tem boa qualidade. Entretanto, na maioria dos casos, é necessário haver sincronização entre processos para a geração da malha nas regiões de interfaces. Portanto, observa-se um conflito de escolha (*trade-off*) entre qualidade da malha final e rapidez de geração da malha.

Em um modelo com trincas, quando as interfaces passam por elas, a geração dos contornos internos pode ser mais complicada, necessitando de tratamentos específicos. Além disso, em técnicas de decomposição discreta, o refinamento da borda é geralmente necessário. Algumas dessas observações, principalmente o de qualidade da malha, violam os quatro primeiros requisitos da técnica apresentados na Seção 1.2. Portanto, a decomposição contínua *a posteriori* foi escolhida como técnica de decomposição neste trabalho.

### 2.3 Geração em paralelo de malhas

Em Merks (1986) e ElGindy (1986), são apresentados dois resultados teóricos sobre triangulação de pontos. Em Merks (1986), uma técnica é apresentada demonstrando que é possível se obter uma triangulação de  $n$  pontos bidimensionais em tempo  $O(\log n)$ , utilizando  $O(n)$  processadores. Em ElGindy (1986), é apresentado um algoritmo para a geração de triangulações em qualquer dimensão em tempo  $O(\log^2 n)$ , com  $O(n/\log n)$  processadores, desde que o fecho convexo desses pontos seja um *simplex* naquela dimensão.

Dados  $n$  pontos, a técnica em Merks (1986) encontra o ponto de menor ordenada e ordena os outros pontos de acordo com o ângulo que formam com esse ponto e o eixo das abscissas. Então, essa lista ordenada é dividida em  $\sqrt{n}$  sub-listas, cada uma com  $\sqrt{n}$  pontos mais o ponto de menor ordenada, e  $O(\sqrt{n})$  processadores são alocados para cada sub-lista. Assim, cada triangulação de cada sub-lista é encontrada recursivamente. Posteriormente, uma estrutura de dados auxiliar é utilizada para achar as conexões entre duas triangulações adjacentes em tempo apropriado. Ao final, a técnica leva somente um tempo de  $O(\log n)$ , usando  $n$  processadores. Apesar do bom resultado, uma versão tridimensional desta técnica não é trivial, por conta da estrutura de dados utilizada para achar as conexões entre triangulações vizinhas.

Em ElGindy (1986), tem-se, como pré-requisito, que o fecho convexo dos  $n$  pontos da entrada seja um *d-simplex*. Assim, o método apresentado encontra um ponto interno adequado que divida o fecho convexo em  $d$  *d-simplices*, de forma que cada *d-simplex* tenha uma fração dos  $n$  pontos. Dessa forma, o método é aplicado recursivamente, alocando-se os processadores de forma adequada para cada *d-simplex*. Ao final, esse algoritmo leva um tempo de  $O(f(d) \log^2 n)$  utilizando-se  $O(n/\log n)$  processadores. Portanto, para uma dimensão fixa, a complexidade é de  $O(\log^2 n)$ . Embora essa técnica seja apropriada para qualquer dimensão, por necessitar que o fecho convexo seja um *simplex*, tem pouca usabilidade na prática.

Em Khan e Topping (1991), uma técnica para geração de malhas bidimensionais triangulares é apresentada. Inicialmente, uma malha de *background* é gerada e cada um de seus elementos é considerado um subdomínio. O número de elementos que serão gerados em cada triângulo da malha de *background* é estimado baseado em parâmetros nodais dados como entrada. Para distribuir os subdomínios para os processos escravos, o processo mestre enche a rede com os elementos, que são recebidos dois a dois por cada processo escravo, que refina as bordas de cada subdomínio triangular e gera elementos internos. Uma borda compartilhada por dois subdomínios é refinada igualmente em dois processos diferentes, uma vez que os parâmetros nodais são os mesmos. Para finalizar, o processo mestre recolhe toda a malha gerada e aplica procedimentos de otimização, como suavização e troca de diagonais.

Em Teng et al. (1993), é apresentado um algoritmo para geração de malhas tridimensionais de Delaunay de um conjunto de pontos. Para cada ponto da entrada, concorrentemente, o ponto mais próximo é encontrado, juntamente com o ponto que forma o maior ângulo com essa aresta. Então, cada uma dessas

faces é expandida concorrentemente, ou seja, é feita uma busca pelo ponto que satisfaz ao critério de Delaunay. Para trabalhar concorrentemente, o algoritmo utiliza uma estrutura de dados parecida com um vetor, que é particionado igualmente entre os processadores envolvidos. Ainda são empregadas estruturas de dados para as buscas e testes de validade, com a finalidade de garantir a boa formação da malha final. Além de inerentemente compartilhado, esta técnica desperdiça tempo gerando elementos repetidos, tendo que removê-los posteriormente.

Uma técnica para gerar malhas tetraédricas em paralelo a partir de *templates* e de avanço de fronteira é apresentada em De Cougny et al. (1994) e melhorada em De Cougny et al. (1996). Inicialmente, uma *octree* é gerada em cada processo, conformando-se à fronteira original, e um refinamento para garantir uma transição suave nos tamanhos das células é aplicado. As células são classificadas em “interiores”, “exteriores” e “sobre” a borda (células internas próximas à borda são também classificadas como “sobre” a borda), e cada célula “sobre” a borda mantém informações sobre as faces da fronteira que a cruzam. A *octree* é linearizada de acordo com o centroide das suas células internas, e esta linearização é particionada entre os processos, de forma que cada processo tem uma porção da *octree*. Malha é gerada nessas células internas utilizando *templates*. Posteriormente, as células classificadas como “sobre” são particionadas da mesma forma, para o procedimento de avanço de fronteira. Para escolher o vértice que formará o tetraedro de uma face, é necessário conhecer a vizinhança da célula da *octree* que a contém. Assim, se alguma célula vizinha estiver em outro processo, as faces dessa célula não avançam. As faces que não observam essa restrição avançam normalmente. Após avançar o máximo possível, as células que não contêm faces são descartadas, e as células restantes são reparticionadas. Esse procedimento de geração de malha e reparticionamento das células da *octree* é feito até que a malha seja completamente gerada, diminuindo pela metade o número de processos sempre que a razão entre o número faces que avançaram e o número de tentativas de avanço for menor que 25% (Figura 2.9). A malha final é reparticionada pela mesma técnica de particionamento das células da *octree*.

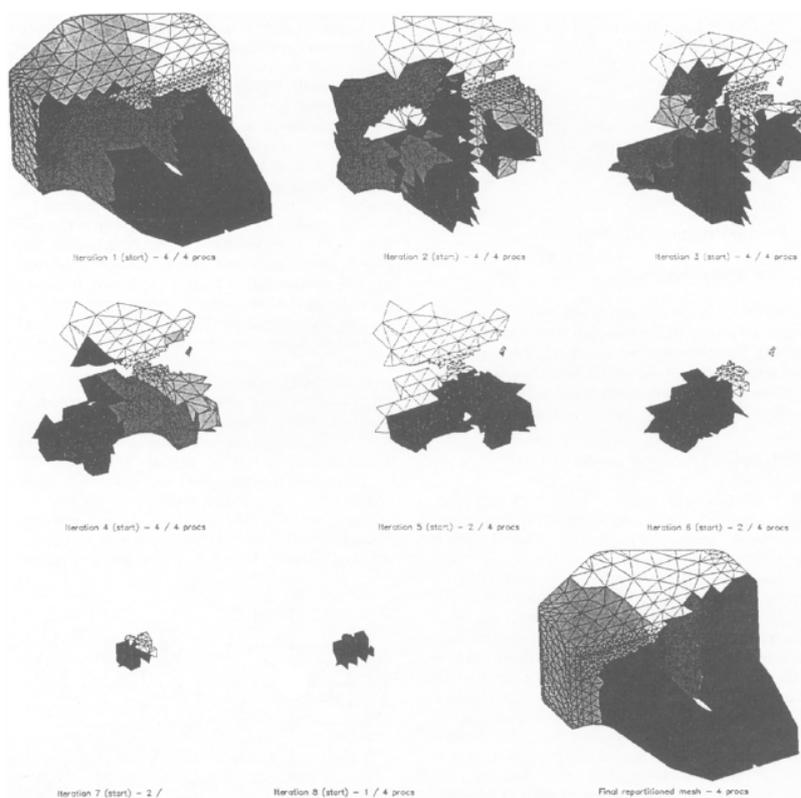


Figura 2.9 – Iterações de avanço de fronteira e malha final particionada em 4 processos (DE COUGNY et al., 1996).

Em Globish (1995), um algoritmo simples para a geração de malhas bidimensionais por avanço de fronteira é apresentado. A entrada é a geometria de um objeto, já dividida em vários subdomínios. Então, paralelamente para cada subdomínio, as arestas de cada borda são criadas, e uma técnica de avanço de fronteira é empregada. Por ser simples, essa técnica não trata os principais problemas envolvidos com o problema de gerar malhas em paralelo. Por exemplo, essa técnica não tenta distribuir a carga de forma balanceada entre os processadores, simplesmente assume que a ordem dada já está balanceada.

Em Okusanya e Peraire (1996), é mostrada uma técnica para refinamento de malhas de Delaunay. Dada uma malha de Delaunay, ela é particionada em blocos cartesianos, encontrados por técnicas de programação linear, e dividida entre os processadores. Então, cada bloco é refinado em paralelo, por meio de inserção de novos pontos. Quando a inserção de um desses pontos influencia uma parte da malha que não se encontra na memória local de um processador, os processadores envolvidos trocam mensagens e parte da malha é transferida para o processador que faz a inserção (Figura 2.10). Travas (*locks*) são utilizadas para garantir que uma mesma parte da malha não seja influenciada por duas inserções distintas simultaneamente. Esta técnica tem a vantagem de gerar uma malha já particionada para a aplicação de MEFs. Entretanto, ela não mostrou uma boa divisão do trabalho entre os processadores durante a geração.

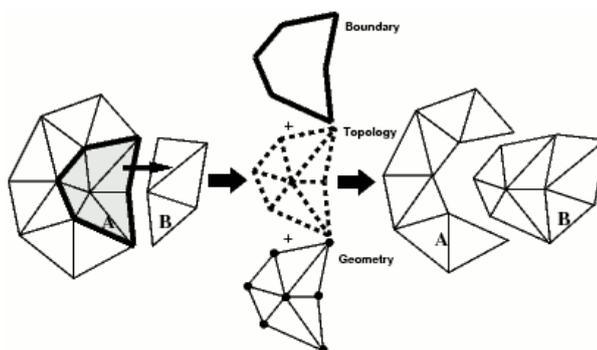


Figura 2.10 – Migração de elementos do processador A para o processador B (OKUSANYA; PERAIRE, 1996).

Em Hodgson e Jimack (1996), é apresentada uma técnica para a geração de malhas bidimensionais, com a vantagem que a malha final já está particionada para a aplicação de um método de elementos finitos. Inicialmente, uma malha de Delaunay (com pontos internos) grosseira é gerada para o contorno dado. Então, as arestas são refinadas, utilizando estimativas nos pontos, estimativas estas que foram dadas como entrada, no caso dos pontos originais, ou interpoladas, no caso dos pontos internos. A carga é calculada para cada um dos triângulos da malha grosseira, tentando prever a quantidade de elementos a serem gerados em cada um. Para calcular a carga, uma generalização para duas dimensões da mesma interpolação empregada para as arestas é utilizada. Então, um balanceamento da carga total é feito, e as malhas são geradas concorrentemente, uma para cada triângulo da malha grosseira, usando o mesmo algoritmo que gerou a malha inicial. O método apresentado tem ainda a opção de agrupar triângulos da malha grosseira em regiões maiores (Figura 2.11). Por utilizar decomposição discreta, essa técnica pode levar à inserção de artefatos no interior da malha, o que pode influenciar a qualidade dos elementos gerados.

Em Wu e Houstis (1996), uma técnica para geração de malhas bidimensionais triangulares ou predominantemente quadrilaterais é apresentada. Inicialmente, uma malha é gerada para a fronteira dada, cujos pontos internos são inseridos com o auxílio de uma *quadtree*. Esta malha é particionada, utilizando-se um método de particionamento de grafos descrito no artigo (particionamento por elementos ou por vértices) ou ainda um método que envolve apenas geometria (por eixos cartesianos, por eixos polares ou por eixos de inércia). Os subdomínios são identificados, as arestas da fronteira original e das arestas de interface

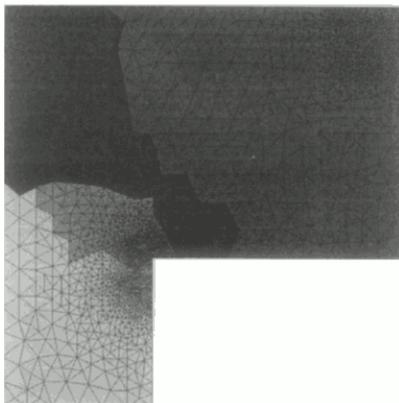


Figura 2.11 – Malha gerada pela técnica de Hodgson e Jimack (1996), já particionada.

são refinadas, e uma malha é gerada para cada subdomínio independentemente dos outros, utilizando o mesmo procedimento para a geração da malha inicial.

Em Wilson e Topping (1998), é apresentada uma técnica para a geração de malhas tridimensionais em paralelo por avanço de fronteira. Uma malha grosseira é dada como entrada, juntamente com a decomposição do domínio, ou seja, quais são os tetraedros atribuídos a um determinado processador. Então, cada processador triangulariza as faces de um tetraedro e, posteriormente, tetraedraliza o seu interior. A mesma face é triangularizada igualmente em processadores distintos, de forma a garantir a conformidade da malha. Esta técnica também sofre com os problemas existentes na decomposição discreta, além de não tentar distribuir melhor o trabalho total entre os processadores.

Em De Cougny e Shephard (1999), dado o contorno de um objeto, uma *octree* distribuída entre os diversos processadores é inicialmente gerada, criando planos de separação do domínio (Figura 2.12a), e suas células internas são concorrentemente preenchidas com *templates*. A região entre o contorno e as células internas são preenchidas por uma técnica de avanço de fronteira, dividida em etapas. Na primeira etapa, são gerados os elementos internos a uma região delimitada pelos planos de separação entre os processadores (Figura 2.12b). Então, os processadores que compartilham um determinado plano de separação entram em consenso para se determinar qual deles gerará os elementos que conectarão as malhas dos dois lados do plano (Figura 2.12c). Posteriormente, uma técnica de consenso parecida é empregada para as retas definidas pelas interseções de dois planos (Figura 2.12d). Para finalizar, os vértices, interseções de três ou mais planos, também são preenchidos. Por usar *templates*, esta técnica pode gerar uma quantidade excessiva de elementos no interior, além de poder levar à geração de elementos ruins nas proximidades do contorno.

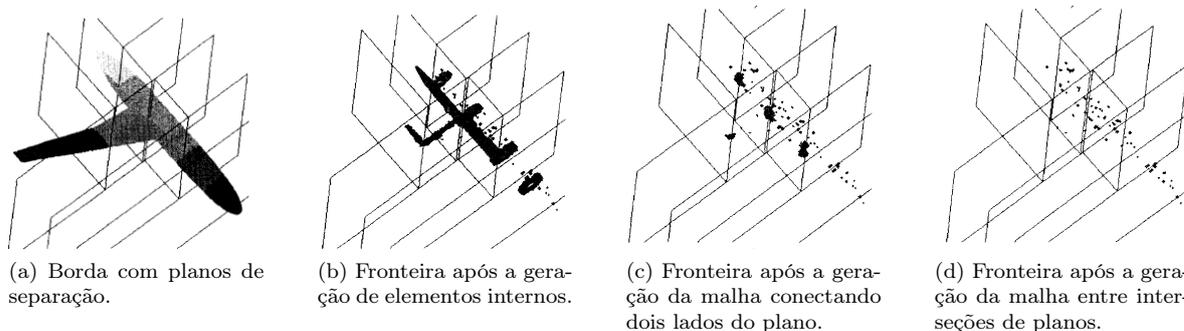
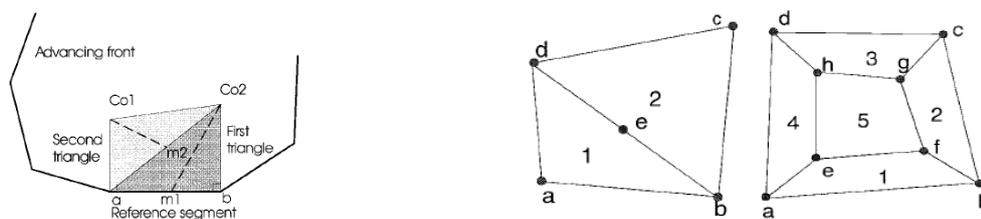


Figura 2.12 – Técnica de De Cougny e Shephard (1999).

Em Bern et al. (1999), é descrita uma técnica para a geração de *quadtrees* e de triangulações baseadas nelas. O artigo faz algumas suposições impraticáveis, como a geometria dada como entrada deve estar limitada a um quadrado de  $(0, 0)$  a  $(1, 1)$  e cujas coordenadas podem ser guardadas em uma única palavra de máquina. Com essas restrições, torna-se fácil (a nível de *bits*), ordenar-se os pontos pelas coordenadas, dividir-se um quadrado ao meio, e descobrir a menor célula da *quadtrees* que contenha dois determinados pontos da entrada. Assim, uma *quadtrees* é sequencialmente gerada de início, até que cada célula contenha, ao máximo, dois pontos da entrada, e é futuramente refinada em paralelo para que cada célula tenha somente um ponto da entrada e para que duas células vizinhas não tenham mais de um nível de distância (refinamento 2:1). Os triângulos são gerados a partir de deformações locais (rotações, escalas, translações e projeções em eixos), de testes de visibilidade e de divisão das arestas da entrada (nos pontos de interseção com as retas da *quadtrees*).

Em Topping e Cheng (1999), uma técnica de geração em paralelo de malhas quadrilaterais é apresentada, na qual cada quadrilátero é formado a partir da combinação de dois passos de geração de triângulos por avanço de fronteira (Figura 2.13a). Primeiramente, uma malha quadrilateral é gerada e cada elemento é considerado um subdomínio. Para tentar melhorar o balanceamento, o número de elementos de cada subdomínio é estimado e, caso um subdomínio venha a gerar uma grande quantidade de elementos, este é subdividido em dois ou em cinco novos subdomínios (Figura 2.13b). Cada subdomínio é enviado a um processo escravo por uma estratégia de demanda, que fica responsável por refinar as bordas desse quadrilátero e gerar a malha interna.



(a) Geração de um quadrilátero com dois triângulos.

(b) Subdivisão de um quadrilátero.

Figura 2.13 – Técnica em Topping e Cheng (1999).

Em Lämmer e Burghardt (2000), é apresentada uma técnica que gera elementos tanto triangulares quanto quadrilaterais. Dada a descrição do contorno, o eixo de maior momento de inércia passando pelo seu centro de gravidade é encontrado. Esse eixo é usado para dividir o domínio em dois, em um procedimento que pode ser aplicado recursivamente. A partir do eixo, uma aresta é formada, e os valores nos seus pontos extremos são interpolados dos valores dados como entrada. Quando o número de subdomínios for igual ao número de processadores, cada aresta é refinada (igualmente em subdomínios diferentes, para manter a conformidade), e uma malha de Delaunay é gerada em cada interior, concorrentemente (Figura 2.14). Os resultados apresentados não apresentam uma boa escalabilidade da técnica.

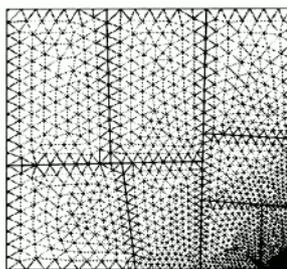


Figura 2.14 – Malha gerada pela técnica de Lämmer e Burghardt (2000).

Em Chrisochoides e Nave (2000), é apresentada uma técnica de refinamento de malhas de Delaunay que, ao final, já estão particionadas para a aplicação de métodos de elementos finitos. Uma vez discretamente decomposta a malha entre os processadores, pontos são inseridos simultaneamente, e uma busca em profundidade é feita para se determinar os elementos existentes influenciados pelo novo ponto. Se alguns desses elementos encontram-se em outros processadores, eles trocam mensagens e a parte da malha influenciada é transferida para o processador inicial, que insere o novo ponto (Figura 2.15a). A nova malha é, então, particionada entre os processadores envolvidos (Figura 2.15b), tentando garantir um bom balanceamento da carga. A busca pelos elementos influenciados e o procedimento de envio/reenvio da parte da malha influenciada podem custar muito tempo de comunicação que pode, por sua vez, limitar a escalabilidade da técnica.

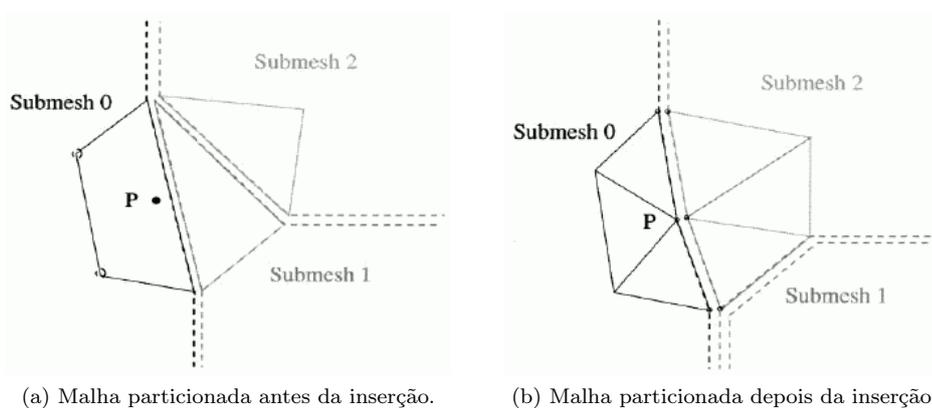


Figura 2.15 – Inserção paralela de um novo ponto (CHRISOCHOIDES; NAVE, 2000).

Em Coupez et al. (2000), é descrita uma técnica inicialmente desenvolvida para refinamento, mas modificada para a geração e a adaptatividade de malhas triangulares ou tetraédricas. A técnica é baseada em procedimentos locais baseados em otimização combinatória, ou seja, fórmulas de minimização e maximização que usam informações de volume do objeto e volume da malha gerada. A mesma otimização é utilizada para se determinar a geração, o reposicionamento ou a remoção de vértices internos, bem como a troca de arestas ou faces (*edge* ou *face swapping*). Para a geração da malha, uma malha qualquer, possivelmente não-conforme, é trivialmente gerada para a fronteira de entrada (Figura 2.16). O grafo representado por essa malha é particionado utilizando um procedimento também de otimização descrito no trabalho. Então, cada processo, responsável por uma partição, aplica as otimizações locais para a geração de uma malha conforme. Após isso, um reparticionamento, com migração de elementos entre processos, é feito para que as mesmas otimizações locais possam ser feitas nas arestas ou faces previamente de interface. Isso acontece iterativamente, até que uma malha totalmente conforme seja gerada.

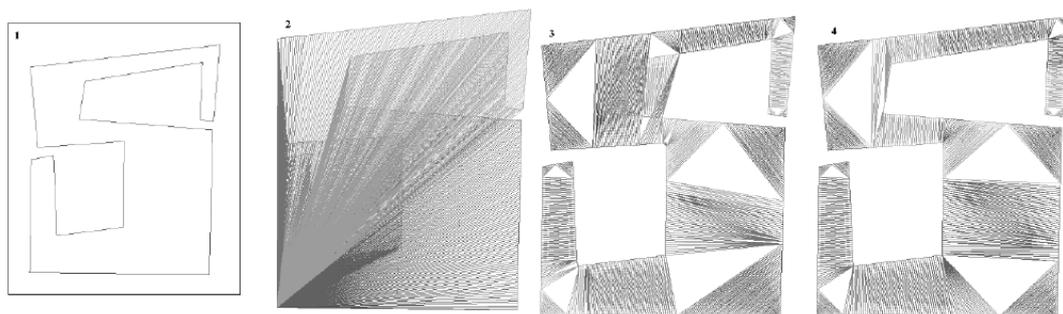
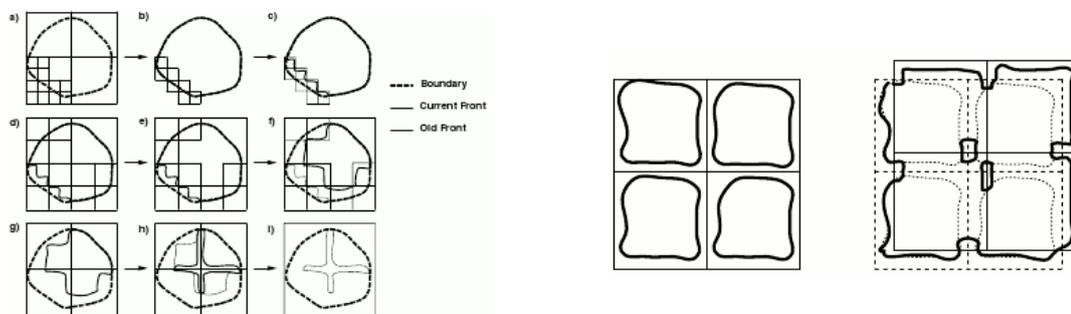


Figura 2.16 – Entrada, malha inicial não-conforme, malha intermediária não-conforme e malha final conforme da técnica em Coupez et al. (2000).

Em Moretti (2001), é descrita uma técnica de geração em paralelo de malhas tridimensionais com trincas, como parte de um sistema computacional completo de análise de elementos finitos. Inicialmente, uma malha grosseira é gerada para um determinado contorno, modificando-se os parâmetros de geração da malha. Esta malha é, então, particionada utilizando-se um algoritmo de particionamento de grafos. A fronteira entre os subdomínios obtidos é refinada de três maneiras com o objetivo de diminuir irregularidades: um elemento da fronteira que tiver mais elementos adjacentes em outra partição troca de subdomínio; uma face muito grande da fronteira tem sua maior aresta dividida ao meio; e um ponto da fronteira sofre uma suavização, modificando sua posição. Esses subdomínios são enviados aos outros processadores, que geram elementos internos independentemente, utilizando-se os parâmetros originais de geração da malha. Apesar de possuir decomposição discreta, esta técnica não refina o contorno, como os outros trabalhos aqui descritos. Além de poder introduzir artefatos no interior da malha final, esta técnica também sofre de desbalanceamento de carga.

Na técnica descrita em Löhner (2001), uma *octree* grosseira, com relação ao contorno dado como entrada, é gerada (Figura 2.17a). Então, as células que contêm a parte da fronteira que gerará os menores elementos são identificadas. Assim, partes da malha, correspondentes a cada célula, são geradas simultaneamente por avanço de fronteira, de maneira que cada parte da malha gerada não pode cruzar as extremidades da célula que a contém. Então, cada octante sofre um pequeno deslocamento na diagonal com o intuito de gerar mais elementos (Figura 2.17b). A razão desse deslocamento é eliminar quase todas as faces entre duas ou mais células e diminuir o tamanho da fronteira para o próximo passo. Então, a nova fronteira é encontrada, uma nova *octree* é construída para ela, e o procedimento é repetido, até que não seja mais possível gerar malha. O autor afirma, apesar de tudo, que essa técnica sofre de desbalanceamento de carga.



(a) *Octree* gerada para uma borda e passos do algoritmo (representação 2D, ou seja, uma *quadtree*).

(b) Deslocamento da *quadtree* para gerar mais malha (a linha escura é a fronteira).

Figura 2.17 – Técnica de Löhner (2001).

Em Larwood et al. (2003) é descrita uma técnica para se gerar malhas tetraédricas a partir de um modelo triangular de superfície. Inicialmente, um plano perpendicular ao eixo de maior comprimento é traçado, de forma que os dois subespaços definidos pelo plano tenham aproximadamente a mesma quantidade de faces. Para evitar a geração de elementos ruins, o posicionamento do plano de divisão pode ser modificado após sua determinação. Então, as arestas que cruzam o plano são identificadas e um aro de comprimento mínimo é criado com algumas dessas arestas e com algumas de suas adjacentes. Então, esse aro é projetado no plano de divisão, e uma triangulação de Delaunay é criada e reprojeta para o espaço 3D. Essa malha triangular separa o domínio em dois subdomínios desconexos, e o procedimento é repetido, concorrentemente e recursivamente, até que cada subdomínio tenha uma quantidade máxima de faces (Figura 2.18). Então, um grafo de vizinhança de subdomínios é criado e particionado. As faces internas a um agrupamento de subdomínios são eliminadas, e cada um desses novos subdomínios é enviado para um processador diferente, onde malhas tetraédricas serão criadas de forma independente.

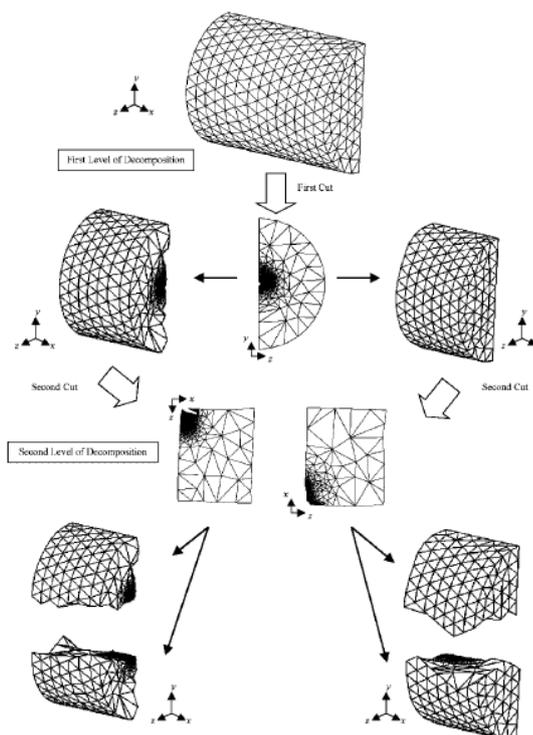


Figura 2.18 – Subdivisão recursiva do domínio em Larwood et al. (2003).

Em Kohout et al. (2005), são apresentadas técnicas de Delaunay de implementação prática para computadores de memória compartilhada. Os algoritmos utilizam um grafo direcionado acíclico para guardar a malha e auxiliar na busca por triângulos influenciados pela inserção de um novo ponto. Os algoritmos se diferenciam no acesso ao grafo, compartilhado entre os processadores, que o utilizam na busca e/ou inserção concorrente de um novo ponto. No primeiro algoritmo, os diversos processadores fazem a busca enquanto que um fica especialmente encarregado da inserção. No segundo, todos os processadores fazem a busca simultaneamente, mas a inserção, que pode ser feita por qualquer processador, é realizada com acesso exclusivo a todo o grafo. No terceiro algoritmo, assim como no segundo, todos os processadores estão aptos a fazer busca ou inserção simultaneamente. Entretanto, diferentemente do segundo algoritmo, a inserção é feita com acesso exclusivo a alguns nós do grafo, e não ao grafo todo. Ainda neste algoritmo, uma técnica de detecção de *deadlock* foi implementada, e os processadores envolvidos decidem qual vai de fato executar a inserção em detrimento dos outros, que retrocedem no procedimento (Figura 2.19). Além de poder ser custosa, a detecção de *deadlock* pode ocasionar que *threads* tenham que desfazer seus trabalhos diversas vezes, degradando o desempenho do algoritmo.

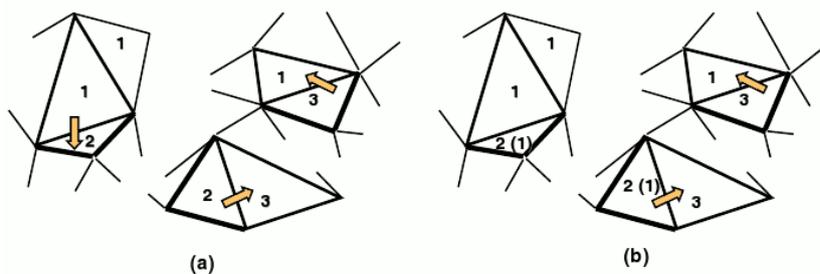


Figura 2.19 – Processador 1 “rouba” um triângulo do processador 2, quando as dependências, indicadas pelas setas, causam um *deadlock* (KOHOUT et al., 2005).

Em Chernikov e Chrisochoides (2005), uma técnica para refinamento em paralelo de triangulações de Delaunay é descrita, gerando malhas com gradação de tamanhos. Essa técnica é uma evolução da apresentada em Chernikov e Chrisochoides (2006), utilizando uma *quadtree* em vez de regiões quadradas uniformes. A inserção de pontos acontece simultaneamente em várias células-folha dessa *quadtree*, com uma restrição: se a inserção estiver acontecendo em uma determinada célula, então inserções não podem acontecer na zona de *buffer* dessa célula, que são suas células vizinhas e as vizinhas das vizinhas na outra direção (Figura 2.20). Essa restrição evita um possível conflito de o mesmo triângulo precisar ser eliminado por duas inserções concorrentes. Depois da geração de uma quantidade pré-determinada de elementos, a célula da *quadtree* é dinamicamente refinada para representar melhor o grau de discretização da malha após o refinamento, e as novas células-folha ficam passíveis de um futuro refinamento, caso necessário.

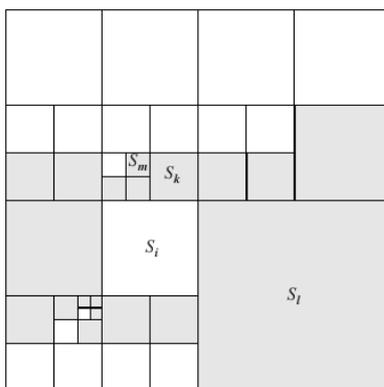


Figura 2.20 – Zona de *buffer* da célula  $S_i$ , na técnica descrita em Chernikov e Chrisochoides (2005).

Em Chernikov e Chrisochoides (2006), uma técnica de refinamento em paralelo de malhas de Delaunay é descrita, gerando malhas uniformes. Dados limites superiores para a área e para a razão do raio do circuncírculo (circunraio) pela menor aresta de um triângulo, o limite superior para o circunraio de um triângulo é encontrado. Então, a malha é sequencialmente refinada até esse limite ser obedecido. Posteriormente, a malha é dividida em regiões quadradas de forma que a inserção de um novo ponto nessa região não influencia outras regiões. Cada região é tratada concorrentemente, de forma que os triângulos mais internos dessa região terão seus circuncentros inseridos na malha, enquanto que os triângulos mais externos formam uma espécie de *buffer*, pois serão apenas modificados, e não refinados. Após o refinamento da região interna, há sincronização com as regiões vizinhas e uma parte da zona de *buffer* é refinada. Isso é feito até que toda a malha seja refinada. Dependendo dos parâmetros dados, essa técnica pode demorar muito no passo sequencial, o que pode limitar a escalabilidade do algoritmo.

Em Ivanov et al. (2006), uma técnica para a geração em paralelo de malhas tetraédricas é descrita, com enfoque no modo de decomposição do domínio. Assim como em Larwood et al. (2003), um plano é criado e as arestas da malha de superfície que cruzam o plano são identificadas. Um aro é encontrado a partir de algumas dessas arestas e de algumas de suas adjacentes. Este aro é projetado no plano encontrado e uma triangulação de Delaunay é construída e reprojeta para o espaço tridimensional. A diferença entre os dois trabalhos está no posicionamento do plano de divisão, que neste trabalho localiza-se no centro de massa do modelo e é perpendicular ao menor eixo principal de inércia. O mesmo procedimento é aplicado recursivamente em paralelo, mantendo-se o mesmo eixo ou encontrando um novo a cada passo. A quantidade de subdomínios gerados é maior que a de processos, para que haja um melhor balanceamento da carga. Alguns critérios para se decidir se o domínio deve ser subdividido ou se a malha tridimensional deve ser gerada fora implementados, como volume dos subdomínios, número de faces ou vértices da superfície de cada subdomínio ou momento de inércia.

Em Linardakis e Chrisochoides (2006), é apresentada uma técnica para a geração de uma triangulação de Delaunay em paralelo. Os circuncentros dos triângulos de uma malha de Delaunay inicial são encontrados, e aqueles que não estiverem dentro do triângulo são descartados. Dos restantes, alguns são selecionados e arestas são criadas destes circuncentros para dois dos vértices do seus triângulos, separando o domínio em dois (Figura 2.21a). A seleção é feita a partir da criação e contração (eliminação de vértices e arestas apropriados) de um grafo. Os vértices do grafo representam a área a ser triangularizada (futuros subdomínios), e as arestas representam a conexão entre essas áreas. A contração do grafo é feita de forma a igualar as áreas dos subdomínios e minimizar o tamanho das arestas de interface. O procedimento é aplicado recursivamente, em paralelo (Figura 2.21b), até que a quantidade de subdomínios seja suficientemente grande, para proporcionar um melhor balanceamento de carga. A cada aplicação recursiva, as arestas do modelo inicial e as de interface são subdivididas, para proporcionar a criação de uma malha com elementos menores e, portanto, maior. As subdivisões das arestas de interface são feitas, ainda, de forma a não precisarem ser futuramente subdivididas (Figura 2.22), podendo ocasionar em uma não-conformidade da malha final. Como os subdomínios são desconexos uns dos outros, a aplicação de refinamento de Delaunay em cada um deles pode ser feita em paralelo.

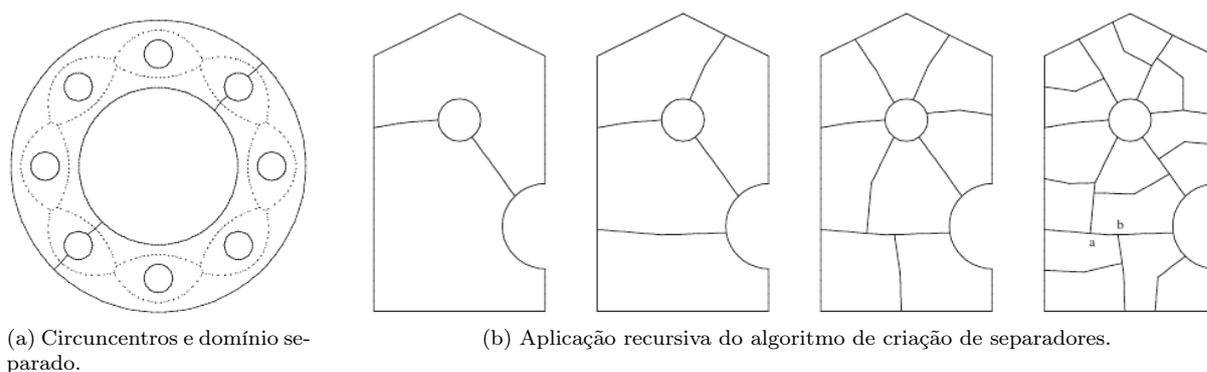


Figura 2.21 – Técnica de Linardakis e Chrisochoides (2006).

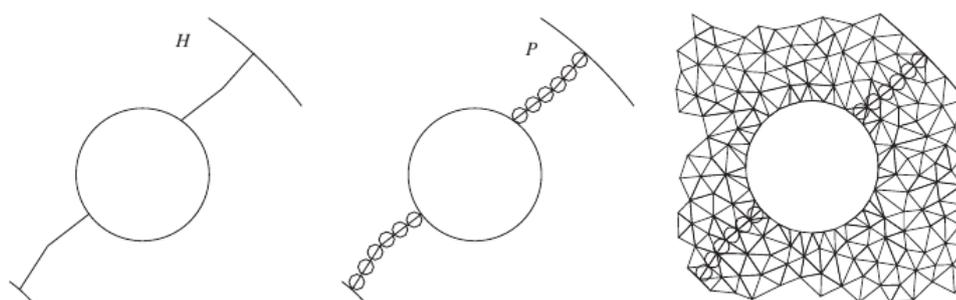


Figura 2.22 – Refinamento das arestas de interface (LINARDAKIS; CHRISOCHOIDES, 2006).

Em Rivara et al. (2006), é apresentada uma técnica para se refinar malhas em paralelo, a partir da subdivisão das arestas terminais da malha. Uma aresta terminal é uma aresta que é a mais longa dentre todos os tetraedros que a compartilham. Desse conceito, o procedimento de caminho de propagação de arestas mais longas é criado, em que, dado um tetraedro, sua aresta mais longa é determinada, juntamente com todos os tetraedros que a compartilham. Caso essa aresta não seja terminal desses tetraedros, a aresta mais longa dentre todos esses tetraedros é determinada e o mesmo procedimento é empregado. Assim, tem-se um caminho de arestas mais longas, desde a aresta mais longa do tetraedro original até uma aresta terminal. O refinamento é dado pela subdivisão das arestas terminais, até a aresta mais longa do

tetraedro original ser subdividida (um exemplo bidimensional pode ser visto na Figura 2.23). Como esse procedimento é bastante localizado, pode ser paralelizado a partir do particionamento da malha original. Em cada subdomínio, as arestas terminais internas são refinadas antes das arestas de interface. Como a subdivisão das arestas é sempre ao meio, a mesma subdivisão é aplicada em subdomínios vizinhos, garantindo a conformidade da malha sem comunicação entre processos.

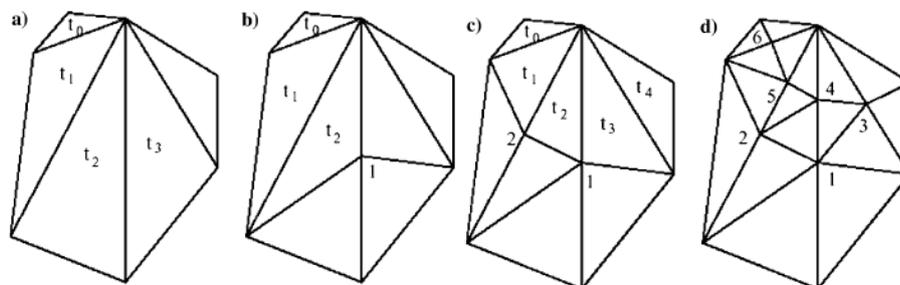


Figura 2.23 – Caminho de propagação de arestas de  $t_0$  e o refinamento empregado (RIVARA et al., 2006)

Em Pébay et al. (2007), é descrita uma técnica para a geração de malhas hexaedrais por varredura. Dadas duas malhas quadrilaterais de superfície e um mapeamento entre os seus quadriláteros, uma decomposição do domínio é criada, com interação do usuário, a partir da modificação do mesmo algoritmo de geração de malhas hexaedrais. Essa modificação faz com que sejam geradas somente as malhas de interface internas. Assim, hexaedros podem ser gerados em cada um dos subdomínios independentemente dos outros. A técnica utiliza ainda informações de mapeamento um-para-um ou um-para-muitos e estimativas da quantidade de hexaedros gerados em cada subdomínio.

Em Ito et al. (2007), é apresentada uma técnica que gera em paralelo uma malha tridimensional utilizando decomposição discreta. Inicialmente, uma malha grosseira é gerada sequencialmente por avanço de fronteira para um dado contorno. Depois, ainda sequencialmente, as arestas são subdivididas e as faces triangulares são refinadas por uma técnica de Delaunay. A malha é particionada utilizando um particionador de grafos (Figura 2.24), e a carga em cada partição é calculada. A carga total é balanceada e as malhas nas partições são geradas em paralelo por avanço de fronteira. Por ser de decomposição discreta, essa técnica também pode apresentar artefatos no interior da malha.

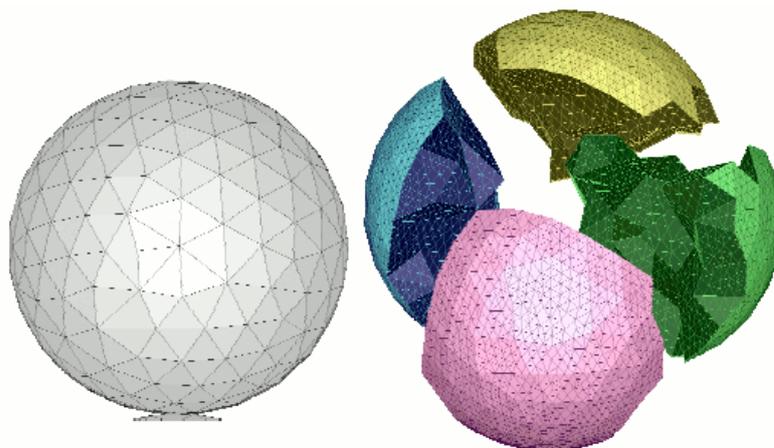


Figura 2.24 – Contorno e particionamento de sua malha grosseira refinada (ITO et al., 2007).

Em Głut e Jurczyk (2008) são descritas duas técnicas de decomposição de domínio para a geração em paralelo de malhas tetraédricas. Em ambas, uma malha de interface triangular é gerada criando dois subdomínios desacoplados, nos quais a mesma técnica pode ser aplicada recursivamente. Na primeira

técnica de decomposição (Figura 2.25a), o centro do eixo de maior comprimento é utilizado como fator para o posicionamento de um plano de partição. As arestas que cruzam este plano são identificadas e projetadas neste plano, onde a malha de interface é criada. Nos subdomínios finais, malhas de Delaunay são criadas simultânea e independentemente. Na segunda técnica de decomposição (Figura 2.25b), uma malha tetraédrica de Delaunay grosseira é gerada, e é particionada utilizando o mesmo critério da primeira técnica, ou seja, o centro do eixo de maior comprimento. A região da malha neste plano de partição é, então, refinada, para levar em conta os parâmetros de tamanho de malha originais. A parte da malha tetraédrica que corta o plano de partição é o que define a malha de interface, e as duas sub-malhas são refinadas em paralelo, caso os subdomínios sejam finais.

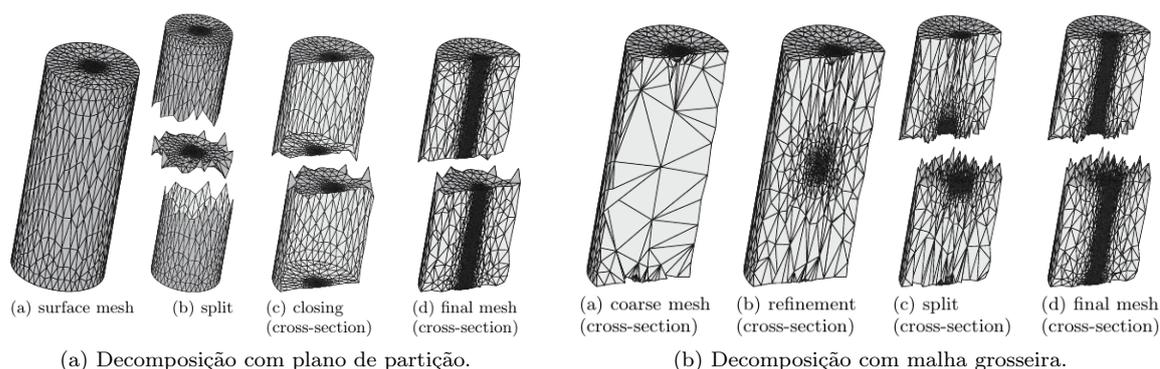


Figura 2.25 – Formas de decomposição do domínio em Głut e Jurczyk (2008).

Na técnica apresentada em Zagaris et al. (2009), uma malha tridimensional é gerada em paralelo por avanço de fronteira. Dado o contorno inicial, o algoritmo encontra um plano que passa pelo seu centro de massa, e gera uma malha tetraédrica considerando como fronteira somente os triângulos que cortam esse plano (Figura 2.26a). Assim, duas sub-regiões desconexas são criadas, onde a mesma técnica de particionamento é aplicada novamente (Figura 2.26b), em diferentes processadores. Quando um determinado número de subdivisões é atingido, o mesmo algoritmo de avanço de fronteira é empregado nas regiões internas para gerar a malha toda. Essa método não apresentou uma boa escalabilidade. Em Pirzadeh e Zagaris (2009), o mesmo trabalho é descrito, porém o enfoque do artigo é na forma de decomposição do domínio.

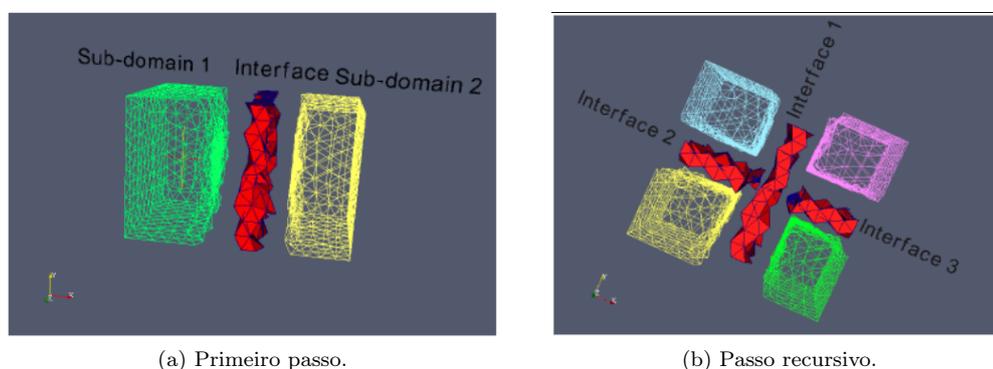


Figura 2.26 – Malhas geradas particionando o domínio (ZAGARIS et al., 2009).

Em Chrisochoides et al. (2009), várias técnicas de geração em paralelo de malhas de Delaunay são agrupadas para a formação de um único programa, a ser executado em uma grade de *clusters*. Inicialmente, uma decomposição do domínio é feita com a inserção de arestas de interface, de acordo com a técnica apresentada em Linardakis e Chrisochoides (2006). Cada subdomínio é destinado a um *cluster*, e as malhas

são geradas sem comunicação, que poderia custar caro. Cada subdomínio, exclusivo de um *cluster*, é posteriormente subdividido, mas a inserção de pontos internos pode ocasionar em comunicação (trabalho parecido com o de Chrisochoides e Nave (2000)). Essa comunicação acontece somente entre os nós de um mesmo *cluster* e, portanto, não é caro. Cada processo, ainda, insere vários pontos simultaneamente utilizando paralelismo de memória compartilhada Chernikov e Chrisochoides (2005).

Em Batista et al. (2010), uma técnica para a geração em paralelo de triangulações de Delaunay é apresentada. Inicialmente, uma malha de Delaunay é sequencialmente gerada com alguns dos pontos. Os vértices restantes são ordenados e particionados espacialmente (em paralelo) utilizando um algoritmo descrito no próprio trabalho. A inserção de vários desses vértices acontece simultaneamente, com o auxílio de travas nos vértices ou nos triângulos da triangulação existente. Quatro estratégias de utilização das travas são descritas para evitar que o mesmo triângulo seja apagado por *threads* diferentes. Às *threads*, são atribuídas prioridades para se evitar *deadlock*, e uma estratégia de roubo de trabalho (*work stealing*) também foi implementada para haver balanceamento de carga.

Em Yilmaz et al. (2010), são propostos três métodos para a geração em paralelo de malhas tetraédricas para domínios simples. No primeiro (Figura 2.27), dada uma geometria, uma malha tetraédrica grosseira é gerada sequencialmente e decomposta usando uma estrutura de árvore binária (BSP) alinhada aos eixos. A malha grosseira é descartada e malhas de interface são geradas nos planos de decomposição. Então, malhas tetraédricas são geradas em paralelo em cada subdomínio, e os elementos da malha final são redistribuídos de acordo com um particionador de malha, para que a malha final esteja balanceada entre os processos. Esse método apresentou problemas em regiões da geometria próximas aos planos de decomposição. No segundo método, uma malha grosseira é gerada sequencialmente para a geometria dada, e distribuída usando um particionador de grafos. Os tetraedros da malha grosseira são, então, refinados em paralelo. Esse método apresentou problemas de qualidade, uma vez que tetraedros ruins presentes na malha grosseira geram elementos ruins na malha refinada. Assim como no segundo método, no terceiro, uma malha grosseira é gerada sequencialmente e distribuída usando um particionador de malhas. Em cada processo, as interfaces da sua partição são identificadas e refinadas, e uma malha tetraédrica é gerada. Em nenhum dos métodos, entretanto, uma boa escalabilidade foi observada, principalmente devido às etapas sequenciais.

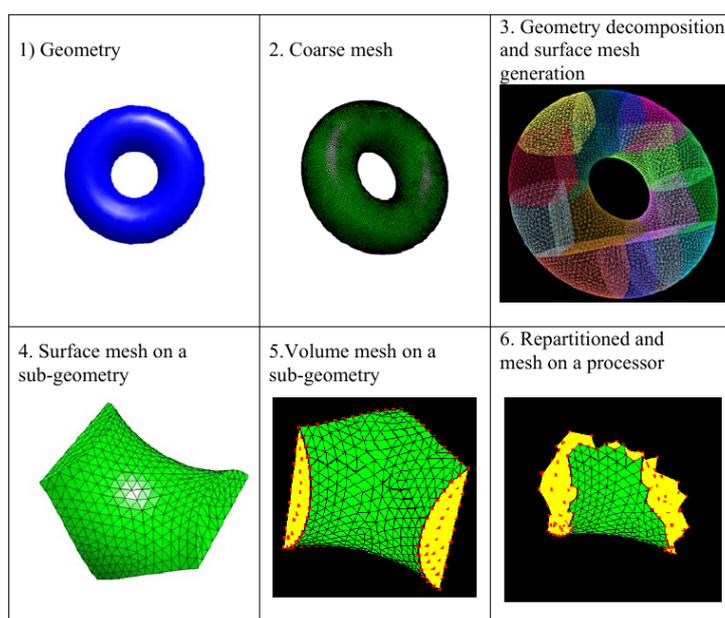


Figura 2.27 – Geração de uma malha tetraédrica pelo primeiro método (YILMAZ et al., 2010).

Em Camata e Coutinho (2010), Camata e Coutinho (2013), é descrita uma técnica para a geração de uma malha de *octree*. Inicialmente, a *octree* é guardada na memória de forma linear, a partir de um código que combina a representação binária do ponto mínimo e do nível de cada célula da *octree*. Esta representação linear pode ser dividida em qualquer ponto, sem depender da hierarquia da *octree*. Assim, ela é dividida entre os processos, que a refinam em paralelo para adaptá-la à superfície triangular de entrada. Posteriormente, ela é redistribuída, e o refinamento 2:1 é aplicado, também em paralelo. Este refinamento é dividido em duas etapas. Na primeira, somente são refinadas as células vizinhas à uma célula desbalanceada que estão presente no próprio processo. Na segunda etapa, há comunicação para o refinamento das células vizinhas pertencentes a outros processos. Por último, é gerada a malha não-estruturada e não-conforme baseada na estrutura linear da *octree*.

Em Simonovski e Cizelj (2011), é apresentada uma técnica para a geração em paralelo de malhas tetraédricas para modelos de visualização volumétrica ou modelos analíticos. No caso de modelos de visualização volumétrica (o enfoque do trabalho) uma malha de superfície é criada para cada isovalor, para modelos analíticos, é criada uma tesselação de Voronoi. As malhas de superfície geradas são imperfeitas, e passam por simplificações e alterações para garantir a não-penetração, eliminação de regiões desconexas e de eventuais sulcos presentes na borda do modelo. Essas malhas passam, ainda, por um refinamento em paralelo, eliminando triângulos de má qualidade e criando uma malha uniforme para cada superfície. Então, para cada subdomínio, definido por um isovalor, uma malha tetraédrica é gerada. Isso é paralelizado, uma vez que as malhas de superfície nas interfaces são conformes.

Em Wu et al. (2011), é descrita uma técnica para gerar em paralelo triangulações de Delaunay pelo paradigma de divisão e conquista. Pela quantidade de vértices da entrada ser muito grande, o trabalho utiliza uma abordagem de fluxo (*stream*), onde os dados só permanecem na memória enquanto são utilizados, e são descarregados para o disco rígido assim que possível. Os vértices são dados em blocos regulares, e uma *kd-tree* é utilizada para se particionar esses blocos (Figura 2.28). Cada *thread* fica responsável por gerar uma triangulação de Delaunay nos vértices de um bloco, e os triângulos cujos circuncírculos não tocam a borda do bloco são gravados no disco (Figura 2.29a). A etapa de conquista, também feita em paralelo, consiste em gerar uma triangulação de Delaunay juntando dois blocos, e em gravar no disco os triângulos com as mesmas características (Figura 2.29b).

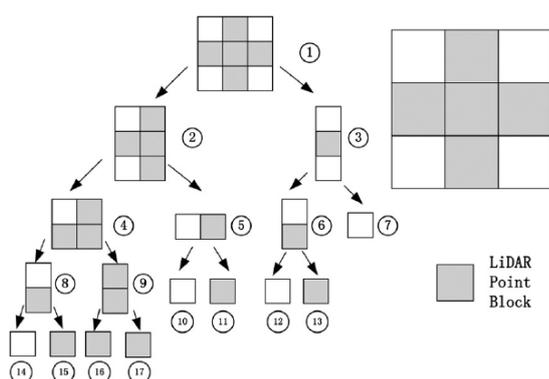


Figura 2.28 – *Kd-tree* para particionar os blocos (WU et al., 2011).

Em Panitanarak e Shontz (2011), é descrita uma técnica de decomposição de domínio para geração de malhas bidimensionais que satisfaz uma restrição de qualidade nas arestas de partição. Essa restrição é de que o ângulo entre duas arestas adjacentes que particionam o domínio em subdomínios (arestas separadoras) deve ser pelo menos 60 graus. Inicialmente, uma malha de Delaunay é sequencialmente gerada para o domínio todo, com a restrição de que o ângulo mínimo de um triângulo deve ser pelo menos 30 graus. Essa malha é decomposta em um determinado número de subdomínios, dado pelo usuário, de

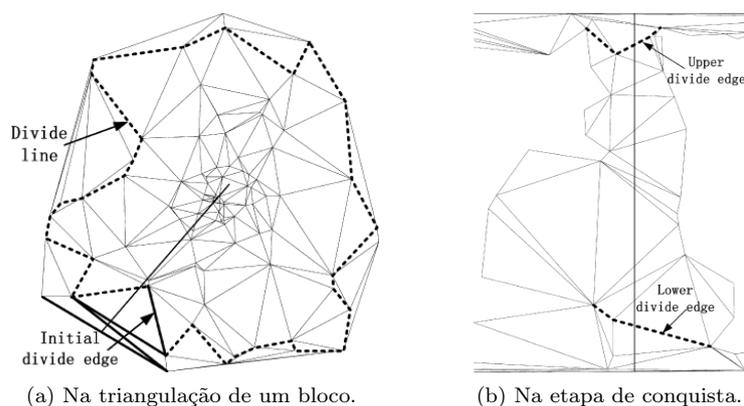


Figura 2.29 – Linhas divisórias dos triângulos que são descarregados para o disco (WU et al., 2011).

forma a balancear a área dos subdomínios. Assim, as arestas separadoras são arestas da triangulação, que são encontradas e tratadas para satisfazer o requisito de ângulo mínimo de 60 graus. Esse requisito somente é quebrado quando um triângulo tem duas arestas separadoras, que pode ser um dos seguintes casos: as duas arestas separam dois subdomínios e nenhuma delas é da borda do modelo; as duas arestas separam dois subdomínios e uma delas é da borda do modelo; as duas arestas separam três ou mais subdomínios. No primeiro caso, as duas arestas separadoras são substituídas pela terceira aresta do mesmo triângulo. No segundo caso, a mesma coisa é feita caso a terceira aresta satisfaça a restrição de ângulo mínimo; caso contrário, o ponto médio da aresta da borda é inserido e ligado ao outro vértice do triângulo, substituindo a aresta separadora que não pertencia à borda. No terceiro caso, o incentro do triângulo é inserido e ligado aos seus três vértices, substituindo as duas arestas separadoras originais (Figura 2.30). Com essas modificações, o ângulo mínimo entre separadores de 60 graus é satisfeito. Então, uma malha de Delaunay com restrição é gerada em cada subdomínio.

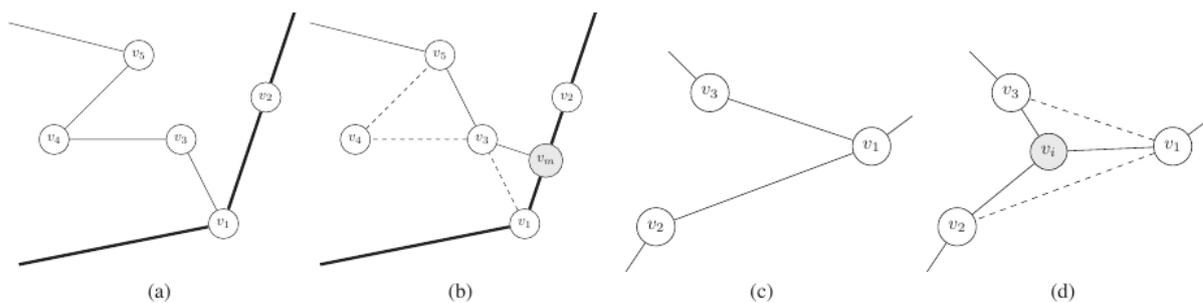
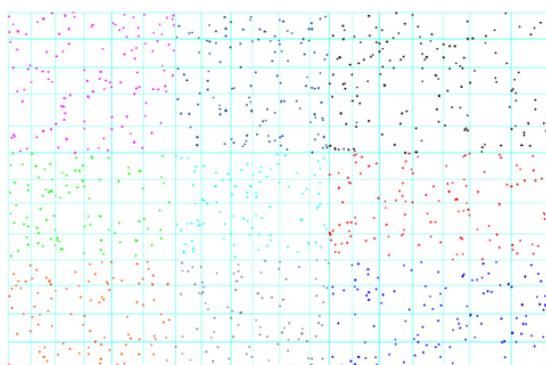


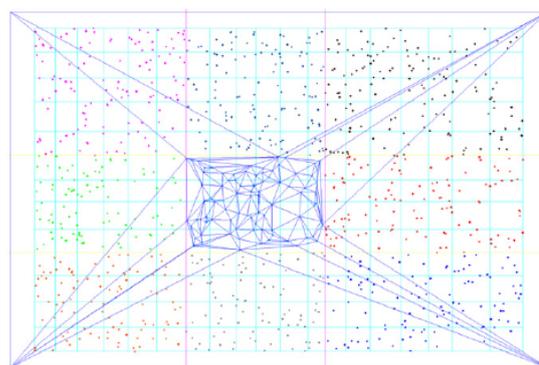
Figura 2.30 – Modificações para garantir o ângulo mínimo de 60 graus. Os dois primeiros casos (a) são resolvidos em (b), e o terceiro caso (c) é resolvido em (d) (PANITANARAK; SHONTZ, 2011).

Em Lo (2012a), um algoritmo para a geração em paralelo de uma malha triangular quase Delaunay de um conjunto de pontos é descrita. Inicialmente, uma caixa delimitadora é construída englobando todos os pontos. Essa caixa é dividida em células retangulares, não necessariamente do mesmo tamanho, de forma que cada célula contenha aproximadamente o mesmo número de pontos, dado pelo usuário. Essas células são, então, agrupadas em zonas retangulares, cada uma de responsabilidade de um processo (Figura 2.31a). Em cada zona, uma malha composta de dois triângulos é criada englobando todo o domínio, e os pontos de cada célula são inseridos aleatoriamente, de forma que os triângulos gerados não se propaguem para outra zona (Figura 2.31b). Então, os circuncírculos dos triângulos da borda da

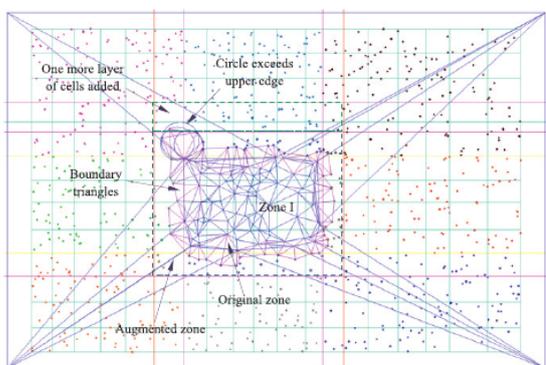
zona são encontrados e, dependendo dos seus tamanhos, uma ou duas camadas de células adjacentes, pertencentes a outras zonas, são adicionadas, e a inserção é propagada para os triângulos satisfazerem o critério de Delaunay (Figura 2.31c). Com isso, alguns triângulos repetidos foram gerados e é necessário um procedimento de remoção de elementos duplicados. Neste procedimento, as zonas responsáveis pelos vértices de um triângulo são identificadas, e a zona de menor identificador se torna a responsável por aquele triângulo. Assim, todos os triângulos repetidos ficam em uma única zona, e o processo relativo a ela elimina os elementos duplicados. Ao final, uma malha cujos triângulos respeitam o critério de Delaunay é gerada (Figura 2.31d). Entretanto, nem todo o fecho convexo do conjunto de pontos é triangularizado, descaracterizando uma triangulação de Delaunay sem restrições de borda. A técnica foi expandida para três dimensões em Lo (2012b).



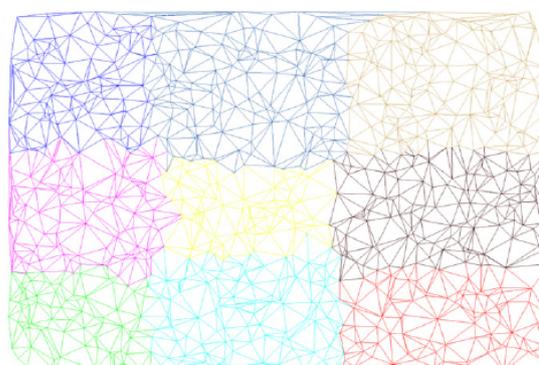
(a) Decomposição dos pontos em 9 zonas (células contém aproximadamente 4 pontos).



(b) Malha de Delaunay dentro da zona central.



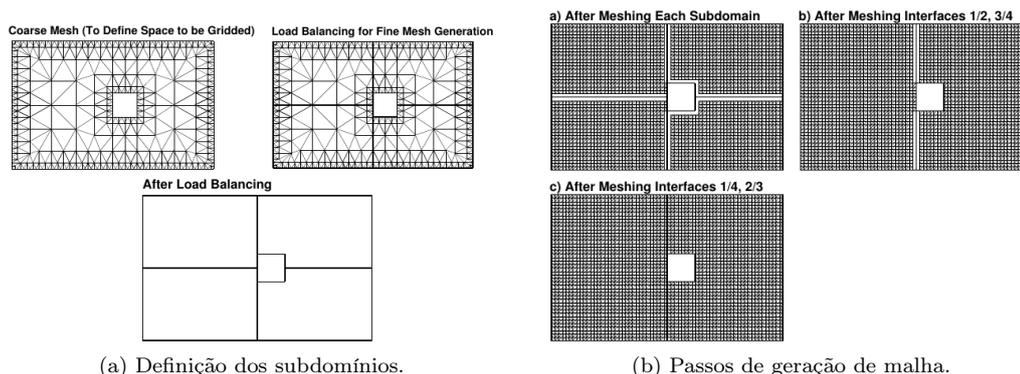
(c) Zona central aumentada e triângulos de borda.



(d) Malha final particionada.

Figura 2.31 – Geração da malha pela técnica de Lo (2012a).

Em Löhner (2012), Löhner (2014), é descrita uma técnica para gerar em paralelo tetraedralizações pela técnica de avanço de fronteira. Inicialmente, uma malha grosseira é gerada e particionada utilizando um particionador de grafos ou malhas (Figura 2.32a). Neste particionamento, o número de subdomínios é igual à quantidade de total de unidades de processamento utilizados, que é multiplicação do número de núcleos de cada processo pelo número de processos. Além disso, o particionamento é feito de forma que a estimativa do número de elementos gerados em cada subdomínio seja a mesma. Uma vez determinados os subdomínios, em cada um deles, uma malha refinada é gerada de forma a não cruzar as bordas da malha grosseira que o define (Figura 2.32b). Posteriormente, as interfaces entre dois subdomínios são distribuídas entre os processos por um esquema de coloração de grafos, e as malhas nessas regiões são geradas. Um terceiro passo de geração de malhas ainda é necessário, nas interfaces que conectam mais de dois subdomínios. Ao final, a malha é redistribuída entre os processos, que aplicam uma melhoria na malha gerada.



(a) Definição dos subdomínios. (b) Passos de geração de malha.  
 Figura 2.32 – Decomposição do domínio e geração de malha em Löhner (2012).

Em Chen et al. (2012), é descrita mais uma técnica para gerar em paralelo tetraedralizações a partir do posicionamento de um plano de decomposição e da geração de uma malha bidimensional neste plano (Figura 2.33). O posicionamento do plano é dado por uma média ponderada dos tamanhos da aresta e pelo eixo inercial principal. As arestas do contorno que cruzam este plano de partição são encontradas e algumas delas são eliminadas para se formar um ou vários polígonos. Estes polígonos são projetados no plano de partição e uma malha bidimensional é gerada. Na transformação dos elementos gerados para o espaço tridimensional para gerar a malha de interface, é possível que esta intercepte a borda do modelo. O trabalho descreve vários tratamentos da malha de interface para a eliminação dessas interseções, como troca de diagonais, movimentação de vértices e movimentação de triângulos. É possível que, depois desses tratamentos, a malha de interface continue interceptando a borda e, neste caso, o plano de partição sofre um deslocamento, e o procedimento é repetido. A geração das malhas de interface é feita somente no processo mestre, até que um número pré-definido de subdomínios desconexos sejam gerados. Então, cada processo escravo recebe dinamicamente uma quantidade de subdomínios e gera uma malha tetraédrica de Delaunay em cada uma delas. Finalmente, a malha como um todo é particionada com um particionador de grafos, e melhorada em paralelo. Por conta das malhas de interface e da melhoria, essa técnica não é de Delaunay.

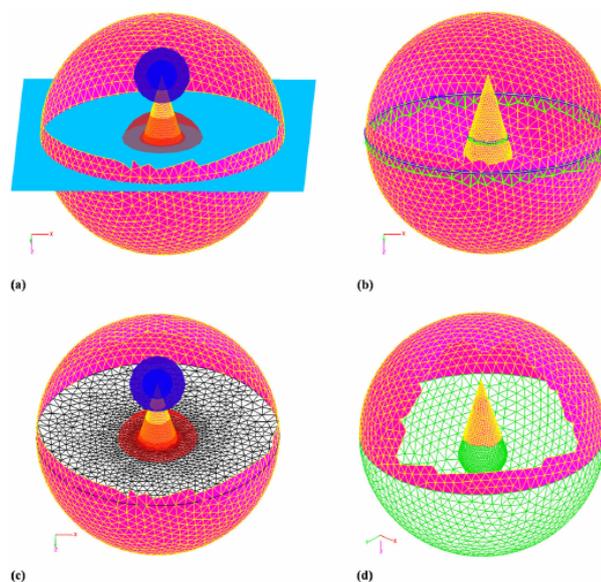


Figura 2.33 – Plano de partição, arestas da interface, malha de interface e subdomínios da técnica de Chen et al. (2012).

Em Freitas et al. (2013), é apresentada uma técnica para gerar triangulações por avanço de fronteira com decomposição contínua *a posteriori*. Dado um contorno, é gerada uma *quadtrees* que represente a função de densidade dos tamanhos dos elementos no interior do domínio. Esta *quadtrees* refinada serve como estimativa de carga para a decomposição do domínio, que é também feita utilizando-se uma outra *quadtrees* (Figura 2.34a), esta mais grosseira que a anterior. A geração da malha em cada subdomínio é feita por avanço de fronteira e, a partir de deslocamentos alinhados aos eixos (Figura 2.34b), as malhas de interface são criadas. Além disso, os deslocamentos permitem que a fronteira avance para o interior de subdomínios que, inicialmente, não continham fronteira. A partir da alternância entre deslocamentos e geração de malha, feita por meio de sincronização global, a malha triangular é completamente gerada, e melhorias são aplicadas nas malhas de interface. As malhas geradas apresentaram boa qualidade, porém a técnica não mostrou-se escalável por conta dos deslocamentos. O presente trabalho é uma extensão de Freitas et al. (2013), porém tratando as suas deficiências.

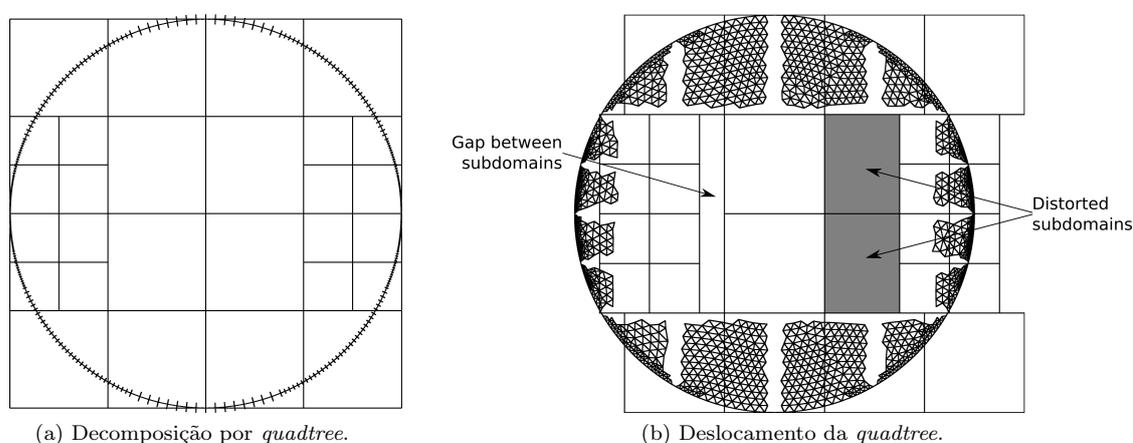


Figura 2.34 – Decomposição por *quadtrees* e deslocamento da *quadtrees* na técnica de Freitas et al. (2013).

## 2.4 Classificação dos trabalhos relacionados

A Figura 2.35 mostra a quantidade de trabalhos relacionados por ano.

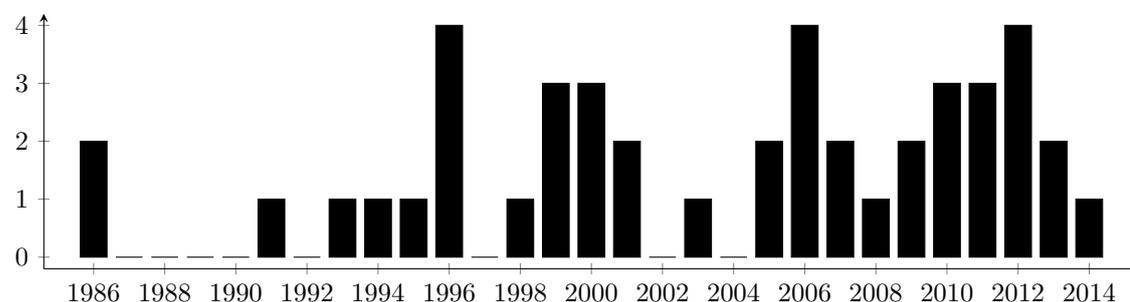


Figura 2.35 – Quantidade de trabalhos por ano.

A Tabela 2.2 agrupa os trabalhos listados na Seção 2.3 de acordo com as classificações apresentadas na Seção 2.2. A grande quantidade de trabalhos classificados como arbitrários deve-se ao fato de que muitos deles geram malhas de Delaunay nos subdomínios, porém a malha na região de interface não satisfaz os critério do círculo/esfera.

Tabela 2.2 – Classificação dos trabalhos.

	Decomposição Discreta	Decomposição Contínua
Avanço de Fronteira	Khan e Topping (1991) Wilson e Topping (1998) Topping e Cheng (1999) Moretti (2001) Ito et al. (2007) Löhner (2012) <sup>2</sup> Löhner (2014) <sup>2</sup>	De Cougny et al. (1994) <sup>1</sup> Globish (1995) De Cougny et al. (1996) <sup>1</sup> De Cougny e Shephard (1999) Löhner (2001) Zagaris et al. (2009) Löhner (2012) <sup>2</sup> Freitas et al. (2013) Löhner (2014) <sup>2</sup>
Delaunay	Chrisochoides e Nave (2000) <sup>3</sup> Linardakis e Chrisochoides (2006) <sup>3</sup> Chrisochoides et al. (2009) <sup>3</sup> Panitanarak e Shontz (2011)	Teng et al. (1993) Okusanya e Peraire (1996) Kohout et al. (2005) Chernikov e Chrisochoides (2005) <sup>3,4</sup> Chernikov e Chrisochoides (2006) <sup>4</sup> Chrisochoides et al. (2009) <sup>3</sup> Batista et al. (2010) Wu et al. (2011)
Arbitrária	Merks (1986) ElGindy (1986) Hodgson e Jimack (1996) Wu e Houstis (1996) <sup>5</sup> Coupez et al. (2000) Rivara et al. (2006) Yilmaz et al. (2010) <sup>6</sup>	Wu e Houstis (1996) <sup>5</sup> Bern et al. (1999) Lämmer e Burghardt (2000) Larwood et al. (2003) Ivanov et al. (2006) Pébay et al. (2007) Głut e Jurczyk (2008) Yilmaz et al. (2010) <sup>6</sup> Camata e Coutinho (2010) <sup>7</sup> Simonovski e Cizelj (2011) Lo (2012a) <sup>8</sup> Lo (2012b) <sup>8</sup> Chen et al. (2012) Camata e Coutinho (2013) <sup>7</sup>

## 2.5 Considerações finais

Este capítulo apresentou algumas técnicas paralelas de geração de malhas que foram classificadas de acordo com a técnica de geração de malha e com a maneira como a decomposição do domínio é realizada.

Apesar dos bons resultados teóricos apresentados pelas malhas de Delaunay, elas nem sempre têm boa qualidade em regiões próximas ao contorno do objeto. Nesse quesito, os algoritmos de avanço de fronteira, em geral, se sobressaem, uma vez que estes geram bons elementos no contorno, deixando os

<sup>1</sup>De Cougny et al. (1996) é uma melhoria de De Cougny et al. (1994).

<sup>2</sup>Löhner (2014) é uma melhoria de Löhner (2012), que tem características tanto de decomposição discreta quanto de decomposição contínua.

<sup>3</sup>Chrisochoides et al. (2009) é uma combinação de Chrisochoides e Nave (2000), Linardakis e Chrisochoides (2006) e Chernikov e Chrisochoides (2005) e, portanto, tem características tanto de decomposição discreta quanto de decomposição contínua.

<sup>4</sup>Chernikov e Chrisochoides (2005) é uma melhoria de Chernikov e Chrisochoides (2006), apesar de ter sido publicado anteriormente.

<sup>5</sup>Wu e Houstis (1996) depende da escolha do usuário, e funciona tanto com decomposição discreta quanto com decomposição contínua.

<sup>6</sup>Yilmaz et al. (2010) depende da escolha do usuário, e funciona tanto com decomposição discreta quanto com decomposição contínua.

<sup>7</sup>Camata e Coutinho (2013) é uma melhoria de Camata e Coutinho (2010).

<sup>8</sup>Lo (2012b) é a extensão de Lo (2012a) para três dimensões.

elementos ruins para o interior do objeto. Assim, o trabalho aqui descrito gera uma malha por avanço de fronteira.

Os algoritmos de decomposição contínua têm diversas vantagens sobre os de decomposição discreta (Tabela 2.1) como, por exemplo, um controle melhor sobre a transição entre elementos de tamanhos diferentes. Além disso, a decomposição discreta geralmente requer que o contorno seja refinado, modificando assim o contorno original, o que pode quebrar os requisitos de um método de elementos finitos. Por essas e pelas razões descritas na Seção 2.2.2.3, a decomposição contínua *a posteriori* foi utilizada neste trabalho.

O presente trabalho tem algumas similaridades com alguns dos trabalhos mencionados neste capítulo, porém, com diferenças significativas. Assim como em Wu e Houstis (1996), Lämmer e Burghardt (2000), Larwood et al. (2003), Ivanov et al. (2006), Głut e Jurczyk (2008), Zagaris et al. (2009), Yilmaz et al. (2010), Wu et al. (2011), Chen et al. (2012), este trabalho subdivide recursivamente o domínio ao meio, entretanto, levando em conta a estimativa da carga, em vez da geometria. Enquanto o trabalho em Wu et al. (2011) utiliza uma estrutura em forma de árvore chamada *kd-tree*, este trabalho utiliza uma estrutura de particionamento espacial binário (BSP, do inglês *Binary Spatial Partitioning*), assim como em Yilmaz et al. (2010). A BSP pode ser vista como uma generalização da *kd-tree*, e dá mais liberdade no posicionamento do plano de decomposição.

Similarmente a De Cougny et al. (1996), Zagaris et al. (2009), Freitas et al. (2013), uma *quadtree/octree* é utilizada para determinar o posicionamento do plano de decomposição. Neste trabalho, esta *quadtree/octree* é usada como forma de estimar a carga, assim como em Zagaris et al. (2009). Entretanto, aqui, esta estimativa de carga é analisada, algo que é feito somente em Freitas et al. (2013).

Diferentemente de Wu e Houstis (1996), Lämmer e Burghardt (2000), Larwood et al. (2003), Ivanov et al. (2006), Zagaris et al. (2009), Yilmaz et al. (2010), Chen et al. (2012), e assim como De Cougny e Shephard (1999), Löhner (2001), Wu et al. (2011), Chen et al. (2012), Löhner (2012), Freitas et al. (2013), Löhner (2014), as malhas de interface entre subdomínios são geradas *a posteriori*. Analogamente a Wu et al. (2011), as malhas de interface são geradas enquanto o algoritmo sobe na estrutura de árvore da BSP. Diferentemente de Wu et al. (2011) e similarmente a Löhner (2001), Zagaris et al. (2009), Löhner (2012), Freitas et al. (2013), Löhner (2014), as malhas são geradas utilizando-se somente uma técnica de avanço de fronteira.

Dada a complexidade de implementação de uma técnica de geração de malhas, não é comum observar comparações entre os seus resultados nos trabalhos relacionados, tanto nas técnicas sequenciais quanto nas paralelas.

Nas técnicas paralelas, a maioria dos trabalhos não apresenta uma métrica de estimativa de carga, na qual a decomposição é baseada. Nos trabalhos onde há alguma estimativa de carga, esta é geralmente feita após a decomposição do domínio. Para evitar grandes desbalanceamentos, são utilizadas técnicas de *overdecomposition*, onde cria-se mais subdomínios que unidades de processamento, para que, de uma maneira geral, todas elas levem aproximadamente o mesmo tempo de execução. Outra forma utilizada para evitar desbalanceamento é limitar a quantidade de elementos gerados em um subdomínio, para forçá-lo a condizer à estimativa feita. Além disso, nos trabalhos onde há estimativa de carga, sua precisão não é analisada.

Assim, as principais diferenças entre o presente trabalho e os trabalhos acima relacionados são a forma de decomposição, que é uma BSP baseada na carga estimada, a análise da estimativa de carga, apresentada no Capítulo 4 e, sobretudo, o cumprimento de todos os requisitos citados na Seção 1.2, especialmente por não precisar fazer nenhum tipo de distinção ao tratar trincas ou buracos no domínio.

## 3 TÉCNICA PROPOSTA

### 3.1 Introdução

O objetivo principal deste trabalho é desenvolver uma técnica de geração de malhas em paralelo para computadores de memória compartilhada, distribuída ou híbrida, que funcione em modelos com ou sem buracos ou trincas, que gere uma malha de boa qualidade, e que apresente um bom balanceamento de carga, bem como um bom *speed-up*.

A técnica paralela é baseada em uma técnica de geração de malha sequencial existente de avanço de fronteira (CAVALCANTE-NETO, 1994; CAVALCANTE-NETO, 1998; MIRANDA et al., 1999; CAVALCANTE-NETO et al., 2001), e a implementação utilizada neste trabalho corresponde a uma desenvolvida pelo *Cornell Fracture Group* (CFG), da Universidade de Cornell, um grupo parceiro em diversas pesquisas de geração de malha.

A técnica sequencial, que será brevemente explicada na Seção 3.2, deve satisfazer a quatro requisitos, que motivaram o seu desenvolvimento:

1. A malha gerada deve respeitar a fronteira dada como entrada, ou seja, a técnica não pode modificá-la sob nenhuma circunstância;
2. A malha gerada deve ser de boa qualidade, embora nenhuma métrica de forma ou tamanho dos tetraedros seja explicitamente utilizada;
3. A malha deve fornecer uma boa transição entre regiões muito refinadas, com muitos tetraedros pequenos, e regiões pouco refinadas, com poucos tetraedros grandes;
4. A técnica deve tratar fronteiras com trincas, e gerar malhas respeitando-as. As trincas são descritas como regiões de volume nulo, ou seja, existem faces/vértices diferentes, porém geometricamente coincidentes (Figura 3.1).

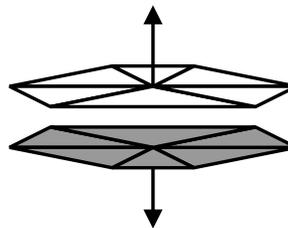


Figura 3.1 – Exemplo de trinca. A distância entre vértices e faces mostrada é apenas esquemática e não existe na realidade.

A técnica paralela foi projetada utilizando-se decomposição contínua, pois um dos requisitos da técnica sequencial, e, portanto, da técnica paralela, é que a fronteira dada como entrada deve ser respeitada, algo que as estratégias de decomposição discreta não satisfazem. Além disso, a abordagem de geração das malhas de interface *a posteriori* foi escolhida para satisfazer ao segundo requisito, de que a malha gerada deve ser de boa qualidade. Os outros requisitos são atendidos pela própria técnica sequencial, que é empregada pela técnica paralela. A paralelização da técnica sequencial será descrita na Seção 3.3.

É importante observar que, apesar de este trabalho focar na geração de uma malha tridimensional, a técnica paralela é genérica, e funciona tanto para duas quanto para três dimensões. Assim, sempre que for mais didático, o caso bidimensional será utilizado para exemplificar alguns aspectos da técnica. Além disso, a técnica paralela funciona igualmente para arquiteturas paralelas de memória compartilhada, distribuída ou híbrida.

## 3.2 Técnica sequencial

A entrada para a técnica é uma discretização da fronteira de um objeto que pode conter trincas ou buracos em seu interior, chamada de contorno ou fronteira inicial. Tal discretização é composta de arestas, na versão bidimensional da técnica, ou por triângulos, na versão tridimensional. A versão tridimensional da técnica será mais detalhada neste trabalho, mas sua correspondente em 2D é direta.

O primeiro passo da técnica sequencial é gerar uma decomposição recursiva do domínio, uma estrutura de dados chamada de *octree*. Esta servirá como guia, ou função densidade, para a inserção dos novos pontos. Posteriormente, baseada no tamanho das suas células da *octree*, a malha tetraédrica é gerada por avanço de fronteira. Ao final, são aplicadas estratégias de melhoria na malha gerada, para aumentar a sua qualidade.

### 3.2.1 Construção da *octree*

Inicialmente, um cubo limitante é gerado, englobando toda a região do domínio. Esse cubo é a célula-raiz da *octree*. Seleciona-se, então, uma face da fronteira e a célula da *octree* que contém o centroide dessa face (inicialmente, a célula-raiz da *octree*) é subdividida em 8 cubos iguais, recursivamente, até que o seu tamanho (a área de seu lado) seja menor que a área da face, multiplicada de um fator. Este fator controla o refinamento da *octree* e, portanto, o tamanho dos tetraedros gerados. Neste trabalho, o fator utilizado é de 0,4, valor encontrado empiricamente após vários experimentos. Esse procedimento é feito para todas as faces da fronteira. As Figuras 3.2a e 3.2b mostram um exemplo de fronteira e sua *octree* gerada.

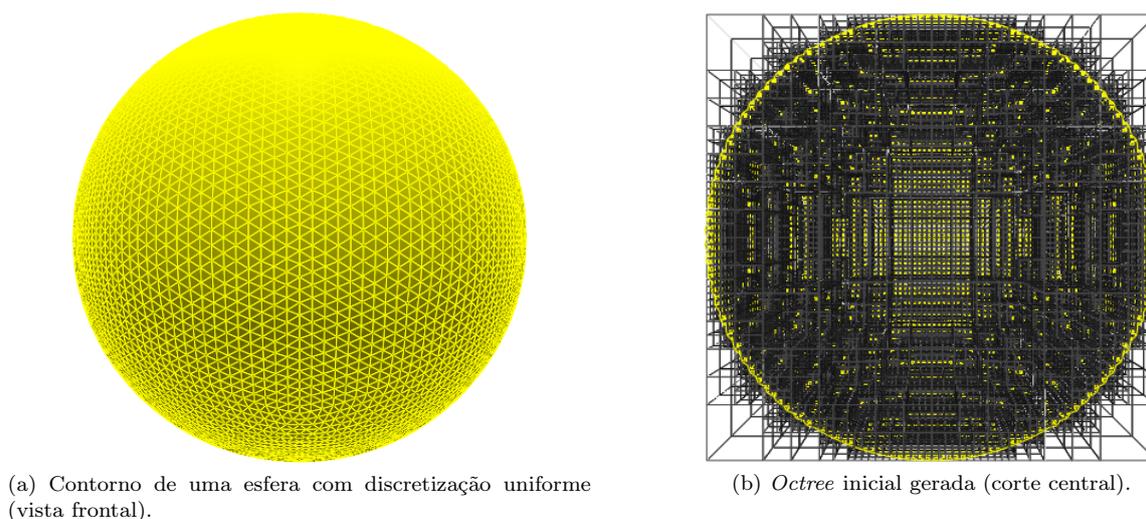
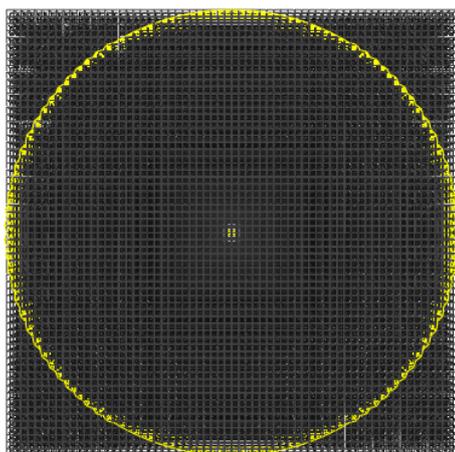
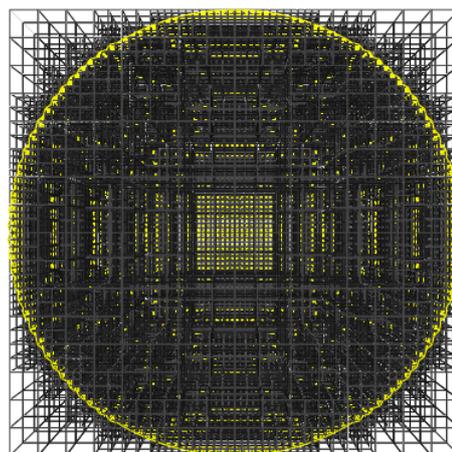


Figura 3.2 – Contorno e *octree* gerada.

Após a criação da *octree*, dois refinamentos diferentes são aplicados. Primeiramente, a maior célula interna ao domínio não deve ser maior que a maior célula da fronteira. Isso é feito para que não existam elementos no interior do domínio excessivamente grandes (Figura 3.2c), uniformizando o tamanho dos maiores elementos da malha. Dependendo das necessidades do usuário, este refinamento pode ser desativado, resultando na geração de uma malha com menos elementos. Este refinamento é conhecido como refinamento uniforme.

Posteriormente, a árvore é refinada para que duas células vizinhas (que compartilham um lado) não tenham tamanhos relativamente muito discrepantes. Em outras palavras, a diferença entre os níveis (na árvore) de duas células vizinhas não pode ser maior que 1 (Figura 3.2d). Este refinamento, conhecido na literatura como refinamento 2:1, garante que a transição entre regiões refinadas (com muitos elementos pequenos) e regiões grosseiras (com poucos elementos grandes) seja suave.

(c) *Octree* somente com o refinamento uniforme (corte central).(d) *Octree* somente com o refinamento 2:1 (corte central).Figura 3.2 – Contorno e *octree* gerada (continuação).

### 3.2.2 Geração da malha

O próximo passo da técnica é o de geração da malha. Este passo consiste em três etapas. A primeira, chamada de etapa baseada em geometria, tenta gerar elementos válidos de boa qualidade. A segunda, chamada de etapa baseada em topologia, apenas gera elementos válidos, mesmo que não sejam de qualidade. A terceira etapa, chamada de geração por retrocesso (*back-tracking*), tenta finalizar a geração da malha em regiões onde não foi possível completar a malha, através da remoção de alguns elementos e da geração de uma malha local por outro método. Apesar dessas três etapas, é possível que uma malha não possa ser gerada em alguns casos patológicos, visto que o problema de tetraedralização nem sempre tem solução.

Por conta das duas últimas etapas, é possível que alguns elementos gerados não tenham boa qualidade. Isto não é um problema sério na maioria das aplicações, especialmente considerando-se que tais elementos estão geralmente localizados longe da fronteira original, onde são criados os melhores elementos.

Para gerar os elementos, duas listas de faces são utilizadas, uma de faces ativas e outra de faces rejeitadas. Estas listas representam a fronteira corrente, que será avançada para gerar os novos elementos.

A etapa baseada em geometria inicia-se com a lista de faces ativas contendo as faces da fronteira, ordenadas crescentemente de acordo com sua área, e com a lista de faces rejeitadas vazia. Então, a face de menor área é retirada da lista de faces ativas (chamada de face base), e um vértice deve ser encontrado para gerar um novo tetraedro. Caso tal vértice não seja encontrado, essa face é movida para a lista de faces rejeitadas. Caso um vértice seja encontrado, um novo tetraedro é formado, e a fronteira é atualizada, inserindo a(s) nova(s) face(s) na lista de faces ativas ou removendo a(s) que já existia(m) previamente, de qualquer uma das listas.

Este procedimento é repetido até que a lista de faces ativas esteja vazia, momento em que a lista de faces rejeitadas passa a ser utilizada como lista de faces ativas, e o procedimento é repetido. Isso é feito pois, como a fronteira foi modificada, é possível que alguns tetraedros ainda sejam gerados. Quando uma face for rejeitada novamente, a etapa baseada em geometria termina, ainda com regiões não preenchidas. Caso as duas listas esvaziem-se, a malha foi completamente gerada.

Para encontrar um vértice para uma face, e assim formar um novo tetraedro, a etapa baseada em geometria segue os seguintes passos (Figura 3.3):

1. Encontre a célula da *octree* onde está localizado o centroide da face base, e pegue o comprimento de sua aresta  $t$  (Figura 3.3a);

2. Gere um ponto ideal  $P$  na reta definida pelo centroide e pelo vetor normal da face base, cuja distância à face base seja  $t$  (Figura 3.3b);
3. Crie uma região de busca esférica centrado em  $P$  e de raio  $t$  (Figura 3.3c);
4. Ache todos os vértices da fronteira corrente que estejam dentro da região de busca (Figura 3.3d) e selecione o melhor deles (ou seja, o vértice que tenha maior ângulo sólido com os vértices da face base, Figura 3.3e) que forme um tetraedro válido (isto é, que não cruze nenhuma face da fronteira corrente);
5. Caso nenhum vértice seja encontrado, teste se o tetraedro formado pela face base e por  $P$  é válido. Se for válido, forme um novo elemento com esse ponto e com essa face base e atualize a fronteira. Se não for, rejeite esta face, selecione uma nova face base, e volte ao passo 1.

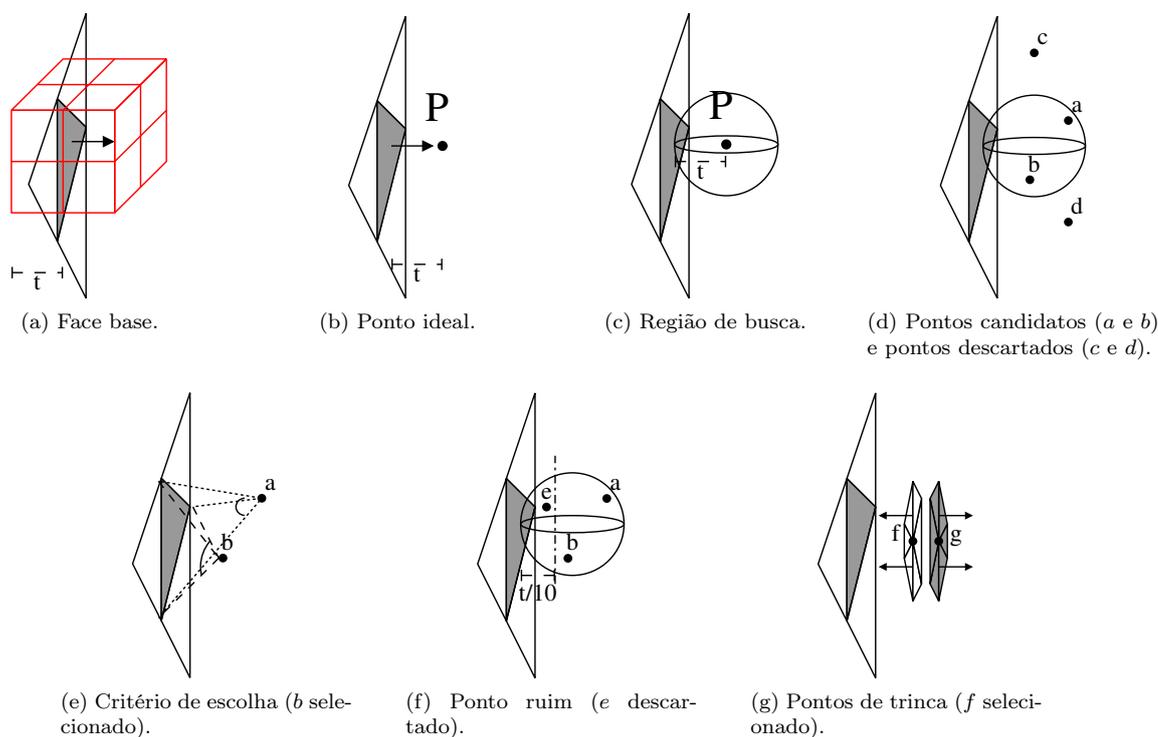


Figura 3.3 – Avanço de uma face da fronteira.

Durante a busca por vértices existentes, duas observações devem ser feitas:

- Na implementação atual, o valor  $t$  do tamanho da célula é multiplicado por 1,5, aumentando assim o raio da esfera de busca;
- Os vértices encontrados que estiverem atrás da face base, ou seja, que não estejam no semi-espaço definido pelo plano da face base e pelo seu vetor normal, são descartados;
- Os vértices encontrados que estiverem à distância de até  $t/10$  da face base são descartados, pois estes formariam tetraedros de má qualidade (Figura 3.3f);
- Quando dois vértices fizerem parte de uma trinca, caso em que terão mesmas coordenadas, o vértice dentre eles a ser selecionado deve ser aquele cujas faces adjacentes, pertencentes à fronteira original, tenham normais apontando para a face base (Figura 3.3g).

A etapa baseada em topologia ocorre somente se uma face for rejeitada duas vezes, a primeira quando estava na lista de faces ativas e a segunda quando estava na lista de faces rejeitadas, de forma que uma malha ainda não foi totalmente gerada para aquela fronteira. Assim como na etapa baseada em geometria, esta etapa também utiliza duas listas, a lista de faces ativas, inicializada como a lista de faces rejeitadas da etapa passada, e a lista de faces rejeitadas, inicialmente vazia.

A primeira diferença entre as duas etapas é que esta não mais insere pontos, ou seja, utiliza somente os vértices já existentes na fronteira. A segunda diferença é que esta etapa não restringe a qualidade dos tetraedros gerados. Assim, todos os vértices que estiverem no semi-plano definido pela normal de uma face base são candidatos a formar um novo tetraedro, e aquele que formar o maior ângulo sólido com a face base é selecionado, desde que o novo tetraedro seja válido.

Na implementação atual, tanto a etapa baseada em geometria quanto a baseada em topologia são precedidas de um teste de ângulo. Se o ângulo da face base com uma de suas faces adjacentes for menor que um limiar, um tetraedro é inserido com os quatro vértices dessas faces, desde que esse tetraedro seja válido. O limiar utilizado é de  $75^\circ$ , valor encontrado através de experimentos. Além disso, para acelerar as buscas feitas no passo 4 do avanço de fronteira, são utilizadas estruturas de dados auxiliares do tipo R-tree (GUTTMAN, 1984).

Na versão bidimensional da técnica, as duas primeiras etapas são suficientes para gerar uma malha triangular. Na versão 3D, entretanto, é possível que uma malha não tenha sido gerada na etapa anterior. A fronteira corrente consiste em cavidades, onde não é possível gerar nenhum tetraedro, nem mesmo com a inserção de novos pontos. Nesses casos, as cavidades são identificadas e, em cada uma delas, é aplicada uma outra etapa de geração de elementos, numa etapa chamada de geração de malhas por retrocesso (*back-tracking*), (CAVALCANTE-NETO et al., 2005). A Figura 3.4 mostra uma esquematização do procedimento, simplificada para duas dimensões, para facilitar a visualização.

Para uma determinada cavidade (Figura 3.4a), um ponto é tentativamente inserido no seu centroide (Figura 3.4b), e testes de visibilidade são feitos gerando-se arestas entre o ponto criado e os vértices da fronteira da cavidade (Figura 3.4c). Caso uma face da fronteira intercepte alguma(s) dessas arestas, esta face é removida, juntamente com o elemento conectado a ele, aumentando-se a cavidade (Figura 3.4d). Assim, retorna-se ao início, tentando-se criar um novo ponto no novo centroide (Figura 3.4e). Quando o centroide for visível a partir de todos os vértices da cavidade, os elementos são criados, preenchendo a cavidade (Figura 3.4f).

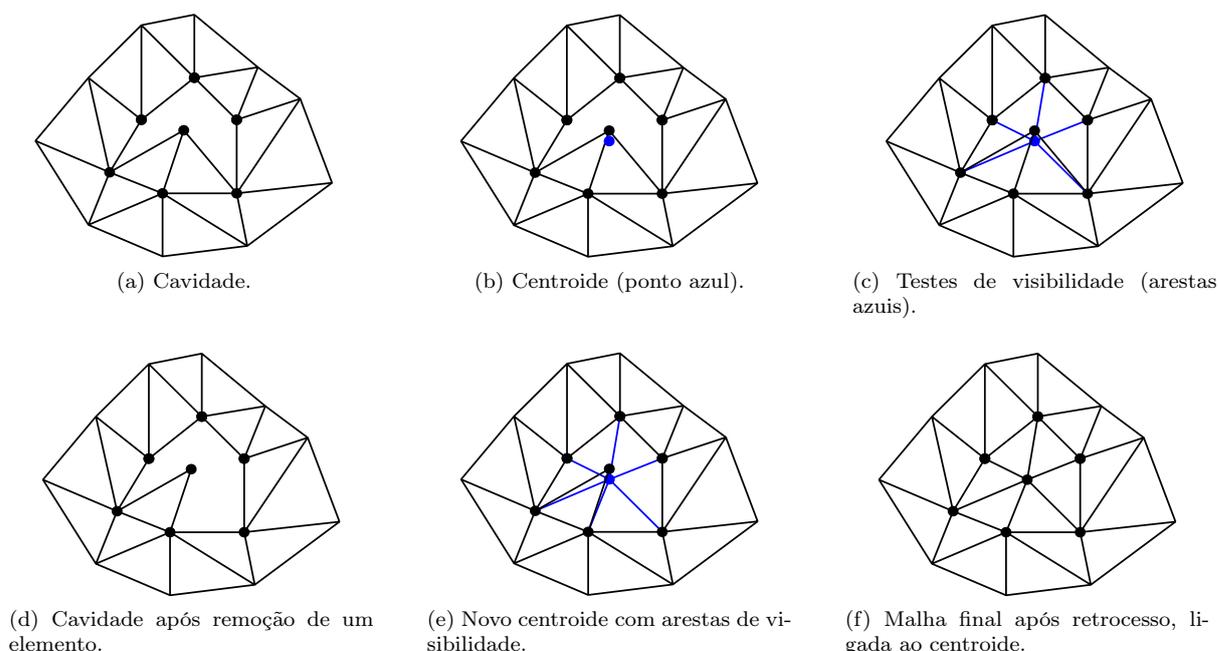


Figura 3.4 – Geração de malhas por retrocesso.

O elemento removido é aquele que interceptar o maior número de arestas, sendo assim aquele que mais atrapalha a visibilidade. Além disso, tetraedros cujas quatro faces pertençam à fronteira da cavidade

também são removidos. Se um tetraedro for adjacente à fronteira original, então ele não é removido pois, como foi criado primeiramente, é o de melhor qualidade para aquela face da fronteira inicial.

Mesmo com esta etapa de avanço de fronteira por retrocesso, é possível, embora raro, que uma malha não seja gerada, caso em que todos os elementos são removidos, restando somente aqueles adjacentes à fronteira original. Neste caso, a técnica falha. Essa possível falha é justificável porque o problema de tetraedralização nem sempre é resolvível, uma vez que é possível criar um modelo cuja malha tetraédrica não exista. Entretanto, isso raramente acontece na prática.

### 3.2.3 Melhoria da malha

Na implementação atual, a malha gerada passa por três etapas de melhoramento, duas que modificam a topologia da malha, e uma que modifica apenas a geometria. As duas primeiras etapas são de trocas de faces (*face swapping*) e de trocas de arestas (*edge swapping*), como descritas nas Figuras 3.5 e 3.6. A troca de arestas acontece somente se esta tiver de 3 a 7 tetraedros adjacentes. Nas duas etapas, as trocas acontecem somente se os elementos gerados após a troca melhorarem a qualidade da malha. Na troca de arestas, as arestas da malha são percorridas 4 vezes, enquanto que, na troca de faces, por ser mais computacionalmente pesada, as faces são percorridas somente uma vez.

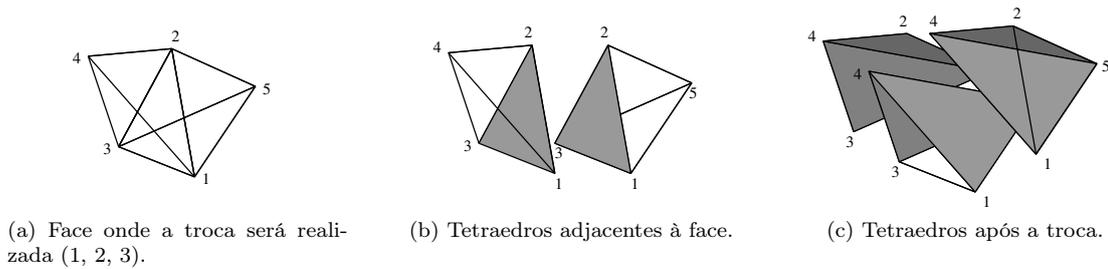


Figura 3.5 – Troca de faces.

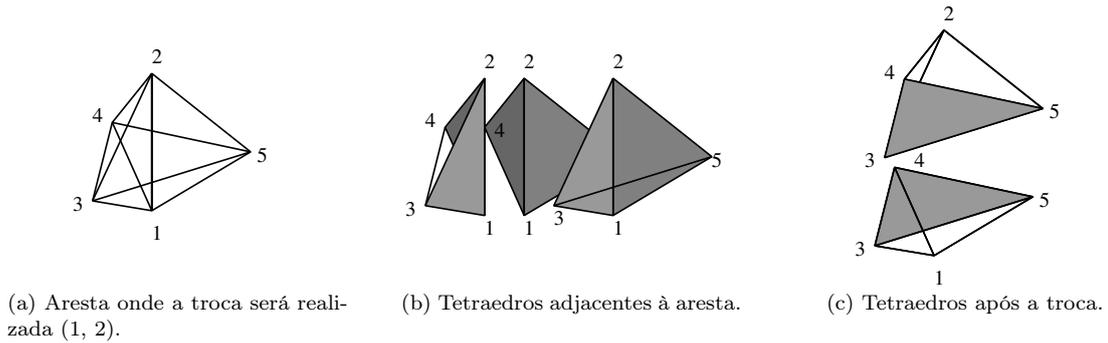


Figura 3.6 – Troca de arestas, com três tetraedros adjacentes (caso inverso à troca de faces).

A etapa que modifica apenas a geometria é a de suavização de vértices, conhecida como suavização de Laplace (referente a Pierre-Simon Laplace, matemático francês), que consiste em mover levemente um vértice interno da malha para que ele se aproxime mais do centroide do polígono definido pelos seus pontos adjacentes, e pode ser definida através da Equação 3.1:

$$X_V^{n+1} = X_V^n + \phi \frac{\sum_{i=1}^m (X_i^n - X_V^n)}{m}. \tag{3.1}$$

Nessa equação,  $X_V^n$  é a posição do vértice  $V$  na iteração de suavização  $n$ ,  $m$  é o número de vértices adjacentes a  $V$  por algum tetraedro,  $i$  corresponde o  $i$ -ésimo vértice adjacente a  $V$ ,  $X_i^n$  é a posição do

$i$ -ésimo vértice adjacente a  $V$  na iteração de suavização  $n$ , e  $\phi$  é o parâmetro de relaxamento, normalmente ajustado para um valor entre 0 e 1 (neste trabalho, o valor de 1 foi utilizado para  $\phi$ ).

A suavização de Laplace é aplicada iterativamente em todos os vértices internos da malha, movendo-os caso os elementos suavizados sejam melhores que o pior elemento de antes da suavização, melhorando a qualidade da malha. Os vértices são percorridos até que menos de 30% dos vértices sejam movidos, momento em que o laço é quebrado e a suavização termina.

Tanto para a suavização de Laplace quanto para as trocas de faces e de arestas, são utilizadas estruturas de dados auxiliares que guardam quais são os elementos adjacentes dos nós, das arestas e das faces. Isso é feito para que não haja buscas exaustivas na estrutura da malha, que poderia levar muito tempo.

### 3.2.4 Resultados

Esta técnica gera malhas de boa qualidade, para fronteiras contínuas ou descontínuas, com ou sem trincas. As malhas geradas têm uma boa transição de elementos de tamanho diferentes. O tempo de execução, calculado através de execuções exaustivas, é de  $O(CN^p \log N)$  seguindo o que foi apresentado na Seção 2.2.1.1. Vários exemplos podem ser encontrados nas referências anteriormente mencionadas.

## 3.3 Técnica paralela

A presente técnica paralela utiliza uma estrutura de árvore binária para decompor o domínio, chamada de BSP (*Binary Spatial Partitioning*). Em uma BSP, a célula-raiz engloba todo o domínio, e uma célula qualquer pode ser subdividida em duas através do posicionamento de um plano de partição no seu interior. A localização desse plano é dado por um ponto e um vetor normal ao plano de partição. Apesar de a estrutura, teoricamente, aceitar um posicionamento qualquer, neste trabalho, o vetor será sempre paralelo a um dos eixos canônicos, X, Y ou Z. Assim, uma célula-folha da BSP tem a geometria de um paralelepípedo retangular.

Por dar liberdade no posicionamento, o ponto onde localiza-se o plano de partição pode ser qualquer. Então, tenta-se posicionar o plano de partição de maneira que divida a carga de trabalho ao meio, ou seja, que melhor equilibre uma estimativa de distribuição de carga pelo modelo. Como cada célula da BSP é subdividida em duas, o número final de células-folha pode ser arbitrário. Assim, a condição de parada das subdivisões é que o número de folhas seja igual ao número de processos/*threads*. A Figura 3.7 mostra um exemplo bidimensional de uma BSP construída com 8 folhas, que pode ser utilizada para 8 processos, 4 processos com 2 *threads*, 2 processos com 4 *threads*, ou 8 *threads*.

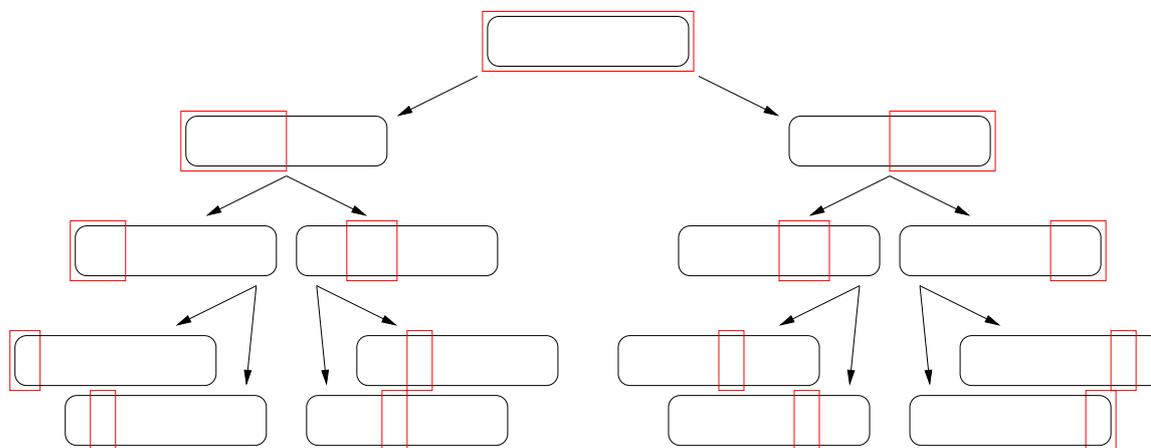


Figura 3.7 – Exemplo de árvore BSP criada para um domínio (em preto). Cada célula (em vermelho), interna ou folha, tem a geometria de um retângulo.

A estimativa de distribuição de carga neste trabalho é inferida a partir da *octree* construída pela técnica sequencial (Seção 3.2.1), pois é esta estrutura que guia a inserção dos novos vértices da malha. Assim, o plano de partição de uma célula da BSP deve coincidir com os limites das células da *octree*, e deve equilibrar o número de células da *octree* que estão dos dois lados do plano.

Cada processo fica responsável por somente um subdomínio, ou seja, uma célula-folha da BSP, e gera malha enquanto puder, empregando o avanço de fronteira sequencial (Seção 3.2.2), que foi modificado para que a fronteira não ultrapasse os limites definidos pelo subdomínio. Além disso, é necessário que exista um espaço livre entre a fronteira e os limites do subdomínio (Figura 3.8a), para que haja espaço suficiente para a formação de bons elementos na geração da malha de interface. O processo aplica, ainda, as mesmas melhorias da técnica sequencial (Seção 3.2.3), que também foram modificadas de forma a não alterar a posição da fronteira depois de avançada.

As interfaces entre os subdomínios são definidas pelas células internas da BSP. Assim, as malhas de interface são geradas subindo-se nível a nível na BSP. Durante esse procedimento, é necessário haver sincronização entre os processos/*threads* responsáveis pelas duas células-filhas de uma célula interna, para que um deles descubra a fronteira do outro e gere a malha de interface, aplicando o mesmo gerador de malha modificado (Figura 3.8b). Uma vez que a malha da célula-raiz da BSP tenha sido gerada, o procedimento paralelo termina com a malha completamente gerada. Caso a malha não consiga ser gerada na raiz, o procedimento paralelo falha, sem gerar a malha por completo. Entretanto, assim como na técnica sequencial, isso raramente acontece, e o procedimento paralelo não falhou para nenhum dos modelos apresentados.

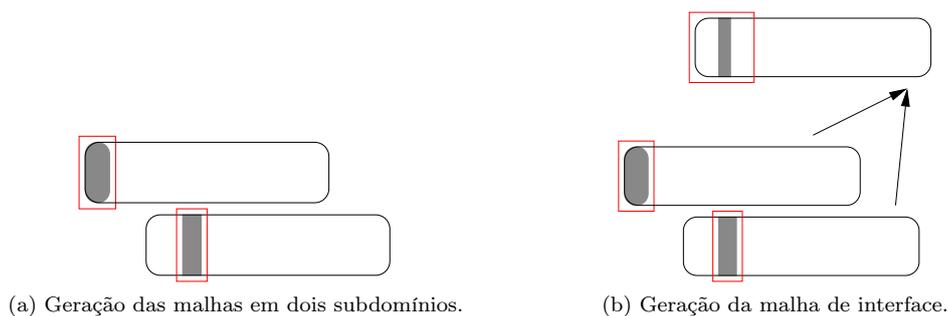


Figura 3.8 – Geração das malhas (em cinza) respeitando a geometria das células da BSP.

### 3.3.1 Estimativa da carga

Em Computação de Alto Desempenho, a carga é um valor que determina a quantidade de computação a ser realizada em um processo. Neste trabalho, como cada processo é responsável por um subdomínio, essa quantidade de computação deve refletir o tempo de execução de um subdomínio. Entretanto, prever o tempo é um problema muito difícil, pois envolve fatores externos à implementação como, por exemplo, tempos de acesso às memórias *cache* e RAM, tempo de comunicação, execução de outros processos, entre outros fatores. Portanto, em vez de calcular o tempo, calcula-se a complexidade do algoritmo, dependendo de outros termos.

A complexidade dos algoritmos de geração de malha por avanço de fronteira é dada em função quantidade de elementos gerados. Assim, calcular a carga em função da entrada, ou seja, do tamanho da malha de superfície, é muito difícil e, portanto, é calculada uma estimativa dessa carga. Em tais técnicas, a estimativa da carga deve ser proporcional a quantidade de elementos gerados. Dessa forma, ela deve levar em conta (FREITAS et al., 2013):

- Para uma mesma fronteira, a carga deve variar se o tamanho desejado dos elementos variar, ou seja, se a malha final for formada por elementos menores, então a quantidade de elementos deve ser

maior e, portanto, a carga total deve ser maior (Figura 3.9);

- Em regiões com discretização uniforme, uma sub-região de maior área deve gerar mais elementos que uma de menor área (Figura 3.10);
- Em domínios com discretização não-uniforme, uma região de transição de elementos não deve ser contada como se fosse regular, pois existirão elementos menores em uma parte e elementos maiores em outra (Figura 3.11a). Isso vale, por exemplo, para uma fronteira com regiões mais discretizadas, que geram mais elementos que regiões menos discretizadas e, portanto, devem contar mais para o cálculo da carga (Figura 3.11b).

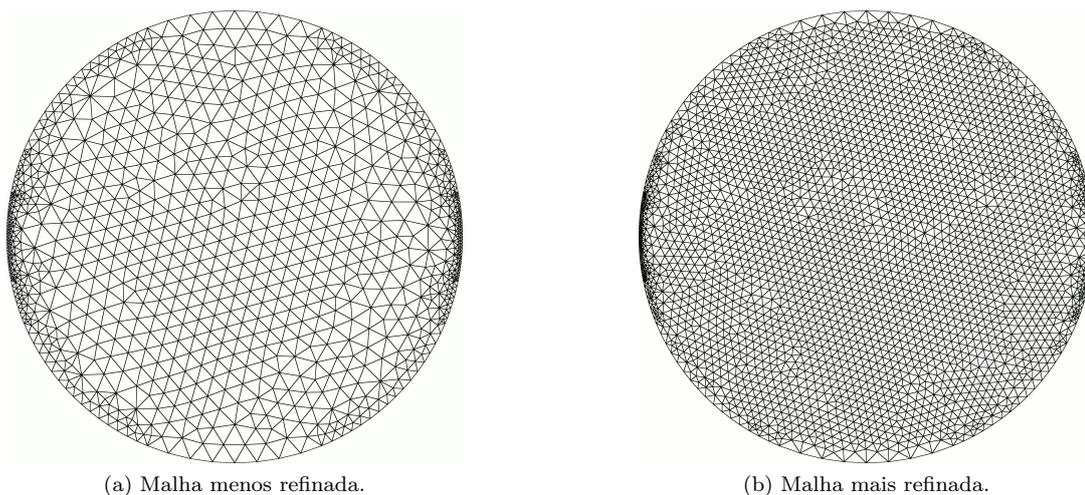


Figura 3.9 – Comparação entre malhas refinada e não refinada. Perceba que a mesma fronteira foi utilizada para as duas malhas.

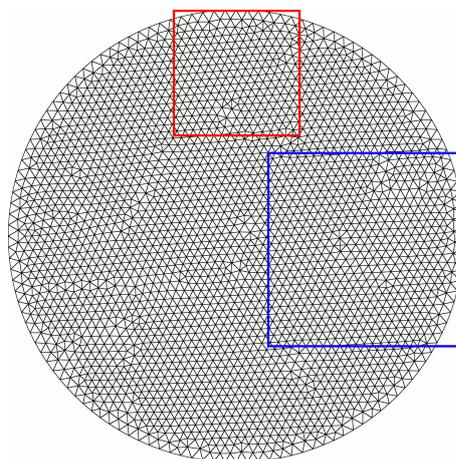
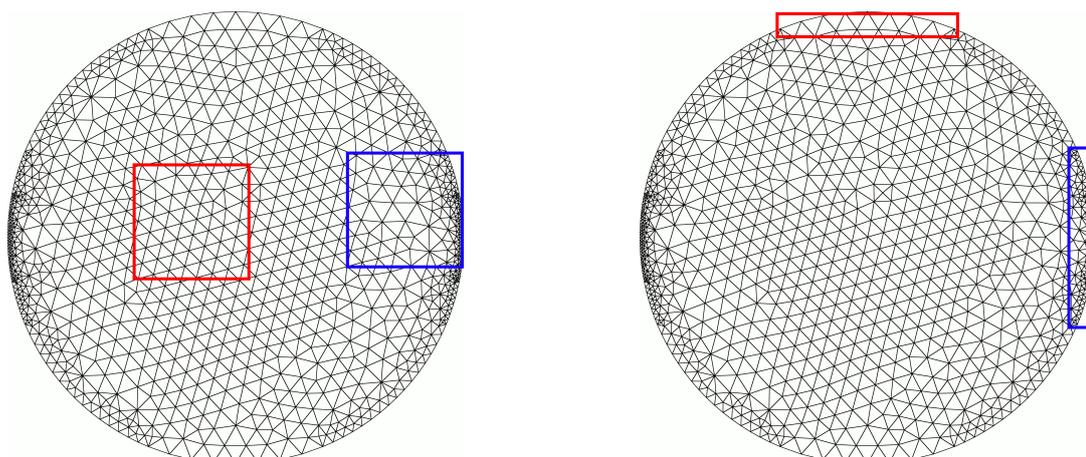


Figura 3.10 – Fronteira com discretização uniforme (azul - área maior, vermelha - área menor).

As Figuras 3.9, 3.10 e 3.11 apresentam exemplos bidimensionais, por proporcionar uma visualização mais compreensiva dos fatores envolvidos estimativa da carga. Entretanto, esses fatores são gerais, valendo para duas ou três dimensões.

A *octree* construída pela técnica sequencial (Seção 3.2.1), que é a sua função densidade, reúne todas essas características, pois é ela que define o tamanho dos elementos gerados, tanto em regiões próximas à fronteira quanto em regiões internas, considerando ainda variações suaves de tamanhos. Portanto, indiretamente, suas células predizem a quantidade de elementos que serão gerados. Por esse motivo, essa *octree* é utilizada também como estrutura de estimativa de carga.



(a) Regiões internas de transição (azul - com transição, vermelho - sem transição).

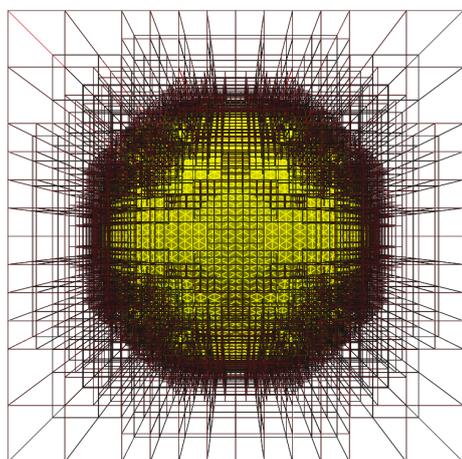
(b) Regiões de transição na fronteira (azul - mais discretizada, vermelha - menos discretizada).

Figura 3.11 – Regiões de transição.

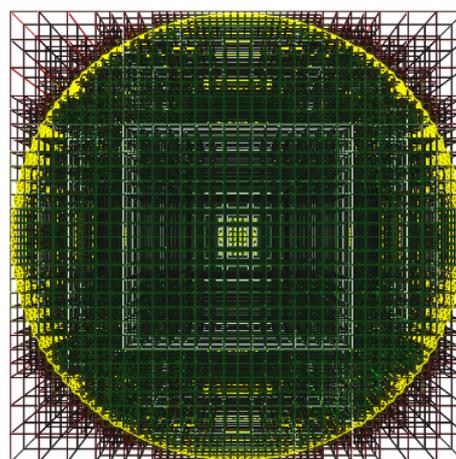
Apesar de a técnica sequencial descrita na Seção 3.2 depender da *octree* para gerar os elementos, esta estrutura pode ser utilizada com outros geradores de malha, desde que sejam implementados os refinamentos apropriados para que a *octree* reflita o tamanho desejado dos elementos da malha final. Por exemplo, em uma simulação de elementos finitos, a geração da malha em um passo de tempo pode depender dos erros da malha do passo de tempo anterior, requerendo um refinamento maior em regiões longe da borda do modelo, ou seja, no interior da malha. Assim, basta-se implementar um refinamento na *octree* que dependa dos erros da malha anterior, para que essa estrutura também apresente elementos menores nessas regiões internas (MIRANDA et al., 2013).

### 3.3.1.1 Classificação das células

Como não são gerados elementos fora do modelo, as células da *octree* de estimativa de carga que estiverem externas não podem ser contabilizadas no cálculo da estimativa de carga. Para tanto, as células devem ser classificadas como internas, sobre a borda, ou externas à fronteira original. A Figura 3.12 mostra a *octree* de estimativa de carga classificada, em que as células verdes estão dentro do domínio, as células vermelhas estão fora do domínio, e as células amarelas estão sobre a fronteira.



(a) *Octree* de estimativa de carga (vista frontal).



(b) *Octree* de estimativa de carga (corte central).

Figura 3.12 – *Octree* de estimativa de carga classificada.

A classificação é feita da seguinte maneira (Figura 3.13):

1. Classifique as células que estão sobre a borda, ou seja, aquelas que cruzam pelo menos uma das faces da fronteira. Nesse passo, as faces que cruzam uma célula são descobertas descendo-se recursivamente a estrutura de árvore da *octree*, utilizando um algoritmo de *clipping* (HEARN; BAKER, 1996; ANGEL, 2008) da face contra o cubo definido pela célula;
2. Encontre uma célula-folha  $C$  classificada como sobre a borda (Figura 3.13a);
3. Encontre uma célula  $N$  vizinha de  $C$  ainda não classificada (Figura 3.13b) e tente classificá-la como interna ou externa:
  - a) Encontre todas as faces que cruzam  $C$  (Figura 3.13c);
  - b) Verifique se as todas normais dessas faces apontam para a direção pela qual  $N$  é vizinha de  $C$  (Figura 3.13d);
  - c) Caso todas as normais apontem para essa direção,  $N$  é classificada como interna. Caso todas as normais apontem para a direção contrária,  $N$  é classificada como externa (Figura 3.13e). Caso não haja unanimidade entre as normais,  $N$  continua sem classificação, e um novo vizinho é escolhido, ou seja, volta-se ao passo 3;
4. Aplique, em  $N$ , um algoritmo parecido com o preenchimento de regiões por *flood-fill* (HEARN; BAKER, 1996; ANGEL, 2008) para classificar as suas células vizinhas:
  - a) Encontre uma célula vizinha de  $N$  não classificada;
  - b) Classifique-a com a mesma classificação de  $N$ ;
  - c) Aplique este algoritmo recursivamente a essa célula, voltando ao passo 4a;
5. Volte ao passo 3 escolhendo um novo vizinho  $N$  de  $C$  ainda não classificado. Este procedimento para quando todos os vizinhos de  $C$  tiverem sido testado, mesmo que algum deles não tenha sido classificado;
6. Volte ao passo 2, encontrando uma nova célula-folha classificada como sobre a borda que tenha algum ainda vizinho não classificado;
7. Caso haja ainda alguma célula não classificada, classifique-a utilizando o algoritmo do ângulo (CARVALHO; FIGUEIREDO, 1991):
  - a) Encontre o centroide dessa célula;
  - b) Some os ângulos sólidos que o centroide faz com todas as faces da borda;
  - c) Se o ângulo sólido for (aproximadamente)  $4\pi$ , a célula está interna ao domínio; caso contrário, está externa ao domínio;
8. Aplique o algoritmo de *flood-fill* nessa célula, para classificar os seus vizinhos;
9. Volte ao passo 7, procurando por células ainda não classificadas. Caso todas as células tenham sido classificadas, a classificação da *octree* está terminada.

O algoritmo de *flood-fill*, nos passos 4 e 8, sempre termina, pois o passo 1 foi justamente o de classificar todas as células que estão sobre a borda.

### 3.3.1.2 Estimativa de carga em cada subdomínio

As células da *octree* contabilizadas para a estimativa de carga, ou seja, as células cheias, são as que foram classificadas como internas ou como sobre a borda. Assim, o número total de células cheias é a estimativa de carga do modelo como um todo.

Durante a construção da BSP, para decompor o domínio, uma célula cheia não pode ser cortadas por um plano de partição e, portanto, os limites de um subdomínio coincidem com limites de células cheias. Logo, elas são separadas pelo plano de partição, e aquelas que estiverem dentro de um subdomínio formam a estimativa de carga para este subdomínio (Figura 3.14).

Como dito no início da Seção 3.3, a geração da malha em um subdomínio termina deixando um espaço livre entre a fronteira e os planos de partição que delimitam o subdomínio. Para melhorar a estimativa de

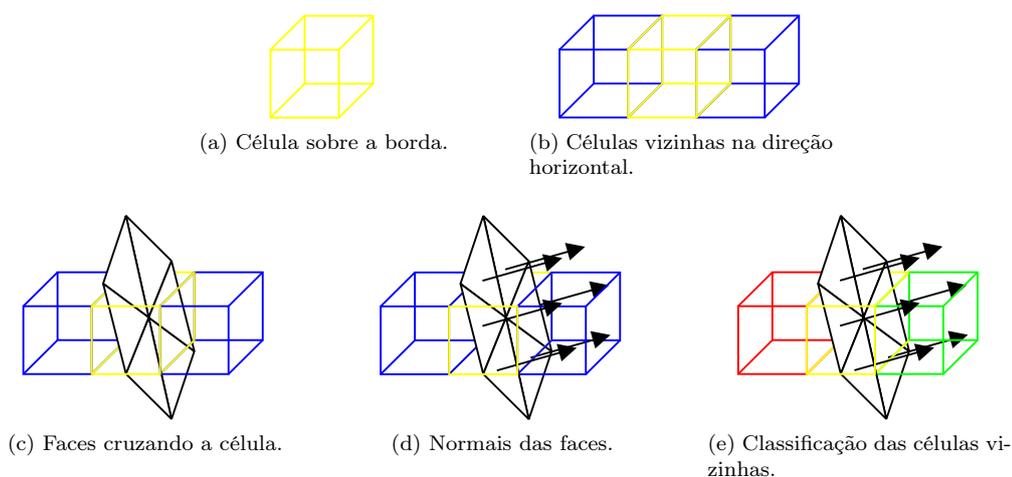


Figura 3.13 – Classificação das células da *octree* de estimativa de carga (amarelo - sobre a borda, verde - interna, vermelho - externa, azul - classificação desconhecida).

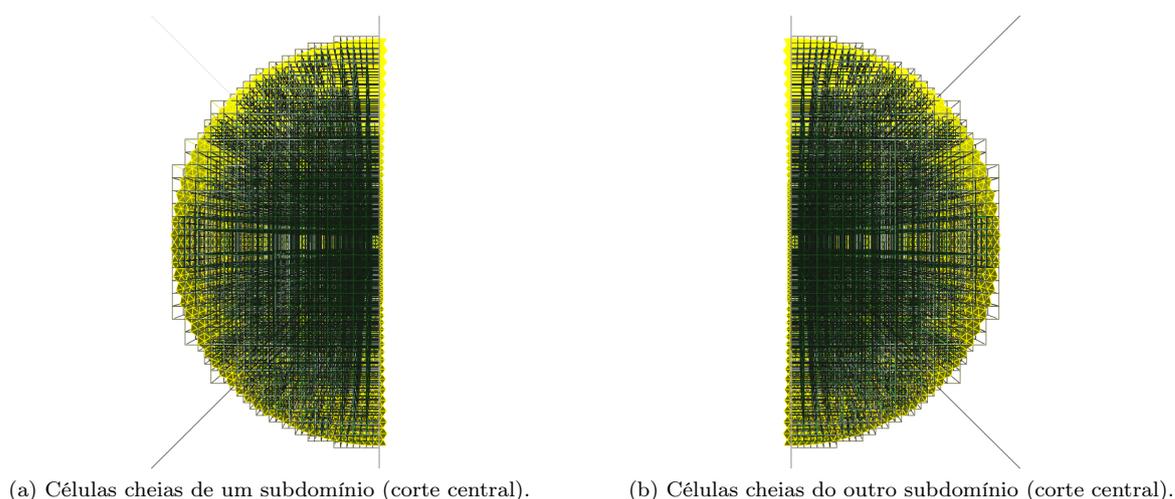


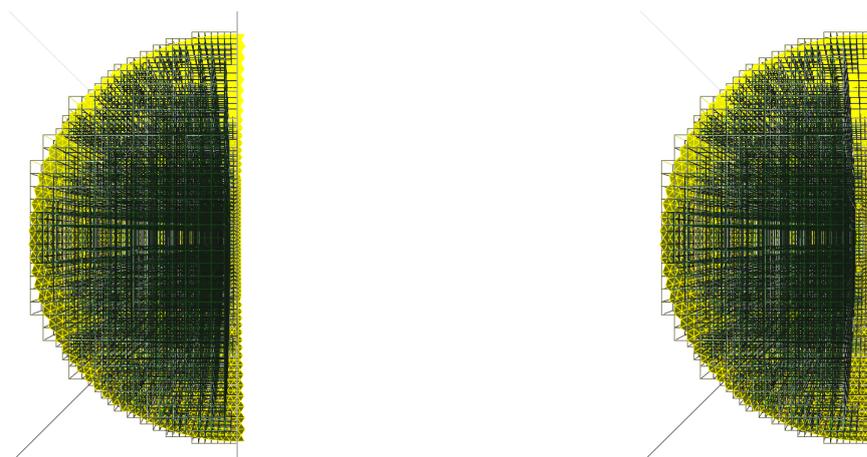
Figura 3.14 – Células cheias em cada subdomínio (amarelo - sobre a borda, verde - interna).

carga, esta região livre deve ser descontada da estimativa de carga. Esse desconto é feito contabilizando-se a quantidade de células internas adjacentes ao plano de partição, e diminuindo-se esse valor da estimativa previamente calculada. A Figura 3.15a mostra um exemplo onde as células adjacentes ao plano de partição foram removidas.

Caso necessário, é possível diminuir ainda mais a estimativa de carga de cada subdomínio, aumentando o valor descontado por conta da região livre. Para isso, remove-se a estimativa de carga relativa às células internas vizinhas às células encontradas no parágrafo anterior (Figura 3.15b). Esse procedimento pode ser repetido, encontrando-se e descontando-se novas camadas de células vizinhas. Entretanto, observou-se que a remoção de somente uma camada, ou seja, somente as células adjacentes ao plano de partição, já era suficiente para uma boa estimativa.

### 3.3.1.3 Estimativa de carga entre subdomínios

Como dito no início da Seção 3.3, as células internas da BSP definem as interfaces entre os subdomínios e, em algum momento, as malhas nessas interfaces serão geradas. A fim de comparar a quantidade de trabalho de geração de malha nas interfaces, uma estimativa de carga também é realizada para cada uma



(a) Removendo-se 1 camada adjacente ao plano de partição (corte central).

(b) Removendo-se 2 camadas adjacentes ao plano de partição (corte central).

Figura 3.15 – Células cheias em cada subdomínio após a remoção de camadas internas (amarelo - sobre a borda, verde - interna).

delas. Esta estimativa de carga corresponde às células internas adjacentes ao plano de partição, dos dois lados do plano. Em outras palavras, as células internas que foram removidas dos subdomínios passam a fazer parte da interface que separa os subdomínios.

### 3.3.2 Decomposição do domínio

A ideia da construção da BSP é fazer uma busca binária recursiva, em cima da *octree*, pelo melhor posicionamento do plano de corte possível para subdividir uma célula da BSP. Isso é feito em duas etapas. Na primeira delas, testa-se cada um dos eixos, X, Y e Z, buscando-se equilibrar as estimativas de carga dos dois subdomínios gerados no eixo testado. Na segunda etapa, a melhor decomposição dentre os três eixos é escolhida, X, Y ou Z, levando-se em conta não somente o equilíbrio das cargas dos subdomínios, mas também as cargas das interfaces nos eixos.

O primeiro passo da construção da BSP é a criação da sua célula-raiz, que é geometricamente igual à raiz da *octree*. Além disso, cada célula da BSP corresponde a um conjunto de células da *octree*, e, inicialmente, a raiz da BSP corresponde ao conjunto unitário formado somente pela raiz da *octree*.

#### 3.3.2.1 Posicionamento do plano de partição

O posicionamento do plano de partição deve ser feito de forma a respeitar os limites das células cheias da *octree* (Figuras 3.16a, 3.16b e 3.16c). Isso deve acontecer para que não haja confusão na estimativa da carga de um subdomínio, isto é, se uma célula da *octree* cheia fosse cortada pelo plano de partição, ela poderia pertencer a qualquer um dos dois subdomínios. Um posicionamento cuidadoso evita esse tipo de ambiguidade. Entretanto, células da *octree* classificadas como externas ao domínio podem ser cortadas pelo plano de partição (Figura 3.16d), pois elas não são contabilizadas para o cálculo da estimativa de carga.

A ideia do algoritmo a seguir é, dado um conjunto de células da *octree*, encontrar uma posição que seja a mediana dentre os limites das células. Como o propósito da decomposição é descer recursivamente na *octree*, somente as células que estão no maior nível são consideradas, desde que tenham sido classificadas como internas ou como sobre a borda.

Assim, dado um conjunto de células da *octree*  $S_0$ , um eixo de direção  $d$ , e os limites geométricos mínimo  $min$  e máximo  $max$  de onde essas células estão, o posicionamento do plano de partição é feito

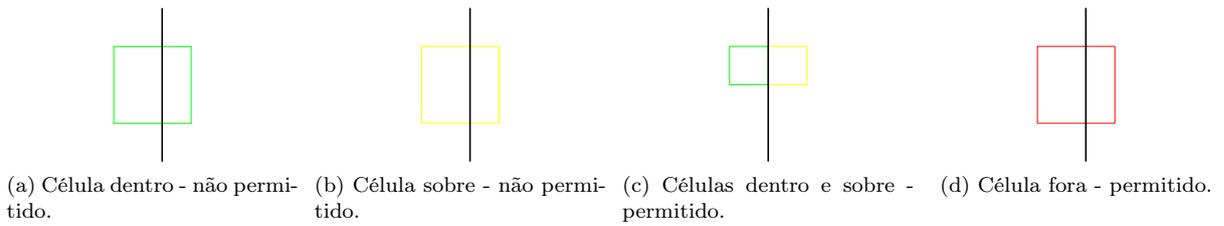


Figura 3.16 – Posicionamento do plano de partição (em preto) com relação às células de uma *quadtree*.

da seguinte maneira:

1. Seja  $V$  um conjunto, vazio, que guardará os valores limites das células da *octree*;
2. Seja  $S \subseteq S_0$  o conjunto das células que têm menor nível (ou seja, as de maior tamanho) dentre as células de  $S_0$  e que não foram classificadas como externas ao domínio;
3. Remova uma célula  $C$  de  $S$ ;
4. Sejam  $C_{min}$  e  $C_{max}$  os limites mínimo e máximo de  $C$  na direção  $d$ , respectivamente;
5. Se  $C_{min} = min$  e  $C_{max} = max$ , retorne  $\frac{min+max}{2}$ ;
6. Se  $C_{min} \neq min$ , adicione  $C_{min}$  a  $V$ ;
7. Se  $C_{max} \neq max$ , adicione  $C_{max}$  a  $V$ ;
8. Caso  $S$  não esteja vazio, volte ao passo 3;
9. Caso  $V$  esteja vazio, retorne  $\frac{min+max}{2}$ ;
10. Retorne o valor mediano de  $V$ .

A Figura 3.17 mostra um exemplo bidimensional, utilizando uma *quadtree*, do funcionamento do algoritmo acima descrito. Dado um conjunto de células, assumidamente todas cheias (Figura 3.17a), as células de maior nível são encontradas (Figura 3.17b), e os posicionamentos nos dois eixos, X e Y, são realizados.

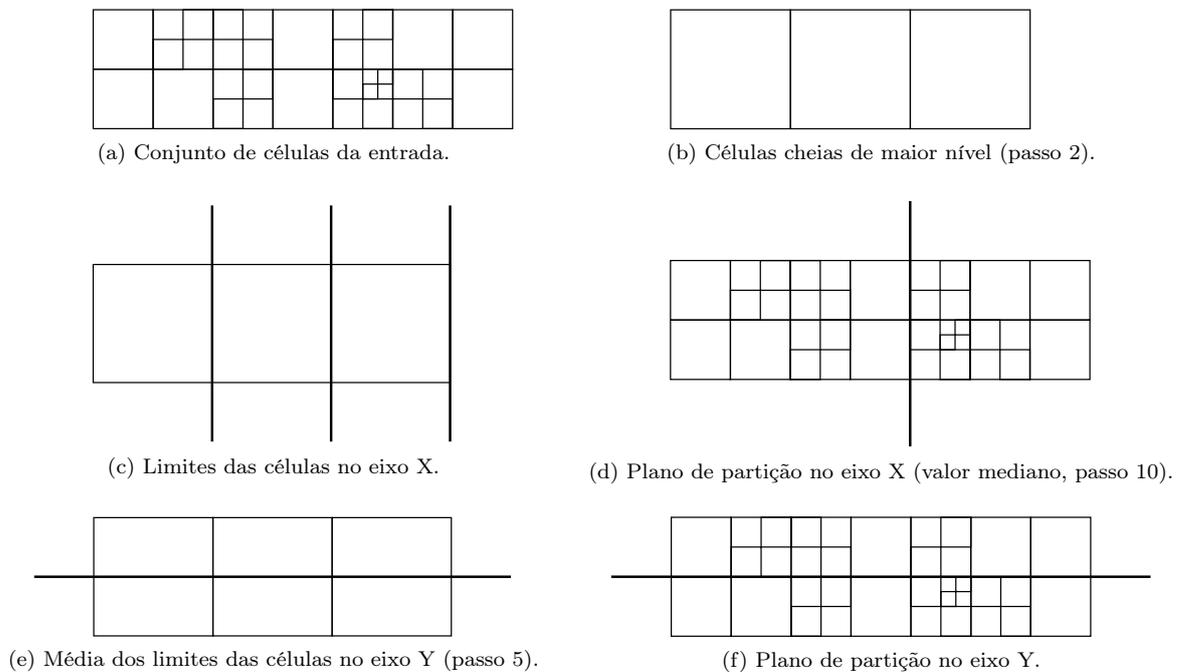


Figura 3.17 – Posicionamento do plano de partição para um conjunto de células de uma *quadtree*.

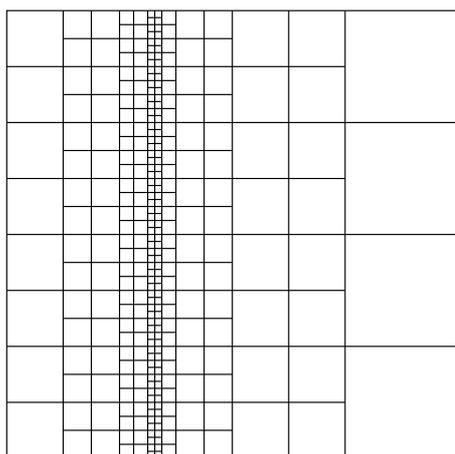
No eixo X, acontece o caso geral, ou seja, os limites internos das células de maior nível não coincidem com os limites globais. Assim, um conjunto com esses valores é encontrado (Figura 3.17c) e o seu valor

mediano é retornado pelo algoritmo (Figura 3.17d). No eixo Y, acontece o caso específico de os limites das células de maior nível coincidirem com os limites globais (Figura 3.17e). Logo, o valor médio é retornado pelo algoritmo (Figura 3.17f). Esse caso acontece, por exemplo, quando o algoritmo é aplicado à célula raiz da BSP, que corresponde à célula raiz da *octree*.

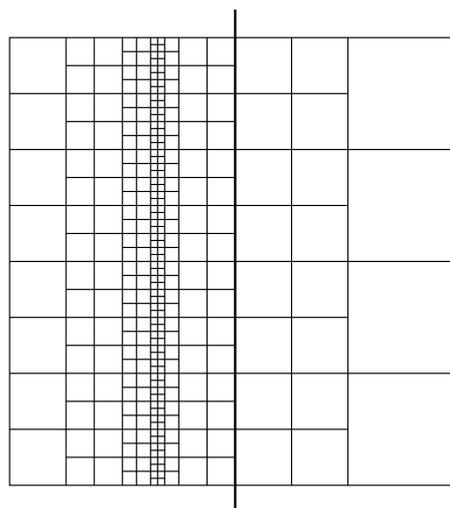
### 3.3.2.2 Decomposição em cada eixo

A Figura 3.18 exemplifica o algoritmo de decomposição aqui descrito, utilizando uma estrutura bidimensional para estimativa de carga, ou seja, uma *quadtree*. Assume-se, para efeito de ilustração, que todas as células da *quadtree* são classificadas como internas, ou seja, todas elas são contabilizadas para o cálculo da estimativa de carga.

Para subdividir, em algum eixo, uma célula da BSP em duas novas células, encontra-se o conjunto de células da *octree* correspondente a ela (Figura 3.18a) e aplica-se o algoritmo descrito anteriormente, para o posicionamento apropriado do plano de partição (Figura 3.18b). Inicialmente, a única célula da BSP existente é a sua raiz, que corresponde à raiz da *octree*, e o plano de partição é posicionado no seu centro geométrico. Esse plano separa os filhos da raiz da *octree*, deixando quatro de um lado e os outros quatro do outro lado.



(a) *Quadtree* de estimativa carga. Carga total: 316. Carga ideal de cada subdomínio: 158.

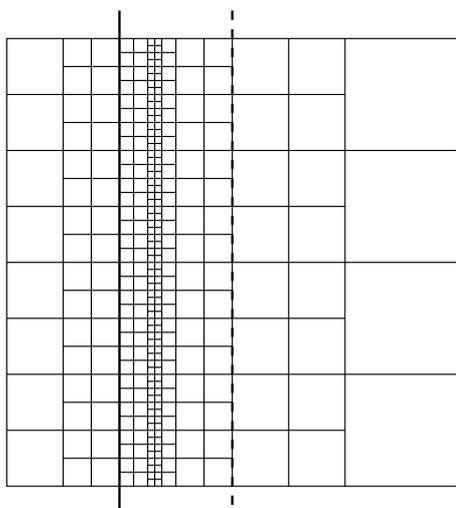


(b) Posição inicial do plano de partição do eixo X. Cargas em cada lado: 280 e 12. Diferença: 268. Carga na interface:  $16 + 8 = 24$ .

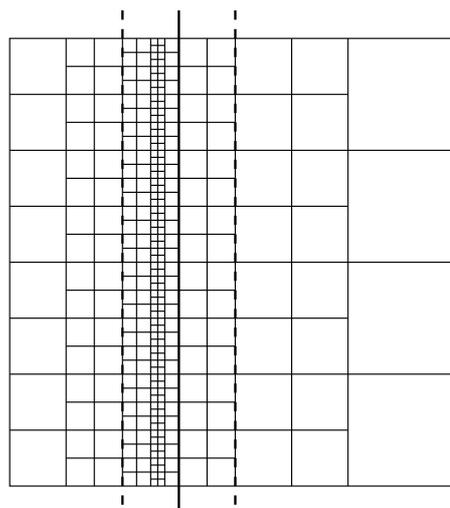
Figura 3.18 – Decomposição do domínio em cada eixo.

Posteriormente, as estimativas de cargas dos dois lados são calculadas, de acordo com o que foi descrito na Seção 3.3.1.2. Se as cargas dos dois lados forem iguais, essa posição do plano de partição é a melhor para esse eixo. Caso contrário, verifica-se qual dos dois lados é o mais pesado, e desloca-se o plano de partição para esse lado (Figura 3.18c).

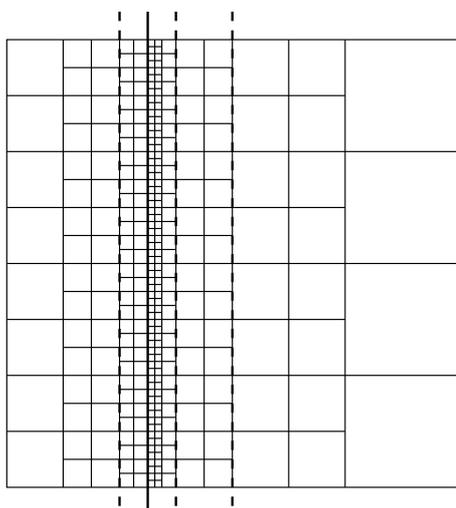
O objetivo do deslocamento do plano de partição é transferir trabalho do lado mais pesado para o lado mais leve, tentando-se equilibrar as estimativas cargas dos dois lados. Para fazer o deslocamento, aplica-se novamente o algoritmo de posicionamento do plano de partição, considerando-se apenas o conjunto de células do lado mais pesado. O conjunto de células do outro lado permanece fixo para esse lado, ou seja, essas células não serão consideradas em futuras execuções do algoritmo de posicionamento do plano de partição. Com isso, o reposicionamento do plano de partição corresponde à descida de um nível na estrutura de árvore da *octree*.



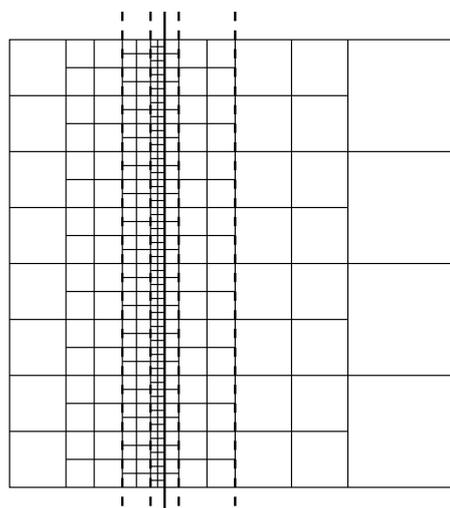
(c) Deslocamento do plano de partição do eixo X para a esquerda. Cargas em cada lado: 24 e 244. Diferença: 220. Carga na interface:  $16 + 32 = 48$ .



(d) Deslocamento do plano de partição do eixo X para a direita. Cargas em cada lado: 232 e 36. Diferença: 196. Carga na interface:  $32 + 16 = 48$ .



(e) Deslocamento do plano de partição do eixo X para a esquerda. Cargas em cada lado: 72 e 148. Diferença: 76. Carga na interface:  $32 + 64 = 96$ .

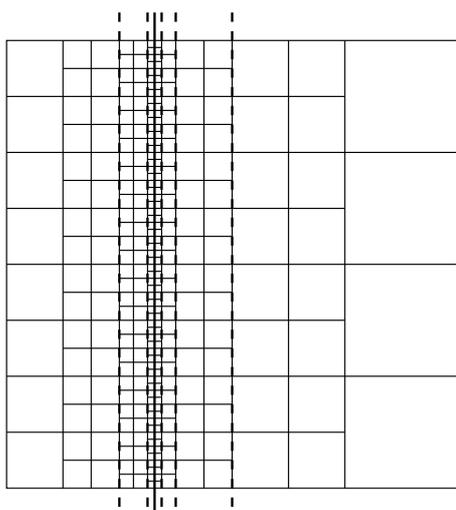


(f) Deslocamento do plano de partição do eixo X para a direita. Cargas em cada lado: 168 e 52. Diferença: 116. Carga na interface:  $64 + 32 = 96$ .

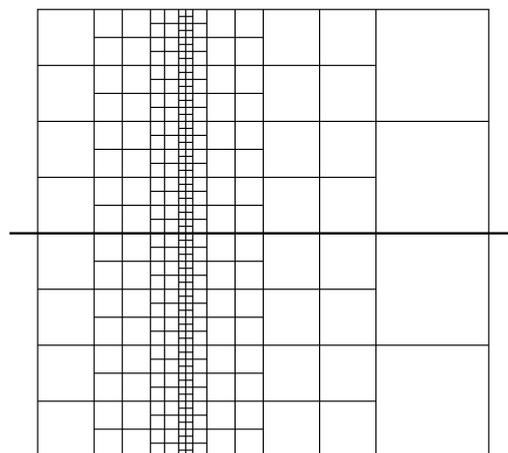
Figura 3.18 – Decomposição do domínio em cada eixo (continuação).

Esse procedimento é feito recursivamente, e tem dois critérios de parada, quando as cargas dos dois subdomínios são iguais ou quando a descida na estrutura da *octree* chega às folhas da *octree*. Das diversas posições testadas, é escolhida aquela que melhor tiver distribuído a estimativa de carga entre os dois lados, ou seja, aquela cuja diferença entre o lado mais pesado e o mais leve é a mais próxima de zero. Isso é feito para todos os eixos, e o melhor posicionamento de cada eixo é guardado. O melhor deles é escolhido em um procedimento que será descrito posteriormente.

As Figuras 3.18d a 3.18g reproduzem o funcionamento recursivo do algoritmo no eixo X, enquanto que a Figura 3.18h ilustra o mesmo processo para o eixo Y. Perceba que, nesse eixo, o primeiro posicionamento do plano de partição já iguala as estimativas de carga dos dois lados, fazendo com que o passo recursivo não seja executado nenhuma vez. Em um exemplo tridimensional, o eixo Z também é testado.



(g) Posição final do plano de partição do eixo X. Cargas em cada lado: 104 e 84. Diferença: 20. Carga na interface:  $64 + 64 = 128$ .



(h) Decomposição do domínio no eixo Y. Cargas em cada lado: 145 e 145. Diferença: 0. Carga na interface:  $13 + 13 = 26$ .

Figura 3.18 – Decomposição do domínio em cada eixo (continuação).

### 3.3.2.3 Melhor decomposição dentre os eixos

Uma vez determinados os melhores posicionamentos do plano de partição em cada eixo, o melhor dentre eles deve ser escolhido. Essa escolha é feita em quatro passos. Nos três primeiros passos, os eixos são comparados dois a dois, e os eixos considerados ruins são descartados. O quarto passo funciona como um desempate, caso exista mais de um eixo remanescente, ou seja, caso nenhum ou apenas um eixo tenha sido descartado.

Assim como na decomposição em cada eixo, o fator de maior importância é o equilíbrio entre as estimativas de carga dos dois lados. Portanto, dados dois eixos, aquele que melhor equilibra a estimativa de carga, ou seja, aquele que minimiza o módulo da diferença entre as estimativas de carga dos dois lados, é mantido, e o outro é descartado. Caso os dois eixos equilibrem igualmente as estimativas, o próximo passo é levado em consideração.

O próximo passo decide qual dos dois eixos é o melhor pela estimativa de carga da interface pois, quanto mais leve a interface, menor será o trabalho de geração de malha na interface e, portanto, menor o tempo de execução em um nó interno da BSP. O eixo que tiver maior carga na interface é descartado. Caso os dois eixos tenham mesma carga de interface, os dois são mantidos.

A execução desses dois passos nessa ordem indica um grau de prioridade: equilibrar os dois lados tem mais prioridade que minimizar a interface. Entretanto, isto pode não ser muito bom, pois, por exemplo, um eixo pode gerar dois lados leves, mas de mesma estimativa de carga, e uma interface muito pesada, enquanto o outro eixo pode gerar dois lados mais pesados, de diferentes estimativas de carga, mas de interface leve. Este segundo eixo pode ser melhor que o primeiro pois, se o primeiro fosse escolhido, a execução dos nós internos da BSP pode levar muito tempo, degradando o tempo de execução da técnica paralela.

Porém, somente inverter a ordem dos passos, testar primeiramente a estimativa de carga das interfaces e depois o equilíbrio entre os dois lados, pode apresentar um problema análogo ao anterior, ou seja, um eixo com interface mais leve mas muito desequilibrado pode ser escolhido. Nesse caso, por estarem desequilibradas, a geração de malha nas folhas da BSP pode acabar prejudicando o tempo de execução da técnica paralela.

Portanto, em muitos casos, pode ser aceitável um certo desequilíbrio entre os dois lados de um eixo,

desde que a interface deste eixo seja bem mais leve que a do outro eixo, que seria mais equilibrado. Logo, é necessário haver um balanceamento entre minimizar a estimativa de carga da interface e equilibrar as estimativas de carga dos dois lados, ou seja, dado que o particionamento em cada eixo é o mais equilibrado possível, a escolha do melhor dentre eles pode rejeitar um eixo mais equilibrado, porém com uma interface muito grande, em favor de outro eixo que não esteja tão equilibrado, mas que diminua consideravelmente o tamanho da interface.

Esse balanceamento é o teste de maior prioridade, e tem a seguinte interpretação: dados dois eixos, aquele que estiver relativamente equilibrado e tiver uma interface bem mais leve que o outro eixo é mantido, e o outro eixo é descartado. Caso os dois eixos estejam muito desequilibrados ou suas interfaces tenham aproximadamente a mesma estimativa de carga, os dois eixos são mantidos.

No balanceamento são feitos dois testes, um para verificar se um eixo é relativamente equilibrado e o outro para verificar se este eixo tem uma interface bem mais leve que o outro eixo. Ambos os testes são controlados por um valor limiar, dado como uma porcentagem.

O teste do equilíbrio relativo depende somente do próprio eixo, e é dado pelo módulo da diferença entre as estimativas de carga dos dois lado do eixo dividido pelo maior valor entre eles. Quanto mais equilibrados, menor a diferença, e a divisão tende a 0. Quanto mais desequilibrados, a diferença entre as estimativas tende ao maior dos dois valores, e a razão tende a 1. Assim, se esta divisão for menor que ou igual a um limiar, é dito que este eixo está relativamente equilibrado. A Equação 3.2 indica o teste do equilíbrio relativo, onde  $EC_{\alpha 1}$  e  $EC_{\alpha 2}$  são as estimativas de carga dos dois lados para o eixo  $\alpha \in \{X, Y, Z\}$  e  $t$  é o limiar:

$$Eq_{\alpha} = \frac{|EC_{\alpha 1} - EC_{\alpha 2}|}{\max\{EC_{\alpha 1}, EC_{\alpha 2}\}} \leq t. \quad (3.2)$$

O teste das interfaces compara os dois eixos, e é definido como a razão entre a estimativa de carga da interface do eixo pela estimativa de carga da interface do outro eixo. Assim, se a interface do primeiro eixo for bem mais leve que a interface do outro eixo, a razão tende a 0. Caso as duas interfaces tenham aproximadamente a mesma estimativa de carga, a razão tende a 1. Se a interface do primeiro eixo for bem mais pesada que a interface do outro eixo, a razão tende ao infinito. Assim, se esta razão for menor que ou igual a um limiar, é dito que este eixo tem interface bem mais leve que o outro eixo. A Equação 3.3 mostra relação entre as estimativas de carga das interfaces, onde  $EC_{\alpha I}$  e  $EC_{\beta I}$  são as estimativas de carga das interfaces para os eixos  $\alpha, \beta \in \{X, Y, Z\}$ , com  $\alpha \neq \beta$ , respectivamente, e  $t$  é o limiar:

$$I_{\alpha\beta} = \frac{EC_{\alpha I}}{EC_{\beta I}} \leq t. \quad (3.3)$$

Finalmente, a ordem dos três primeiros passos para decidir qual o melhor eixo é a seguinte (Algoritmo 3.1): o teste de maior prioridade é o que balanceia o equilíbrio entre os lados de um eixo e a carga da interface; depois o teste do equilíbrio entre os lados; e, por último, o teste da interface mais leve. Estes testes servem para descartar os eixos considerados ruins, mantendo os eixos considerados bons. Primeiramente, o eixo X é comparado com o Y e, posteriormente, o melhor deles (ou qualquer um deles, caso sejam equivalentes) é comparado com o eixo Z. Caso mais de um eixo seja mantido, o quarto passo é feito.

Este quarto passo, de desempate, é meramente topológico, pois depende somente da BSP, e não das estimativas de cargas. Caso essa decomposição seja a primeira de todas, ou seja, da raiz da BSP, o primeiro dos eixos mantidos é escolhido. Caso não seja a raiz, a célula-pai é encontrada, verifica-se qual o eixo que foi utilizado na sua decomposição, e escolhe-se um eixo diferente ao do pai, mais especificamente, o eixo seguinte dentre os mantidos. Este passo serve para que a decomposição não tenda a ser feita sempre no mesmo eixo.

**Algoritmo 3.1** Seleção de eixo(s)

**Entrada:** Eixos  $\alpha$  e  $\beta$ , estimativas de carga para os dois lados de cada eixo  $EC_{\alpha 1}$ ,  $EC_{\alpha 2}$ ,  $EC_{\beta 1}$  e  $EC_{\beta 2}$ , estimativas de carga para as interfaces de cada eixo  $EC_{\alpha I}$  e  $EC_{\beta I}$ , limiar  $t$ .

**Saída:** Eixo(s) selecionado(s).

▷ Cálculo dos fatores de balanço

$$d_{\alpha} \leftarrow |EC_{\alpha 1} - EC_{\alpha 2}|$$

$$d_{\beta} \leftarrow |EC_{\beta 1} - EC_{\beta 2}|$$

$$Eq_{\alpha} \leftarrow d_{\alpha} / \max\{EC_{\alpha 1}, EC_{\alpha 2}\}$$

$$Eq_{\beta} \leftarrow d_{\beta} / \max\{EC_{\beta 1}, EC_{\beta 2}\}$$

$$I_{\alpha\beta} \leftarrow EC_{\alpha I} / EC_{\beta I}$$

$$I_{\beta\alpha} \leftarrow EC_{\beta I} / EC_{\alpha I}$$

▷ Testes usando os fatores de balanço e o limiar

**se**  $Eq_{\alpha} \leq t$  e  $I_{\alpha\beta} \leq t$  **então retorne**  $\alpha$

**se**  $Eq_{\beta} \leq t$  e  $I_{\beta\alpha} \leq t$  **então retorne**  $\beta$

▷ Testes usando o balanço entre os lados

**se**  $Eq_{\alpha} < Eq_{\beta}$  **então retorne**  $\alpha$

**se**  $Eq_{\beta} < Eq_{\alpha}$  **então retorne**  $\beta$

▷ Testes usando as interfaces

**se**  $EC_{\alpha I} < EC_{\beta I}$  **então retorne**  $\alpha$

**se**  $EC_{\beta I} < EC_{\alpha I}$  **então retorne**  $\beta$

▷ Retorno de eixos igualmente balanceados

**retorne**  $\alpha$  e  $\beta$

No exemplo da Figura 3.18, o eixo X tem equilíbrio relativo  $\frac{|104 - 84|}{\max\{104, 84\}} = \frac{20}{104} \cong 19,23\%$  e carga de interface relativa  $\frac{128}{26} \cong 492,30\%$ . Já o eixo Y tem equilíbrio relativo  $\frac{|145 - 145|}{\max\{145, 145\}} = \frac{0}{145} = 0\%$  e carga de interface relativa  $\frac{26}{128} \cong 20,31\%$ . Assim, um limiar de 25% escolheria o eixo Y já no primeiro passo, descartando o eixo X. Já um limiar de 20%, por exemplo, manteria os dois eixos no primeiro passo, e o eixo X seria descartado no segundo passo, por ter equilíbrio  $|104 - 84| = 20$ , enquanto que o eixo Y tem equilíbrio  $|145 - 145| = 0$ . Nesse exemplo, nem o terceiro nem o quarto passo são necessários.

Por último, balancear o trabalho feito na interface com o trabalho feito nos subdomínios é um problema bastante difícil em algoritmos paralelos pois, de uma maneira geral, o desafio é equivalente a balancear a quantidade de comunicação entre os processos com a quantidade de computação feita nos processos. Este balanceamento varia de acordo com o problema e com os exemplos que estão sendo testados e é, muitas vezes, determinado a partir de observações e resultados empíricos. Assim, neste trabalho, o valor do limiar encontrado depois de alguns testes foi de 50%.

A Figura 3.19 mostra a decomposição feita no modelo de uma esfera uniforme. Perceba que os subdomínios mais externos são mais largos que os mais internos. Isso acontece porque, na decomposição do eixo X, o posicionamento do plano de partição no centro geométrico deixaria os subdomínios mais externos mais leves que os mais internos. Assim, durante a decomposição, por conta da remoção de uma camada de células da *octree* no cálculo da carga, o posicionamento é melhor ajustado.

### 3.3.2.4 Proporcionalidade nas estimativas de carga

A decomposição descrita nesta seção é realizada até que cada processo seja responsável por um único subdomínio. Se o número de processos for potência de 2, então os subdomínios gerados têm aproximadamente a mesma estimativa de carga pois, a cada subdivisão de uma célula da BSP, a estimativa de carga é aproximadamente subdividida ao meio. Entretanto, quando o número de processos não for potência de 2, alguns dos subdomínios terão estimativa de carga com aproximadamente o dobro da estimativa de carga de outros subdomínios (Figura 3.20).

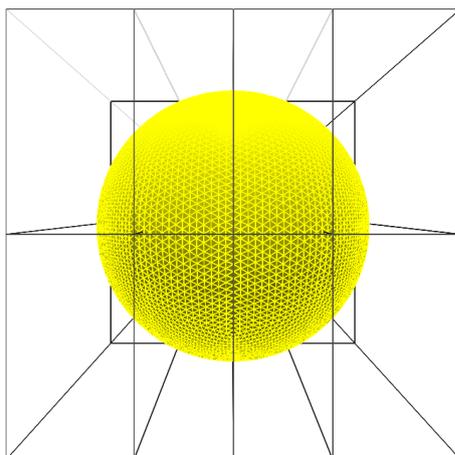


Figura 3.19 – Decomposição no modelo de uma esfera em 16 subdomínios.

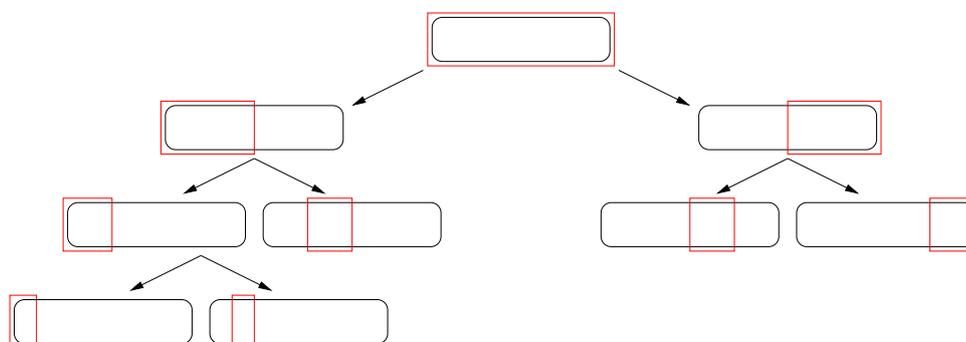


Figura 3.20 – Exemplo de árvore BSP criada para 5 subdomínios.

Para evitar esses casos e fazer com que os subdomínios tenham todos aproximadamente a mesma estimativa de carga, independentemente do número de processos, a decomposição utiliza um fator de proporcionalidade nas estimativas de carga. Sempre que as estimativas de carga dos dois lados forem comparadas uma com a outra, seja na decomposição de um eixo, seja na escolha do melhor eixo, elas são multiplicadas por um fator de proporcionalidade que denota quantas vezes um lado deve ser mais pesado que o outro.

O fator de proporcionalidade é dado da seguinte maneira. Seja  $P$  o número desejado de subdomínios. Então, um dos lados deverá ter proporcionalidade  $m = \lfloor P/2 \rfloor$  e o outro,  $M = \lceil P/2 \rceil$ , pois deseja-se que um dos lados, o mais leve, corresponda a  $m$  subdomínios, enquanto que o outro, o mais pesado, corresponda a  $M$  subdomínios. Assim, na comparação das estimativas de cargas dos dois lados, a estimativa de carga do lado mais leve é dividido por  $m$  e a estimativa do mais pesado, por  $M$  (ou, equivalentemente, a estimativa do lado mais leve deve ser multiplicada por  $M$  e a do mais pesado, por  $m$ ).

Depois de subdividir uma célula da BSP, as subdivisão dos seus filhos devem obedecer a proporcionalidade dada anteriormente, ou seja, o número desejado de subdomínios para o filho mais leve é  $m$  e, para o filho mais pesado, é  $M$ . Este fator de proporcionalidade  $m : M$  desequilibra propositalmente os nós internos da BSP com o objetivo de fazer com que as folhas da BSP, que são os subdomínios, tenham todos proporcionalidade 1, ou seja, tenham aproximadamente a mesma estimativa de carga.

Por exemplo, considere que o número de processos seja 5 (Figura 3.21). Quando se for subdividir a raiz da BSP, em vez de se comparar se as estimativas de carga dos dois filhos são iguais, compara-se se a estimativa de um deles, multiplicada por 2, é igual à estimativa do outro, multiplicada por 3, ou seja, utiliza-se o fator de proporcionalidade 2 : 3 para a raiz. O filho mais pesado da raiz deverá gerar

3 subdomínios, então terá fator de proporcionalidade  $1 : 2$ . Deste, apenas um dos filhos, o mais pesado, será novamente subdividido, com fator  $1 : 1$ . O segundo filho da raiz, o mais leve, também terá fator de proporcionalidade  $1 : 1$ . Com isso, serão gerados 5 subdomínios, todos com aproximadamente a mesma estimativa de carga. Quando o número de processos for potência de 2,  $m = M$ , que também é potência de 2. É importante, ainda, mencionar que nenhum dos trabalhos relacionados utiliza algum tipo de fator de proporcionalidade entre as estimativas de cargas dos subdomínios, como o descrito nesta seção.

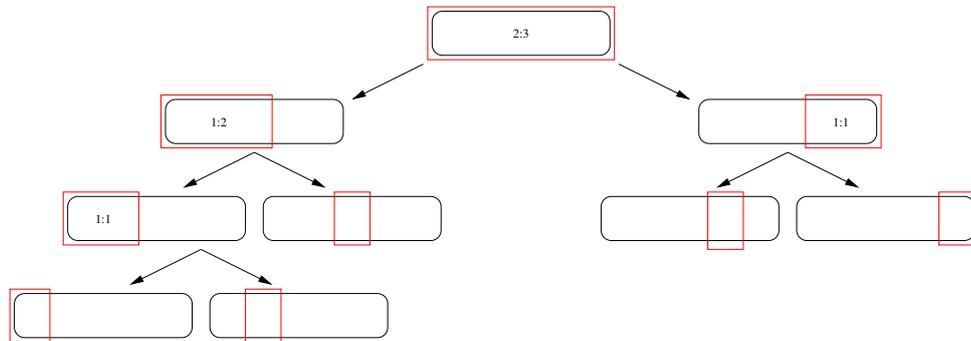


Figura 3.21 – Exemplo de árvore BSP criada com fator de proporcionalidade para 5 subdomínios. Os fatores estão indicados em cada nó interno da BSP.

### 3.3.2.5 Decomposição em paralelo do domínio

Como dito anteriormente, o procedimento de subdivisão da BSP é feito até que o número de subdomínios seja igual ao número de processos. Por ser um procedimento local, não é necessário que um processo mestre fique dedicado à geração da BSP, podendo esta ser gerada ao mesmo tempo em todos os processos, desde que cada um deles consiga determinar qual a sua própria célula da BSP.

Isso pode facilmente ser feito utilizando-se o identificador de um processo e o número total  $P$  de processos. Na subdivisão de uma célula da BSP, dos  $P$  processos,  $p$  são alocados para o seu primeiro filho e  $P - p$ , para o segundo filho. Logo, caso  $P$  seja par,  $p = P/2$ , e os processos de identificadores 0 a  $p - 1$  são alocados para o primeiro filho, e os processos restantes, de identificadores  $p$  a  $P - 1$ , são alocados para o segundo filho. Caso  $P$  seja ímpar, o filho mais pesado deve ficar com o processo extra, ou seja, se o filho mais pesado for o primeiro,  $p = \lceil P/2 \rceil$  e, se o filho mais pesado for o segundo,  $p = \lfloor P/2 \rfloor$ .

Dessa forma, cada processo subdivide somente o ramo da BSP que interessa a ele, não precisando subdividir a BSP em um ramo onde ele não vai gerar malha. Como vários processos acabam compartilhando um mesmo nó interno da BSP, deve-se estabelecer um deles para ser o responsável por este nó. Neste trabalho, o processo escolhido é o de menor identificador, fazendo com que o processo 0 fique responsável pelo primeiro filho da raiz e o processo  $p$ , pelo seu segundo filho.

Caso o paralelismo de memória compartilhada esteja ativado, continua-se a subdivisão da BSP em cada *thread*, tomando cuidado para não haver escrita simultânea de dados na mesma posição de memória. Nesse caso, o algoritmo é praticamente o mesmo, utilizando-se o identificador da *thread* e o número de *threads* em vez do identificador do processo e do número de processos, respectivamente.

A decomposição em paralelo é melhor ilustrada nas Figuras 3.22 e 3.23. A Figura 3.22 mostra uma visão global da BSP e quais são os processos/*threads* responsáveis por cada nó, interno ou folha, da BSP. A Figura 3.23 mostra a visão local da BSP em cada processo. Os rótulos indicam os processos (P) ou as *threads* (T) responsáveis por um determinado ramo exterior ao ramo do próprio processo, ou seja, com quem o processo local deve se comunicar e trocar mensagens durante a geração da malha nas interfaces, a ser descrita na próxima seção.

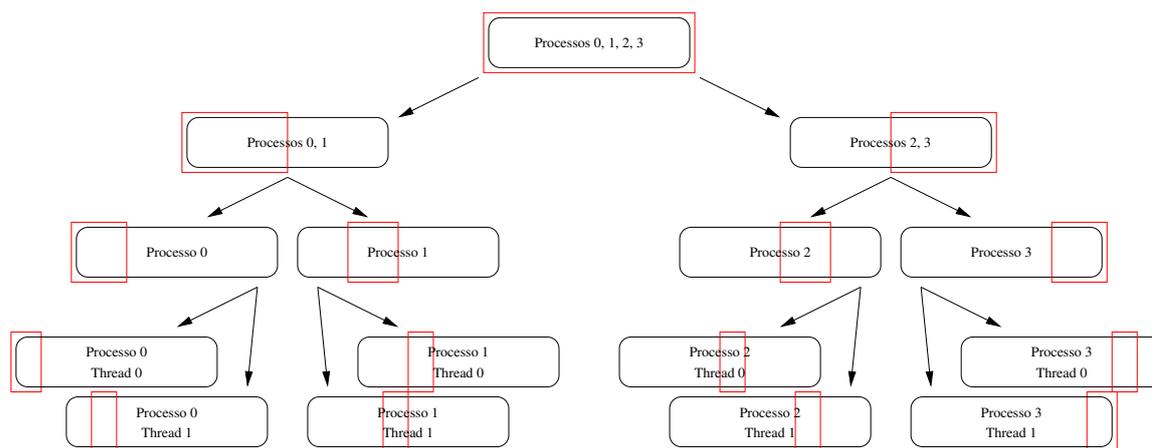


Figura 3.22 – Visão global da BSP criada para um domínio dado em preto. Os subdomínios dos processos ou *threads* são mostrados em vermelho.

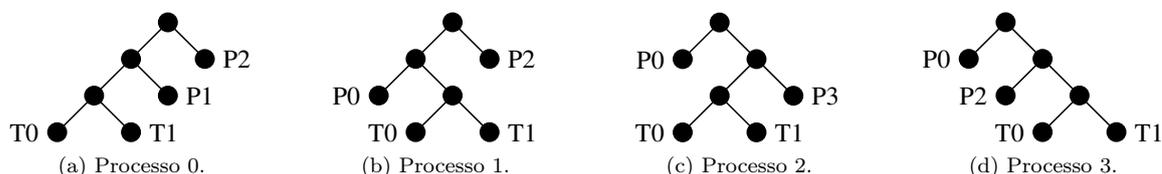


Figura 3.23 – BSPs locais criadas em cada processo.

### 3.3.3 Geração da malha

Uma vez que a BSP tenha sido gerada, cada processo inicia a geração da malha em um subdomínio. Após gerar a malha nos subdomínios, as malhas de interface, são geradas, conectando os subdomínios. Quando a malha da raiz da BSP tiver sido gerada, a malha está completamente gerada, distribuída pelos processos. Caso a geração da malha na raiz falhe, a técnica paralela falha, e a malha não é gerada. Entretanto, como dito anteriormente, isso raramente acontece em modelos práticos, e não foi observado em nenhum dos exemplos do Capítulo 4.

#### 3.3.3.1 Fronteira inicial de um subdomínio

A entrada para o algoritmo de geração de malha é uma fronteira. Nos subdomínios que cruzam a fronteira original, as faces que estão completamente dentro do subdomínio pertencem a ele, e as que cruzam o subdomínio são compartilhadas entre ele e pelo menos mais um subdomínio. A Figura 3.24 ilustra a decomposição da fronteira, em que as faces totalmente dentro do subdomínio estão coloridas de verde e as faces compartilhadas estão em vermelho.

É possível, entretanto, que alguns dos subdomínios sejam totalmente internos ao modelo, e não tenham fronteira inicial. Esse problema é resolvido com a criação de 8 faces auxiliares, próximas ao centroide do subdomínio no formato de uma trinca, quatro delas com normal para cima e quatro com normal para baixo, no eixo Y (Figura 3.25). Se o gerador de malha não suportasse trinca, o vértice central poderia sofrer um pequeno deslocamento para cima, nas faces de cima, e para baixo, nas faces de baixo.

Após a geração da malha no subdomínio, as faces auxiliares são apagadas e a estrutura da malha é atualizada para ajustar a nova vizinhança de vértices e elementos.

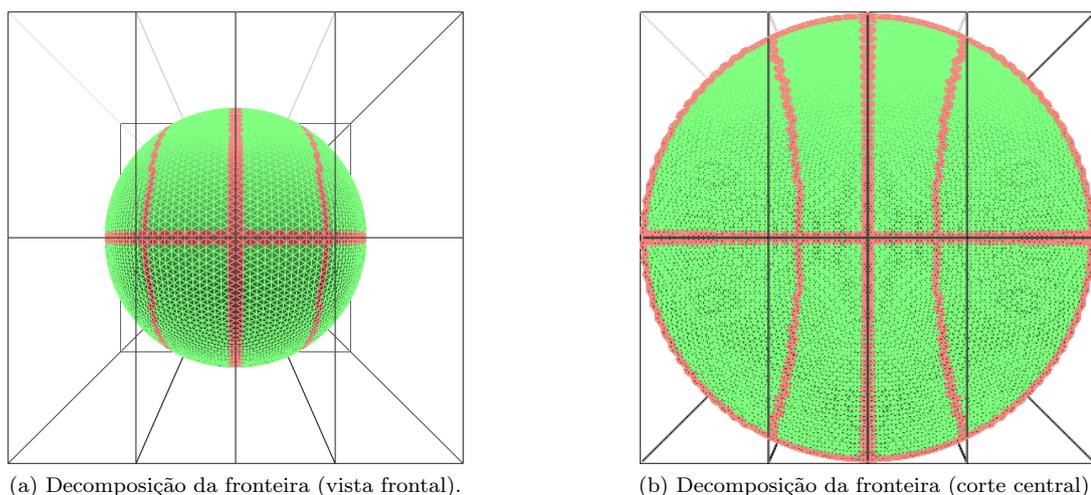


Figura 3.24 – Decomposição da fronteira de uma esfera.

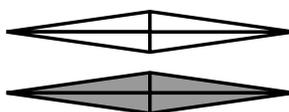


Figura 3.25 – Exemplo de fronteira inicial em forma de trinca. A distância entre vértices e faces mostrada é apenas esquemática e não existe na realidade.

### 3.3.3.2 Geração da malha em um subdomínio

Para gerar a malha em um subdomínio, a técnica sequencial é utilizada (Seção 3.2.2), juntamente com a melhoria aplicada na malha (Seção 3.2.3), com algumas modificações para certificar que os elementos gerados estejam completamente contidos na região definida pelo subdomínio. As modificações são as seguintes:

1. Uma face que cruza ou toca o subdomínio não pode participar do avanço de fronteira (Figura 3.26a);
2. Uma face cuja região de busca esférica cruzar o subdomínio também não pode avançar (Figura 3.26b). Quando isto acontece, um vértice que formaria um tetraedro bom poderia estar em um subdomínio vizinho, que não seria encontrado localmente no processo. Algumas observações devem ser feitas:
  - Como a restrição é que nenhum tetraedro formado cruze o subdomínio, poderia-se mudar o teste para que somente o ponto ideal de uma face não esteja fora dele, em vez de a região de busca. No entanto, isso poderia levar à geração de elementos muito ruins entre dois subdomínios, na geração das malhas de interface, uma vez que o espaço livre para a criação dos tetraedros seria consideravelmente reduzido;
  - A etapa baseada em topologia também deve obedecer essa restrição, ou seja, esta etapa deve saber como construir a região de busca da etapa baseada em geometria para decidir se ela cruza ou não a borda do subdomínio, mesmo que não a utilize para a geração dos elementos;
3. Após avançar o máximo que puder, pelas etapas baseadas em geometria e em topologia, as faces remanescentes formam a fronteira atualizada (Figura 3.27), que será futuramente utilizada na geração das malhas de interface;
4. Durante a etapa de avanço de fronteira por retrocesso, se uma das faces da fronteira atualizada for atingida, este retrocesso é cancelado;
5. A fronteira atualizada não pode ser modificada pela etapa de melhoria da malha. Isto ocorre porque algumas informações de vizinhança de vértices ou elementos, necessárias para a melhoria, estão

ausentes ou ainda não foram determinadas. Logo, tanto a suavização quanto as trocas de arestas e de faces são rejeitadas quando aplicadas a vértices, arestas ou faces desta fronteira atualizada.

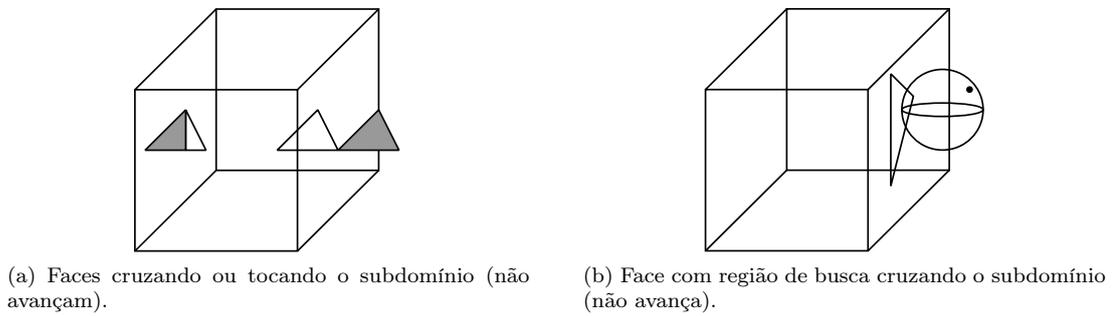


Figura 3.26 – Restrições no avanço de fronteira.

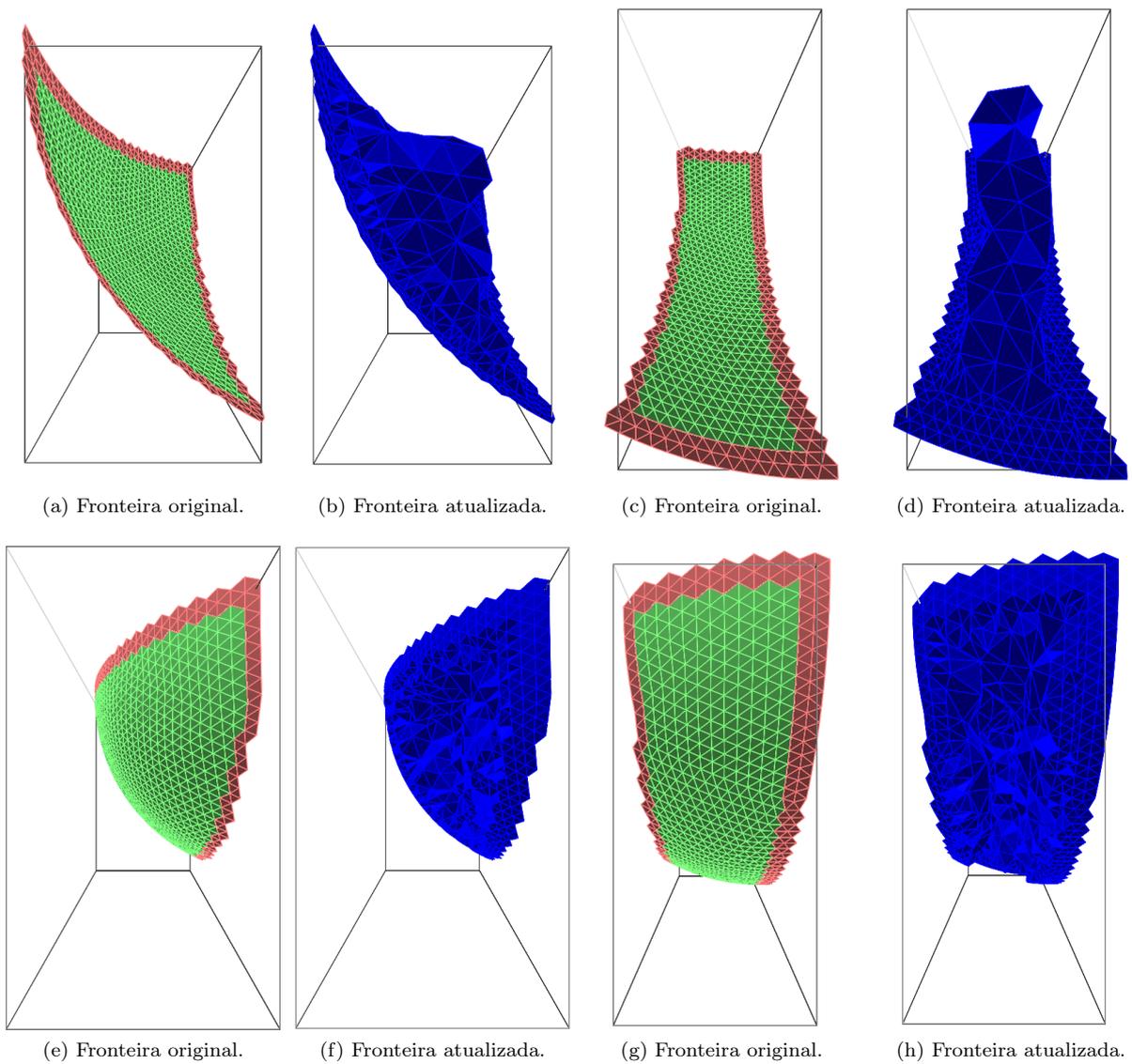


Figura 3.27 – Fronteiras original e atualizada de diversos subdomínios do modelo da esfera.

A Figura 3.28 mostra as malhas geradas em alguns dos 16 subdomínios do modelo da esfera, onde cada cor representa um processo ou uma *thread* diferente.

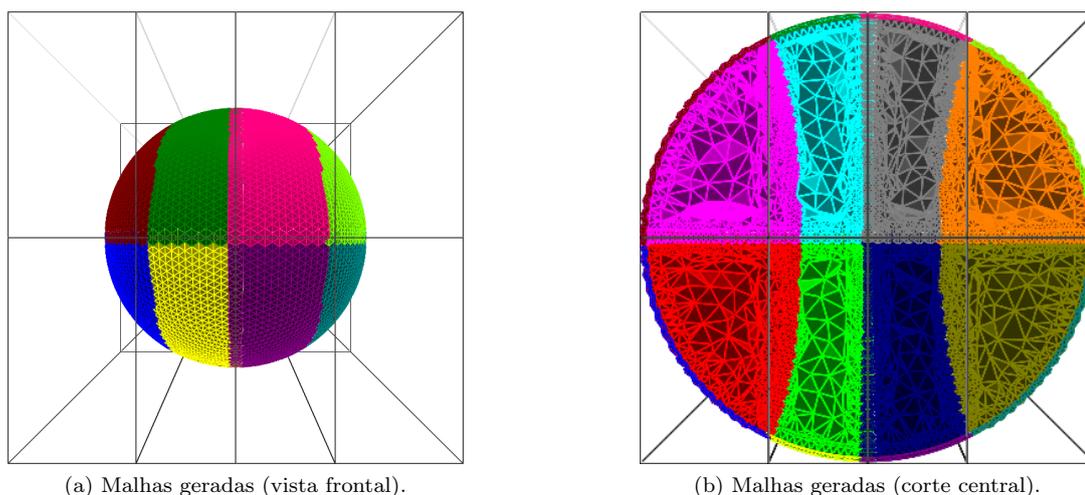


Figura 3.28 – Malhas geradas nos subdomínios do modelo da esfera.

### 3.3.3.3 Geração da malha entre subdomínios

Para gerar a malha entre dois subdomínios, ou seja, a malha de interface, é necessário haver uma sincronização local entre os processos. Pela forma como a BSP foi construída, é sempre possível que um processo responsável por um nó, folha ou interno, saiba qual é o processo responsável pelo seu nó-irmão. Desses dois processos, o de menor identificador é o responsável pelo nó-pai.

Depois de esses dois processos gerarem as malhas nos seus subdomínios, eles sincronizam e o de maior identificador envia ao de menor identificador a fronteira atualizada, juntamente com os elementos adjacentes aos vértices da fronteira atualizada. O processo de menor identificador finaliza sua fase de geração de malha, guardando o restante da malha que foi gerada para o seu subdomínio.

O processo de menor identificador, por sua vez, altera a geometria do seu subdomínio, incorporando também a geometria do subdomínio irmão, ou seja, a geometria do subdomínio passa a ser a mesma da geometria da célula-pai. Este mesmo processo, então, gera a malha na região vazia correspondente à interface entre os dois subdomínios anteriores, aplicando também uma melhoria nessa malha de interface.

Para a geração dessa malha de interface, a camada de elementos recebida e uma camada de elementos determinada pelo próprio processo são também dadas como entrada para o gerador de malha, para que a geração de malhas por retrocesso tenha liberdade de remover elementos e, principalmente, para que as fronteiras atualizadas dos dois subdomínios anteriores possam sofrer melhorias.

Posteriormente, este processo entra em espera para sincronizar com o processo que ficou responsável pelo seu nó-irmão, e este mesmo procedimento é executado, subindo-se na estrutura de árvore da BSP. Quando o processo de identificador 0 chega ao nó-raiz da BSP, as restrições de geração de malha apresentadas nesta seção, que dizem respeito à geometria do subdomínio, não são mais necessárias e, portanto, não são aplicadas.

Depois que o processo 0 gerar malha na raiz da BSP, o procedimento paralelo termina. É teoricamente possível, entretanto, que algum processo não consiga gerar a malha completa no seu subdomínio, fazendo com que essa fronteira acabe chegando à raiz da BSP. Neste caso, é feita uma última tentativa de finalizar a malha, com uma segunda aplicação do avanço de fronteira por retrocesso e, caso a malha ainda assim não seja gerada, o algoritmo falha.

Como a geração em paralelo da malha termina no processo 0, ele pode ser considerado como o processo mestre. O procedimento de geração das malhas nas interfaces pode ser visto na Figura 3.29.

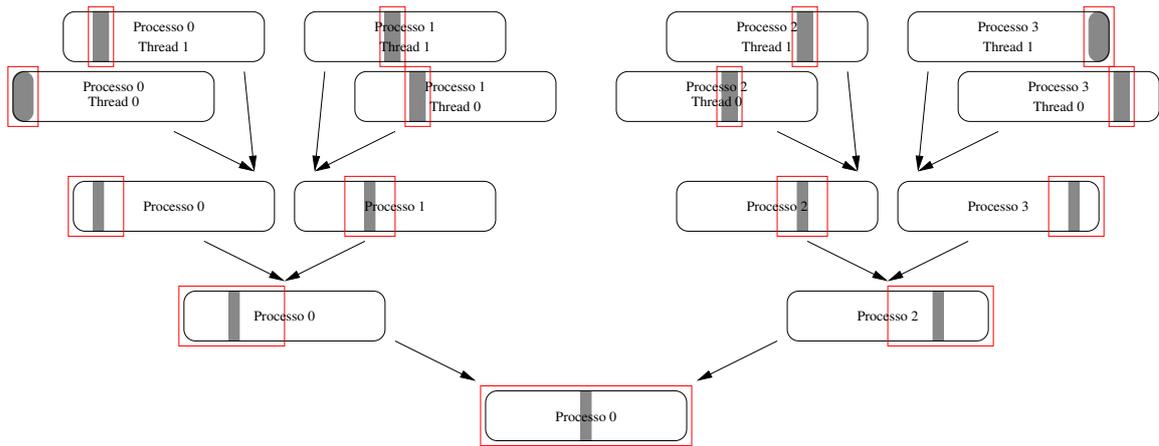


Figura 3.29 – Geração das malhas nas *threads* e nos processos. A malha gerada está representada em cinza.

### 3.4 Considerações finais

Este capítulo descreveu em detalhes uma técnica paralela de geração de malhas utilizando uma BSP para decompor o domínio. Essa BSP é construída com base em uma *octree* utilizada para estimar a carga e, portanto, os subdomínios gerados têm aproximadamente a mesma quantidade de trabalho.

Este algoritmo acontece em  $O(\lceil \log_2 P \rceil + 1)$  passos de geração de malha, onde  $P$  é o número de processos/*threads*. Desses passos, um deles leva mais tempo que os outros, o de geração das malhas nas folhas da BSP. Os outros  $\lceil \log_2 P \rceil$  passos, de geração das malhas de interface, relativos aos nós internos da BSP, devem acontecer rapidamente, pois essas regiões são menores, quando comparadas com as regiões das folhas da BSP. Perceba, inclusive, que alguns processos podem estar operando em nós internos enquanto que outros ainda estão operando nas folhas da BSP.

O próximo capítulo apresenta os resultados obtidos a partir da implementação da técnica descrita neste capítulo. Uma análise teórica da técnica paralela pode ser encontrada no Apêndice A.

## 4 EXEMPLOS E RESULTADOS

### 4.1 Introdução

Este capítulo apresenta os resultados obtidos com a implementação da técnica descrita neste trabalho. Estes resultados englobam tanto aspectos de geometria computacional, como a qualidade da malha, quanto aspectos de computação de alto desempenho, como o tempo de execução e o *speed-up* da implementação.

O programa foi desenvolvido em C++ (The C++ Standards Committee, 2015), utilizando a biblioteca de paralelismo em memória compartilhada OpenMP (Open Multi-Processing) (OpenMP Architecture Review Board, 2015) e a biblioteca de passagem de mensagens MPI (Message Passing Interface) (MPI Forum, 2015) para paralelismo de memória distribuída. Na versão com interface, foram ainda utilizadas a biblioteca de renderização OpenGL (The OpenGL Architecture Review Board, 2015) (com suas bibliotecas glu, glut (KILGARD, 2015) e glew (IKITS; MAGALLON, 2015)), a biblioteca de interfaces wxWidgets (wxTeam, 2015) e a biblioteca de para gravação de vídeo OpenCV (OpenCV Dev Team, 2015).

O gerador de malhas utilizado foi desenvolvido pelo Cornell Fracture Group (CFG) (WARNER; INGRAFFEA, 2015), da Universidade de Cornell. O gerador sequencial foi adaptado para poder rodar em paralelo, bem como otimizado em alguns pontos específicos.

Apenas a versão de memória distribuída foi testada, apesar de a versão de memória compartilhada também funcionar. O *cluster* utilizado foi o Stampede (The University of Texas at Austin, 2015), pertencente ao Texas Advanced Computing Center, da Universidade do Texas em Austin, e utilizado através da parceria com o CFG. O compilador utilizado foi o g++ 4.7.1 (Free Software Foundation, Inc, 2015), com suporte ao C++11, e a versão do MPI foi o MVAPICH2 1.9a2 (The Ohio State University, 2015).

Cada nó do *cluster* é equipado com dois processadores Intel® Xeon™ E5-2680 de 2,7 GHz e 8 placas de memória RAM DDR3 com 4 canais, cada uma com 4 GB e de 1300 MHz, totalizando 16 núcleos e 32 GB de memória. Os nós são conectados por uma rede InfiniBand FDR de 56 GB/s com *switches* Mellanox, formando uma topologia de *fat tree* de 2 níveis, como visto na Figura 4.1. Dessa forma, uma mensagem entre dois processos necessita de, no máximo, 5 saltos para chegar ao destino. Os testes foram feitos com até 16 processos, utilizando o máximo de processos por nó, desde que não exceda a memória.

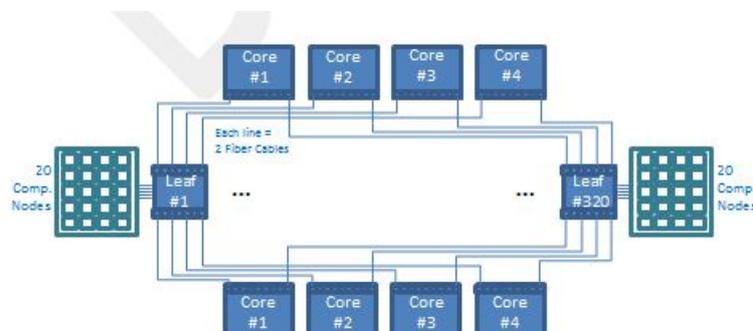


Figura 4.1 – Topologia da rede do *cluster* Stampede. Fonte: <[www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide](http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide)>.

## 4.2 Exemplos

Os exemplos considerados neste capítulo englobam uma geometria teórica (uma Viga) e três geometrias de aplicação prática, uma delas com fratura (uma Engrenagem) e dois modelos sem fraturas (um Rotor, e uma Engrenagem semelhante à anterior) (Figura 4.2).

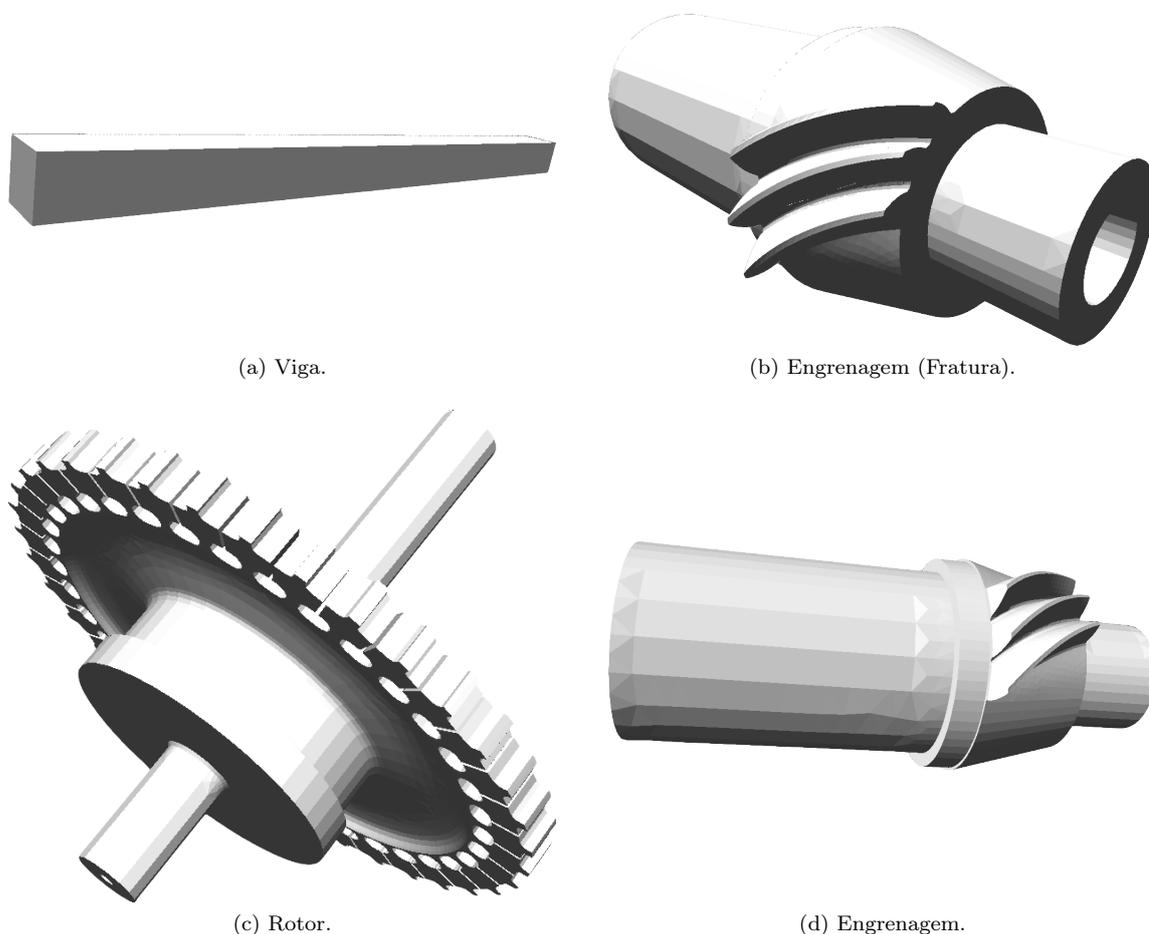


Figura 4.2 – Exemplos.

O tamanho das malhas geradas foi definido baseado em alguns critérios. O primeiro deles é que a malha deve ser grande o suficiente para compensar o uso de paralelismo. Em segundo lugar, o gerador de malhas sequencial deve conseguir gerar a malha do tamanho desejado, para que seja possível o cálculo do *speed-up*. Portanto, a quantidade de memória utilizada na geração da malha sequencial não pode ser maior que a disponível em um nó do *cluster*. Em terceiro lugar, deseja-se sempre minimizar o número de nós utilizados, uma vez que a utilização do *cluster* é taxada em um valor fixo por nó, independentemente do número de processos rodando neste nó. Por conta dessa taxação, cada configuração de cada modelo foi executada somente uma vez.

Somente a malha da Engrenagem com fratura foi gerada com o refinamento uniforme da *octree* de densidade desativado, pois o gerador sequencial falhou em gerar a malha com este refinamento ativado. Na técnica paralela, todos os modelos precisaram usar até dois nós do *cluster* para gerar suas malhas. No modelo da Viga, o primeiro nó comportou até 10 processos e o segundo, o restante dos processos, quando 12, 14 ou 16 processos foram utilizados. Nos outros três modelos, o primeiro nó do *cluster* comportou até 8 processos, e o segundo, os processos restantes.

O modelo da Viga foi utilizado para se determinar o valor do limiar de balanceamento entre as cargas dos subdomínios e as cargas das interfaces, na escolha do melhor eixo de posicionamento do plano de

partição, durante a decomposição do domínio (Seção 3.3.2.3), e foi o modelo que gerou a maior malha, com pouco mais de 9 milhões de elementos. Além disso, com o objetivo de se perceber melhor o efeito dos vários valores do limiar, este modelo foi testado com 2, 4, 6, 8, 10, 12, 14 e 16 processos, enquanto os outros foram testados com apenas 2, 4, 8 e 16 processos.

### 4.3 Viga

A Figura 4.3 mostra detalhes do modelo da Viga, como sua discretização e sua borda. O modelo é alinhado aos eixos X, Y e Z, tem a geometria de um quadrado nos eixos Y e Z e é comprido no eixo X.

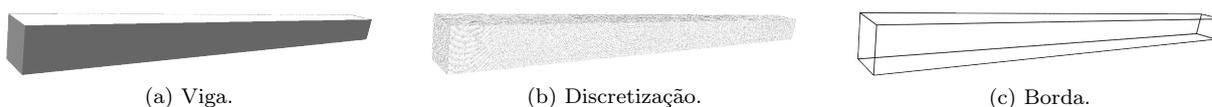


Figura 4.3 – Modelo da Viga.

Para se determinar o valor do limiar, a técnica paralela foi testada neste modelo com vários valores para o limiar. No teste mais básico, não foi atribuída nenhuma estimativa de carga à interface entre os subdomínios, ou seja, o equilíbrio entre as estimativas de carga dos dois filhos foi o único fator considerado para o posicionamento do plano de partição. Para isso, nenhuma camada de células da *octree* adjacente à interface é atribuída como estimativa de carga da interface. Como consequência, todas as células cheias da *octree* dentro de uma célula da BSP são a estimativa de carga dessa célula.

No segundo teste mais básico, uma camada de células adjacentes à interface foi removida de cada lado do plano de partição e atribuída a ele, contabilizando para a estimativa de carga da interface. Entretanto, o limiar foi mantido em 0%. Assim, a estimativa de carga da interface passa a servir como desempate quando dois eixos equilibram igualmente as estimativas de carga dos seus lados. O limiar foi, então, variado até 50%, em intervalos de 5%, ativando o balanceamento entre o equilíbrio dos dois filhos e a estimativa de carga da interface. Estão sendo mostrados nesta seção apenas os valores de limiar que modificaram a decomposição do domínio e tiveram impacto no desempenho da técnica paralela.

A Figura 4.4 mostra as decomposições de cada teste feito no modelo da Viga para 16 processos. As decomposições com os limiares de 15% e 20% foram as mesmas, bem como com os limiares de 25% e 40%. Observe que, dada a geometria comportada (sem reentrâncias nem saliências), comprida e fina do modelo, a decomposição mais intuitiva é dividir o modelo somente no eixo X (Figuras 4.4i e 4.4j).

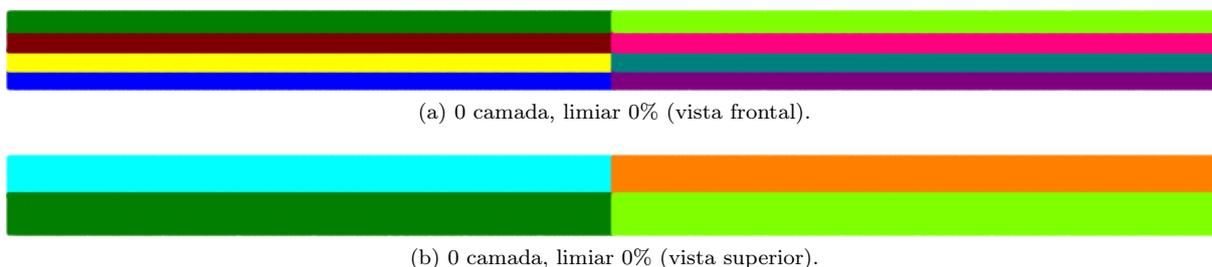


Figura 4.4 – Decomposições do modelo da Viga com 16 processos.

#### 4.3.1 Tempo de execução e *speed-up*

A Figura 4.5a mostra o tempo de execução obtido nas diversas execuções da implementação paralela. Apesar de útil para ver o tempo de execução absoluto, este gráfico não dá uma ideia muito boa de quantas vezes o algoritmo paralelo é mais rápido que o sequencial. O melhor tipo de gráfico para fazer



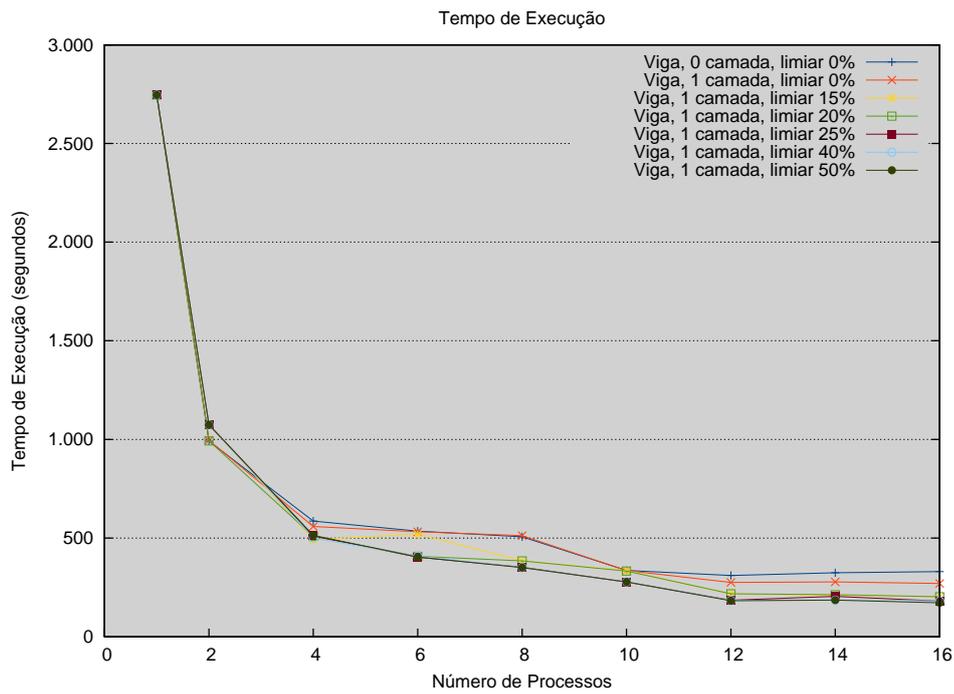
Figura 4.4 – Decomposições do modelo da Viga com 16 processos (continuação).

essa comparação é o *speed-up*, que é definido como a razão entre o tempo de execução da implementação sequencial pelo tempo de execução da implementação paralela com uma determinada quantidade de processos. O *speed-up* das várias execuções do modelo da Viga pode ser visto na Figura 4.5b.

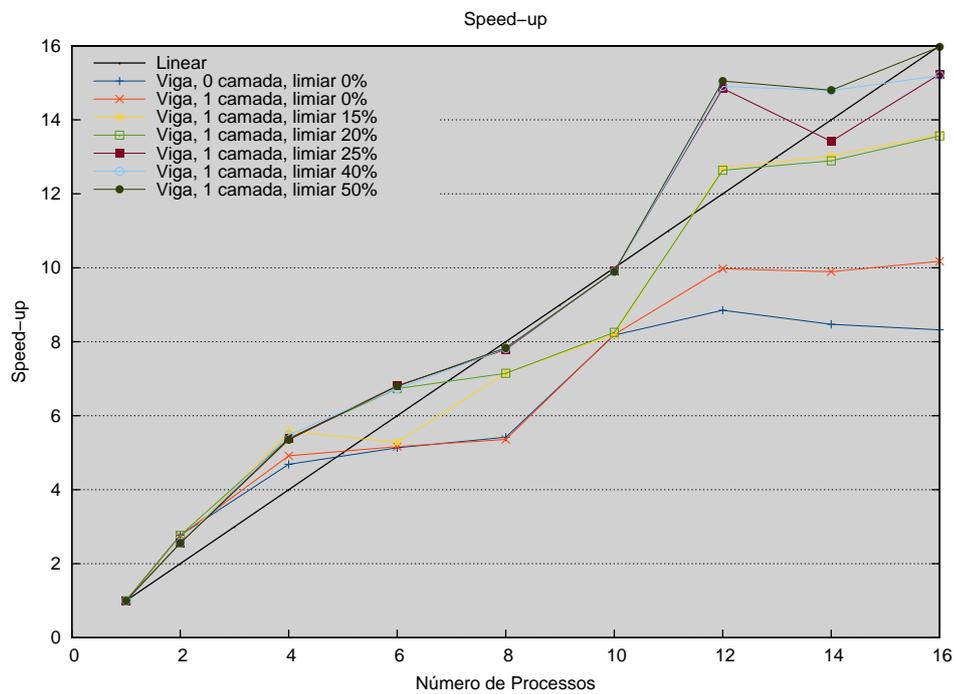
Como pode-se ver nos gráficos, conforme aumenta-se o limiar, diminui-se o tempo de execução, e o algoritmo paralelo aproxima-se do *speed-up* linear. O *speed-up* linear indica que, com  $P$  processos, a implementação paralela foi  $P$  vezes mais rápida que a implementação sequencial, e é o *speed-up* ideal dos algoritmos paralelos, embora seja geralmente difícil de se obter na prática, por conta de porções sequenciais ou tempo de comunicação da implementação paralela.

Com poucos processos, especificamente, com 2, 4 e 6 processos (em alguns casos), a implementação paralela obteve *speed-up* super-linear. Isso é um indício de que otimizações ainda podem ser feitas na implementação sequencial. Outro indício de uma implementação sequencial não otimizada é o rápido caimento do gráfico de *speed-up*, como acontece nos casos com limiar baixo. Entretanto, tal questão pôde ser superada por um melhor ajuste do limiar.

Os casos em que o número de processos não é potência de 2, principalmente o caso de 12 processos, apresentam um comportamento irregular por conta do pouco controle em sua decomposição, devido ao fator de proporcionalidade inserido na estimativa da carga (Seção 3.3.2.4). Apesar disso, os valores de



(a) Tempo de execução.



(b) Speed-up.

Figura 4.5 – Tempo de execução e speed-up da Viga.

speed-up foram consistentes com o restante do gráfico.

O caso de melhor speed-up foi o com o limiar de 50%, e esse foi o valor utilizado nos outros modelos. Este valor quer dizer que, no primeiro passo da escolha do melhor eixo para o posicionamento do plano

de partição, para os dois lados do plano estarem equilibrados, um lado não pode ter estimativa de carga menos da metade da estimativa do outro. Além disso, para um eixo ser escolhido neste primeiro passo, além de ter os lados equilibrados, a estimativa de carga da sua interface deve ser menos da metade da estimativa de carga da interface do outro eixo.

### 4.3.2 Detalhamento do tempo de execução

A Figura 4.6 mostra o detalhamento do tempo de execução, no processo mestre, de cada passo da implementação paralela para o melhor e o pior caso de *speed-up* da Viga. A Figura 4.7 mostra o mesmo detalhamento, porém em porcentagem, ou seja, com relação ao tempo de execução total.

Os passos são de Inicialização, Construção das estruturas de dados do Contorno, Geração e Classificação da *Octree* (Seções 3.2.1 e 3.3.1.1), Cálculo da estimativa de Carga (Seções 3.3.1.2 e 3.3.1.3), Geração da decomposição por BSP (Seção 3.3.2), Decomposição da Fronteira (Seção 3.3.3.1), Geração de Malha nos Subdomínios (Seção 3.3.3.2), Geração de Malha entre os Subdomínios (Seção 3.3.3.3), Geração da Malha Final, que é a aplicação do último avanço de fronteira por retrocesso na raiz da BSP, e *Overhead*, que é o tempo total menos o somatório dos tempos anteriores, e contabiliza tempos intermediários entre as chamadas de uma ou de outra função.

Em primeiro lugar, vê-se que, com dois processos, o procedimento paralelo ficou mais lento no segundo gráfico. Isso acontece por conta do maior tempo no cálculo da estimativa de carga e da decomposição do domínio.

Em segundo lugar, vê-se que o tempo de geração das malhas nos subdomínios é o tempo dominante, quando o número de processos é menor que 10 no primeiro gráfico, e no segundo gráfico todo. No primeiro gráfico, como o tempo de geração das malhas de interface aumenta muito com 12 ou mais processos, o *speed-up* acaba caindo consideravelmente.

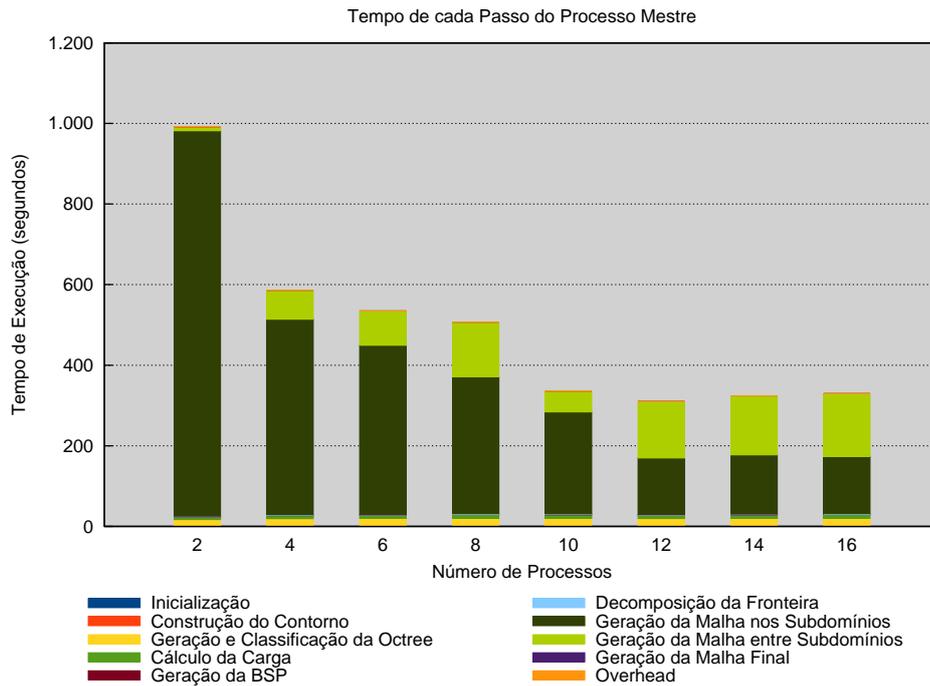
Em terceiro lugar, vê-se que, no segundo gráfico, o tempo de geração das malhas, tanto nos subdomínios quanto nas interfaces, diminui com relação ao primeiro gráfico. Em particular, nos casos com 12 ou mais processos, o tempo de geração das malhas de interface volta a ficar menor que o tempo de geração das malhas nos subdomínios. Isso acontece porque a decomposição é modificada, passando de uma decomposição que era nos eixos Y ou Z para uma decomposição no eixo X, diminuindo o tamanho da interface.

Finalmente, como consequência, quando o número de processos é maior que 2, o procedimento paralelo ficou mais rápido no segundo gráfico, como esperado.

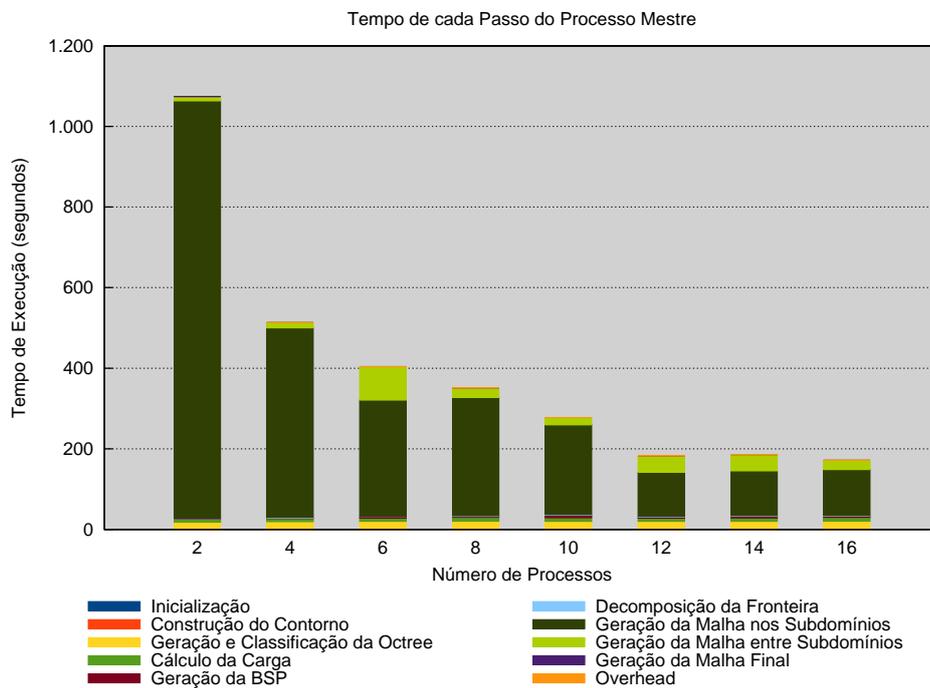
Observa-se, também, que os passos de gerar e classificar a *octree* e de calcular a estimativa de carga, apesar de terem um tempo independente do número de processos (Figura 4.6), proporcionalmente, este tempo cresce quando o número de processos cresce (Figura 4.7). Isso acontece porque estes passos são sequenciais, e chegam a atingir, no segundo gráfico, com 16 processos, aproximadamente 20% do tempo de execução total, mesmo em uma geometria comportada como a da Viga. Assim, paralelizar estes passos torna-se essencial para se melhorar os resultados de *speed-up*, especialmente em subdomínios mais complexos (como os que serão mostrados na Seção 4.4) e mais refinados, que gerariam malhas maiores.

### 4.3.3 Detalhamento do tempo de execução de geração de malha

Os gráficos da Figura 4.8 detalham os tempos de execução dos passos de Geração de Malha nos Subdomínios e Geração de Malha entre Subdomínios, para o processo mestre. A Figura 4.9 apresenta os mesmos gráficos, proporcionais à soma dos tempos de execução dos dois passos. As etapas de Inicialização, Gerência de Memória, Comunicação, Junção das Malhas e Finalização são incorporações ao código sequencial para paralelizá-lo. O código sequencial corresponde somente às etapas de Geração de Malha,



(a) Viga, 0 camada, limiar 0%.

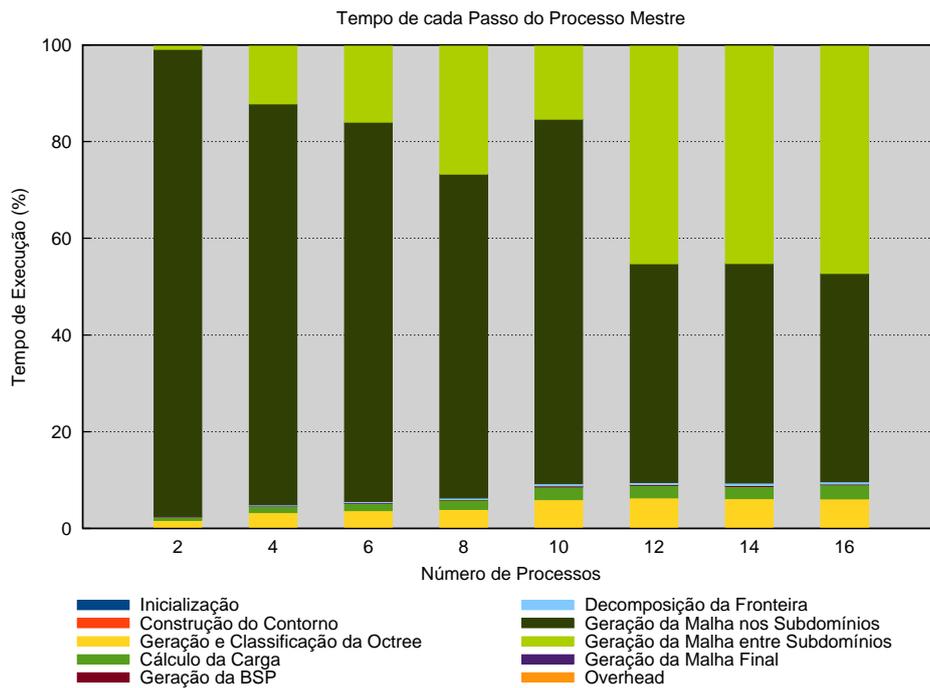


(b) Viga, 1 camada, limiar 50%.

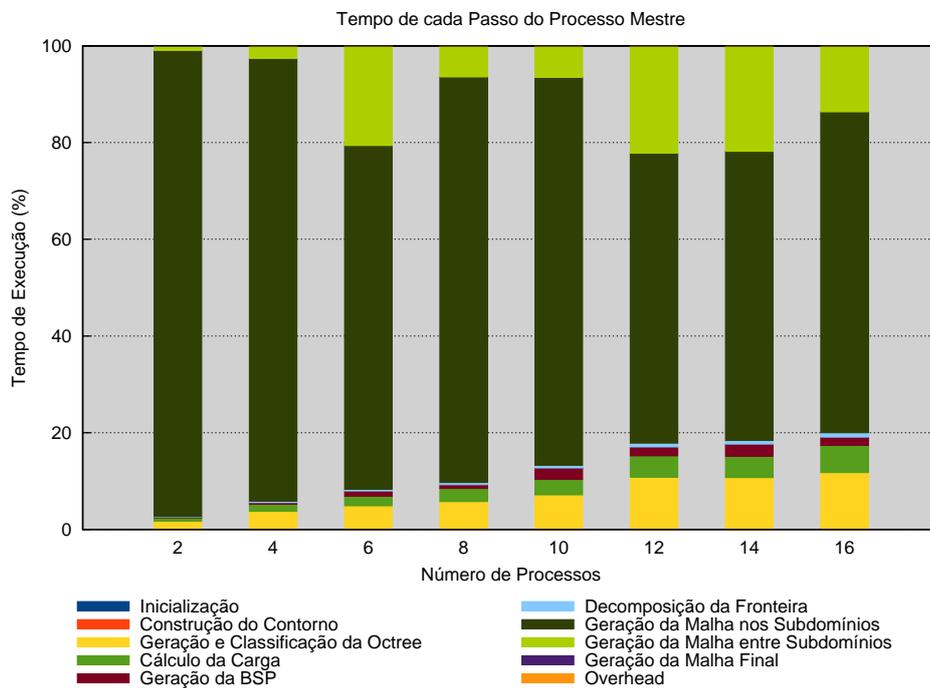
Figura 4.6 – Detalhamento do tempo de execução.

que são feitas com as modificações descritas na Seção 3.3.3.2 para viabilizar a geração da malha em um subdomínio.

Pode-se ver, nos gráficos, que um dos motivos de a geração das malhas de interface com 6, 12 e 14



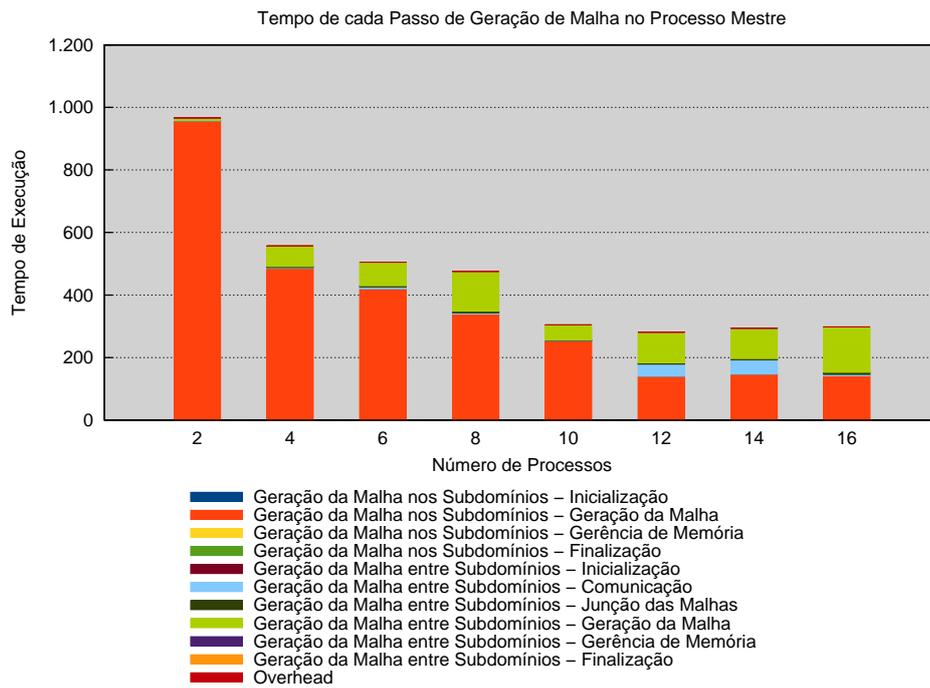
(a) Viga, 0 camada, limiar 0%.



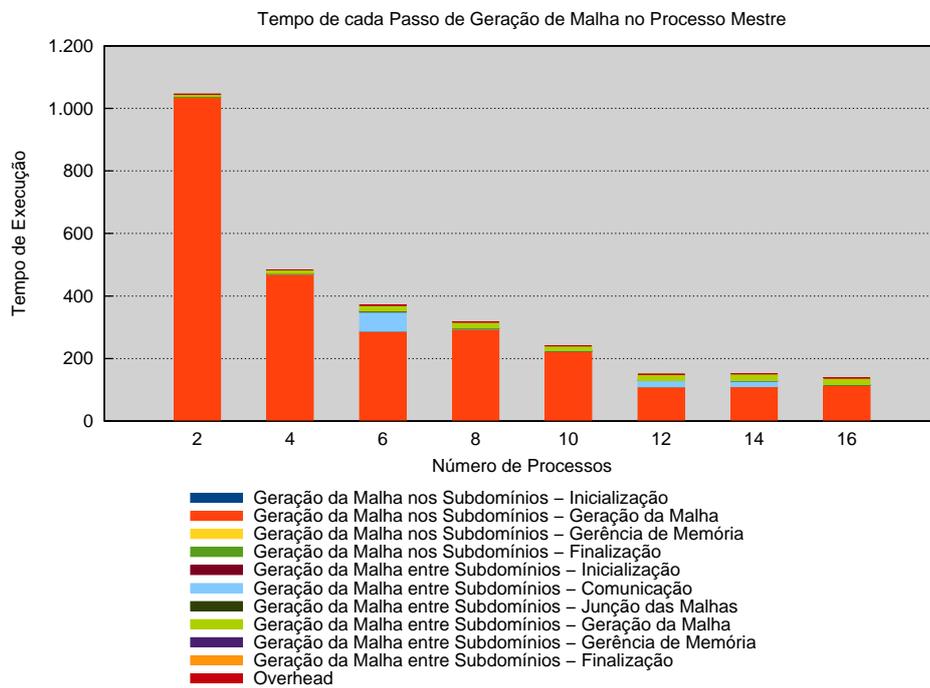
(b) Viga, 1 camada, limiar 50%.

Figura 4.7 – Detalhamento do tempo de execução em porcentagem.

processos ter demorado é o tempo de espera pela sincronização entre processos. Isso acontece devido à irregularidade na decomposição, ocasionada pelo fator de proporcionalidade das estimativas de carga no posicionamento do plano de partição. Apesar disso, sem esse fator de proporcionalidade, a espera pela



(a) Viga, 0 camada, limiar 0%.

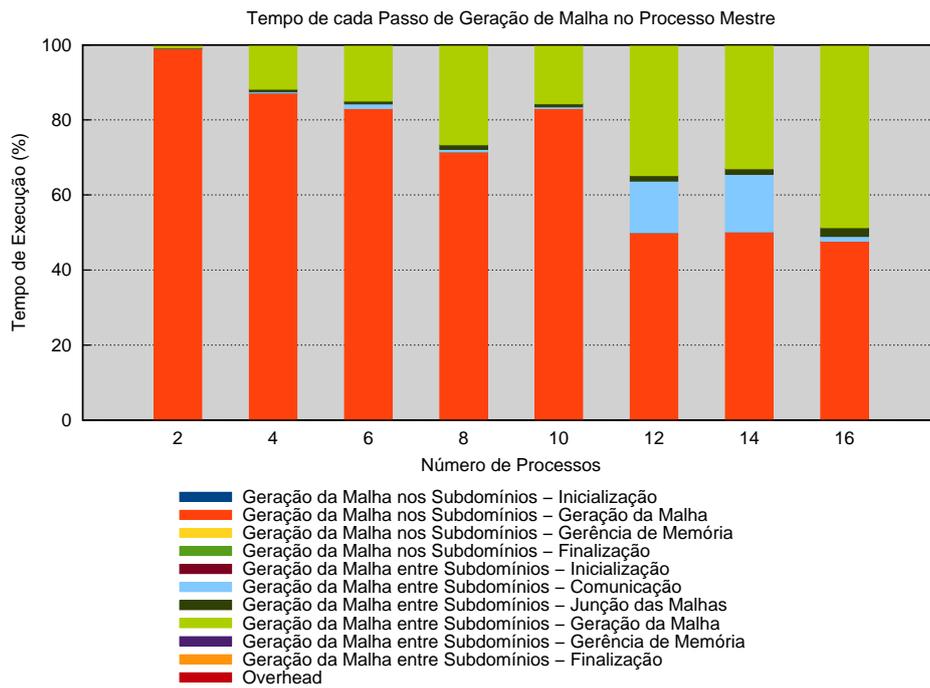


(b) Viga, 1 camada, limiar 50%.

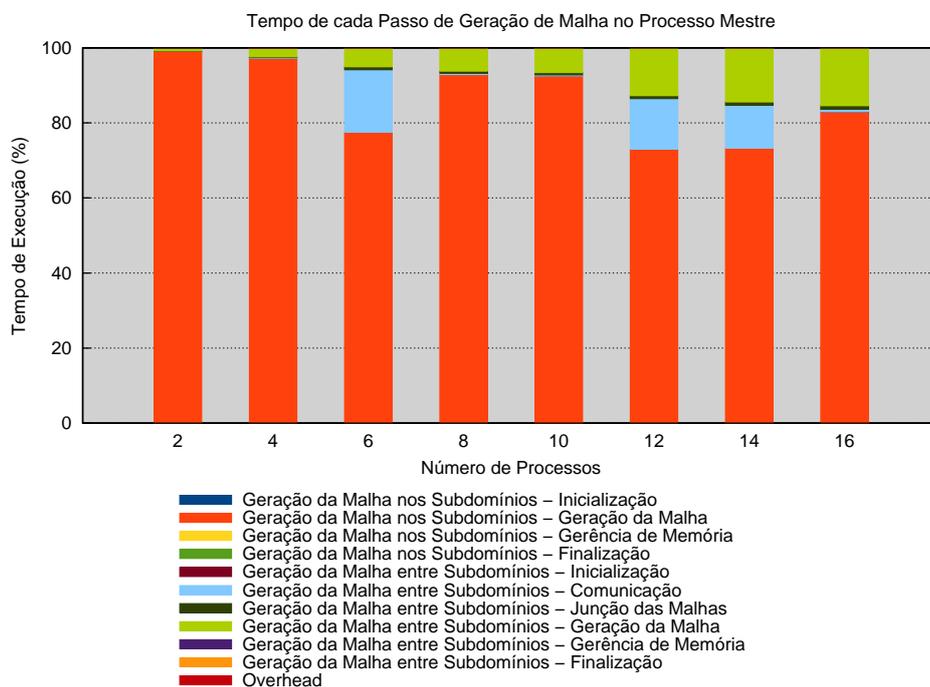
Figura 4.8 – Detalhamento do tempo de execução de geração de malha.

sincronização seria bem maior.

Vê-se, também, que o tempo dominante ainda é o de geração de malha, e não a comunicação ou os outros passos de junção de malhas ou de gerência de memória, por exemplo. Apesar disso, para um bom



(a) Viga, 0 camada, limiar 0%.



(b) Viga, 1 camada, limiar 50%.

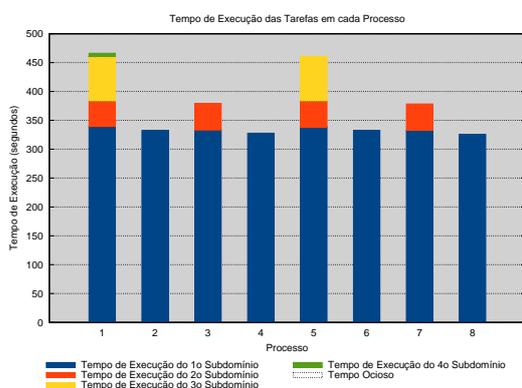
Figura 4.9 – Detalhamento do tempo de execução de geração de malha em porcentagem.

funcionamento da técnica, o ideal é que tempo de geração da malha nos nós internos da BSP, isto é, nas interfaces, seja menor que o tempo de geração da malha nos subdomínios. Isso implica que, de fato, as interfaces devem permanecer pequenas, mantendo os subdomínios equilibrados para não haver muita

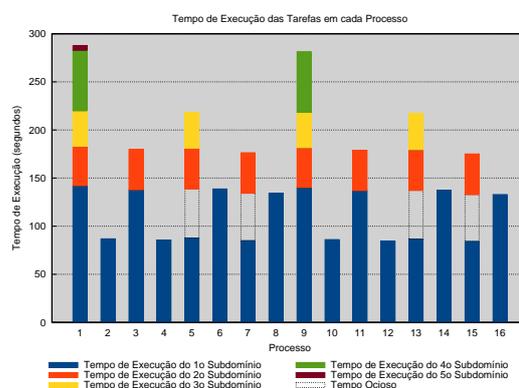
comunicação.

### 4.3.4 Balanceamento da carga

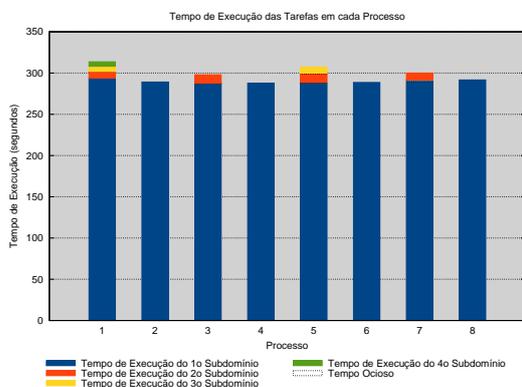
A Figura 4.10 mostra como a carga está balanceada entre os processos, ou seja, como o tempo de execução da geração da malha está distribuído entre os processos, comparando-se o pior caso e o melhor caso de *speed-up* da Viga, para 8 e para 16 processos. A barra de cor azul corresponde ao tempo de execução de geração de malha do subdomínio do processo indicado, ou seja, ao nó-folha da BSP. As barras de outras cores correspondem ao tempo de execução de geração das malhas de interface. O tempo de espera de um processo, ou seja, o tempo em que o processo está em espera, é indicado por uma barra transparente.



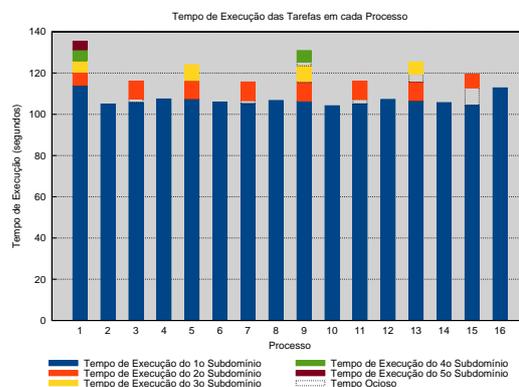
(a) Viga, 0 camada, limiar 0%, 8 processos.



(b) Viga, 0 camada, limiar 0%, 16 processos.



(c) Viga, 1 camada, limiar 50%, 8 processos.



(d) Viga, 1 camada, limiar 50%, 16 processos.

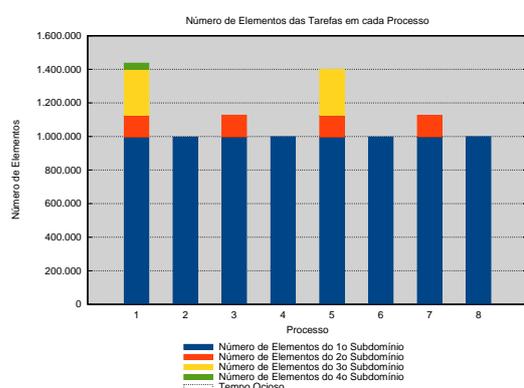
Figura 4.10 – Tempo de execução por processo.

Com 8 processos, no pior caso, a decomposição do domínio foi feita inicialmente no eixo X, depois no eixo Y e, finalmente, no eixo Z. Como pode-se ver na Figura 4.10a, todos os subdomínios levaram aproximadamente o mesmo tempo. Entretanto, nas interfaces, o tempo foi excessivamente grande. No melhor caso, a decomposição foi feita somente no eixo X e, portanto, a geração das malhas das interfaces levou bem menos tempo (Figura 4.10c). Perceba que o tempo de execução dos nós-folhas da BSP foi aproximadamente o mesmo que no pior caso.

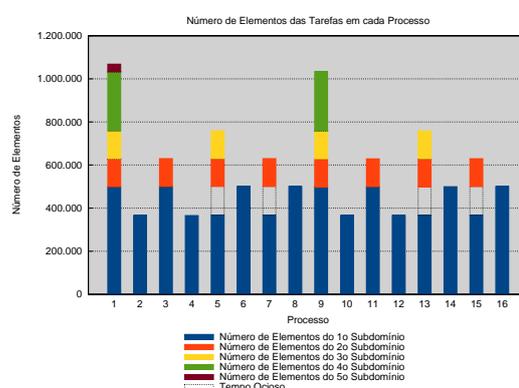
Com 16 processos, no pior caso, existe mais uma decomposição, feita no eixo Y, aumentando-se uma interface pesada (Figura 4.10b) enquanto que, no melhor caso, a decomposição é feita novamente no eixo X, aumentando-se uma interface leve (Figura 4.10d).

Observe, ainda na Figura 4.10b, que, apesar de as estimativas de cargas estarem equilibradas nos subdomínios, elas estão erradas, por que alguns subdomínios demoram bem mais que outros. Isso pode acontecer por dois motivos. Primeiro, a estimativa de carga tenta estimar o número de elementos, e não o tempo de execução. Assim, mesmo que dois subdomínios gerem a mesma quantidade de elementos, eles podem levar tempos bastante diferentes, dependendo da complexidade da geometria da fronteira em cada subdomínio. Portanto, é necessário analisar o número de elementos gerados em cada subdomínio.

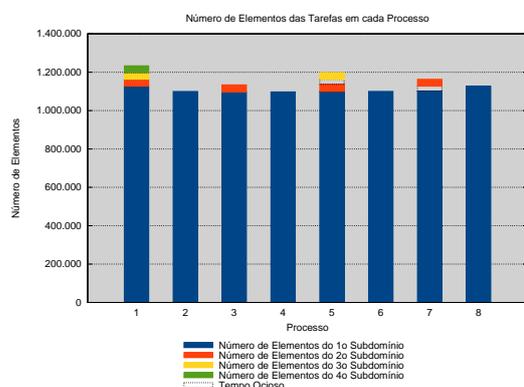
A Figura 4.11 mostra o balanceamento de carga para os mesmos casos anteriores, entretanto, levando em conta o número de elementos em vez do tempo de execução. Perceba que, na Figura 4.11b, o número de elementos continua desbalanceado, apesar de a estimativa de carga estar balanceada. Isso acontece porque, de fato, a geometria do modelo é simples e, portanto, a geometria nos subdomínios também é simples. Assim, o gráfico do tempo de execução não difere muito do gráfico do número de elementos.



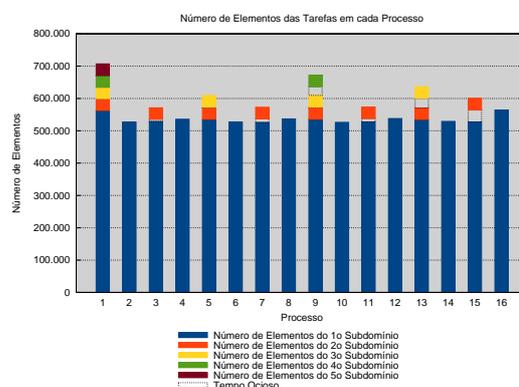
(a) Viga, 0 camada, limiar 0%, 8 processos.



(b) Viga, 0 camada, limiar 0%, 16 processos.



(c) Viga, 1 camada, limiar 50%, 8 processos.



(d) Viga, 1 camada, limiar 50%, 16 processos.

Figura 4.11 – Número de elementos por subdomínio por processo.

O segundo motivo para o desbalanceamento dos subdomínios é que a estimativa de carga pode estar errada, e isso será melhor explicado na próxima seção. Perceba, entretanto, que, na Figura 4.11d, isso foi corrigido e, portanto, o desbalanceamento não é tão grande.

### 4.3.5 Estimativa da carga

Antes de mostrar os resultados de estimativa de carga, é necessário fazer algumas observações. A estimativa da carga de um subdomínio, dada pela *octree* de estimativa de carga, é um valor adimensional, cujo valor absoluto, em si, não tem significado concreto mas que, quando comparado à estimativa de um

outro subdomínio, ganha significado. Por exemplo, uma estimativa de 200 não necessariamente quer dizer que 200 elementos serão gerados em um subdomínio mas, quando comparado à uma estimativa 100 de outro subdomínio, o valor de 200 quer dizer que este subdomínio gerará o dobro de elementos do outro subdomínio.

Como é somente esse valor relativo que é necessário para a estimativa da carga, é possível escaloná-lo, multiplicando-o por um valor qualquer, desde que esse valor seja o mesmo para todos os subdomínios. Como dito na Seção 3.3.3, existem duas fases diferentes de geração de malha, uma para os subdomínios e outra para as interfaces. Assim, dois escalonamentos são feitos, um para os subdomínios e outro para as interfaces.

O cálculo desse escalonamento é feito de acordo com a Equação 4.1, onde  $N$  é o número de subdomínios,  $c_i$  é a estimativa de carga do subdomínio  $i$ ,  $e_i$  é o número de elementos gerados no subdomínio  $i$ ,  $\bar{c}$  e  $\bar{e}$  são os valores médios de estimativa de carga e de número de elementos, respectivamente, e  $e_i^{est}$  é a estimativa do número de elementos do subdomínio  $i$ . A escala é dada por  $\bar{e}/\bar{c}$ . Perceba que o cálculo da estimativa do número de elementos somente pode ser feito após a geração da malha, pois é necessário saber o número de elementos gerados:

$$\begin{aligned}\bar{c} &= \frac{\sum_{i=1}^N c_i}{N}, \\ \bar{e} &= \frac{\sum_{i=1}^N e_i}{N}, \\ e_i^{est} &= c_i \frac{\bar{e}}{\bar{c}} = c_i \frac{\left(\sum_{i=1}^N e_i\right) / N}{\left(\sum_{i=1}^N c_i\right) / N} = c_i \frac{\sum_{i=1}^N e_i}{\sum_{i=1}^N c_i}.\end{aligned}\quad (4.1)$$

Uma equação análoga à Equação 4.1 é utilizada para as interfaces, em vez dos subdomínios. Trocando-se o número de elementos pelo número de vértices ou pelo tempo de execução, é possível relacionar a estimativa de carga com essas medidas. Entretanto, como dito na Seção 3.3.1, deseja-se estimar o número de elementos neste trabalho.

A Figura 4.12 mostra a estimativa do número de elementos nos mesmos 4 casos apresentados anteriormente. As barras representam o número de elementos gerados em cada tarefa, seja ela de geração de malhas nos subdomínios (as 8 ou 16 primeiras barras) ou nas interfaces (o restante das barras). Nas Figuras 4.12a e 4.12b, não há estimativa de número de elementos nas interfaces.

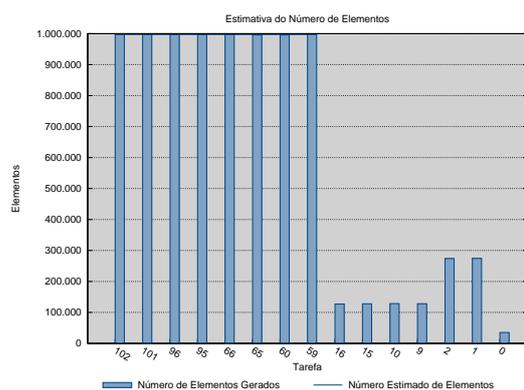
A Figura 4.12b indica que, apesar de todos os subdomínios terem tido a mesma estimativa de número de elementos, em razão de terem tido a mesma estimativa de carga, o número de elementos gerados em cada subdomínio variou. A Figura 4.12d mostra que a estimativa foi bem mais precisa nesse caso.

Isso acontece porque existe uma região vazia entre a fronteira final e a borda do subdomínio onde elementos não são gerados. Portanto, é necessário remover essa região da estimativa de carga, como dito na Seção 3.3.1.2, e isso é feito removendo-se 1 camada de células da *octree* adjacentes ao plano de partição. Com isso, a estimativa de carga fica mais precisa, levando a uma melhor decomposição do domínio.

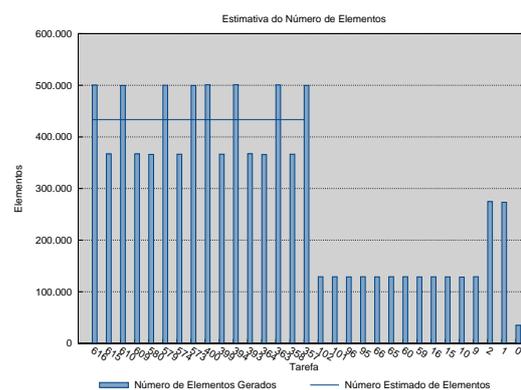
Essa precisão da estimativa do número de elementos pode ser melhor analisada através do erro da estimativa do número de elementos. Esse erro é o erro relativo, em porcentagem, entre o valor estimado e o valor medido, de acordo com a Equação 4.2, ou seja, entre a estimativa do número de elementos para um subdomínio e o número de elementos gerados neste subdomínio:

$$err_i = 100 \frac{c_i^{est} - e_i}{e_i^{est}}. \quad (4.2)$$

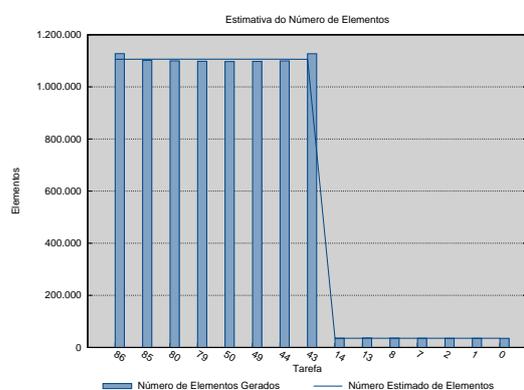
O erro relativo é calculado com relação ao valor estimado por que é esse o valor que se tem a princípio, antes de se gerar a malha. Um erro positivo indica que o número de elementos foi superestimado, um erro negativo indica que o número de elementos foi subestimado, e um erro 0,0 indica que o número de



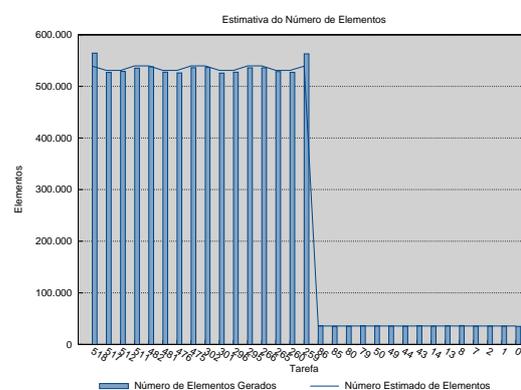
(a) Viga, 0 camada, limiar 0%, 8 processos.



(b) Viga, 0 camada, limiar 0%, 16 processos.



(c) Viga, 1 camada, limiar 50%, 8 processos.



(d) Viga, 1 camada, limiar 50%, 16 processos.

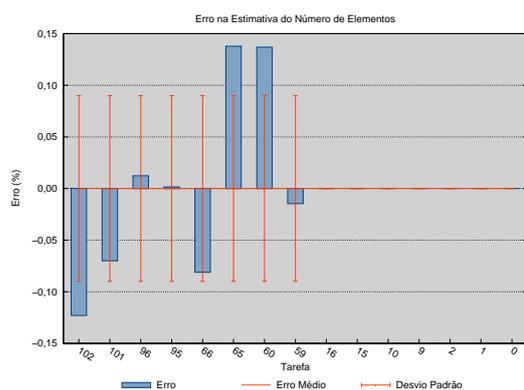
Figura 4.12 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces.

elementos gerados foi o mesmo que o estimado. O mesmo raciocínio pode ser aplicado às interfaces, em vez dos subdomínios

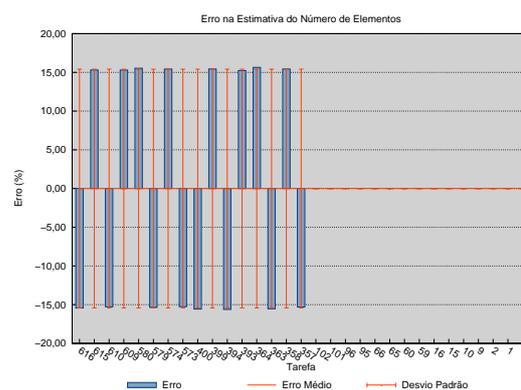
Os gráficos da Figura 4.13 mostram o erro na estimativa do número de elementos em cada tarefa, seja ela de geração de malha nos subdomínios ou nas interfaces. Além dos erros, representados pelas barras, os gráficos mostram também o erro médio e o desvio padrão. O desvio padrão indica o quão longe da média estão os valores individuais dos erros. Assim, quanto menor o desvio padrão, mais próximos da média estão os valores individuais. Dado um desvio padrão pequeno, quanto menor a média, mais próximo de 0,0 é o erro relativo, indicando que a estimativa do número de elementos de fato se aproxima ao número de elementos gerados. Portanto, deseja-se que tanto a média dos erros quanto o desvio padrão sejam próximos de 0,0.

Perceba, na Figura 4.13a, que o maior erro, em valor absoluto, não chega a 0,15%. Entretanto, na Figura 4.13b, o pior caso chega a mais de 15%. Com a remoção da camada de células, o erro aumenta na Figura 4.13c, porém não chega a 3%. Já na Figura 4.13d, vê-se que o erro cai para menos de 5%, indicando que, com a remoção da camada de células na estimativa de carga, a estimativa do número de elementos torna-se precisa, para o modelo da Viga.

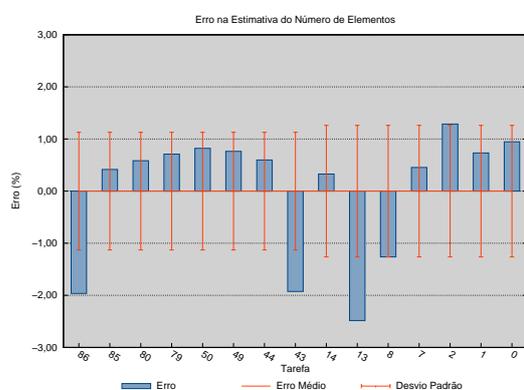
Essa afirmação pode ser melhor comprovada através da Figura 4.14, que mostra tanto o erro quanto o desvio padrão da estimativa do número de elementos para os subdomínios em todos os testes rodados (Figura 4.14a) e para os subdomínios e para as interfaces no melhor caso do *speed-up* (Figura 4.14b). Observe que, antes da remoção da camada de células na estimativa de carga, o erro chegou a mais de 15%,



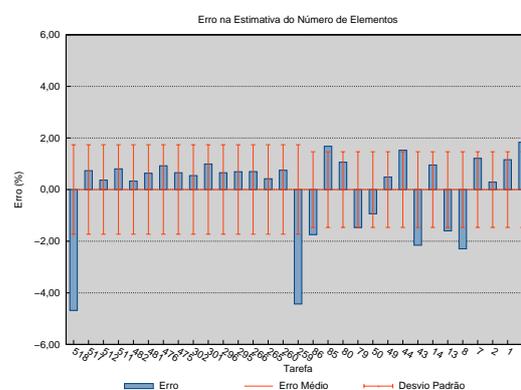
(a) Viga, 0 camada, limiar 0%, 8 processos.



(b) Viga, 0 camada, limiar 0%, 16 processos.



(c) Viga, 1 camada, limiar 50%, 8 processos.



(d) Viga, 1 camada, limiar 50%, 16 processos.

Figura 4.13 – Erro na estimativa do número de elementos em cada tarefa, subdomínios ou interfaces.

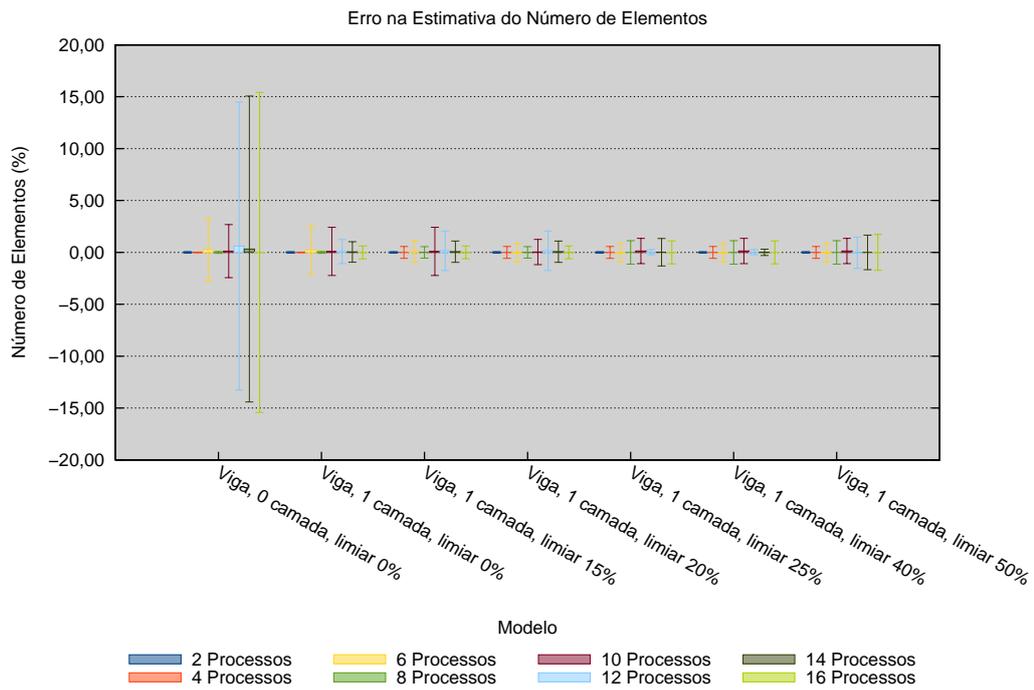
com 12 ou mais processos. Após a remoção dessas camadas, o erro médio e o desvio padrão passaram a ter um comportamento mais constante, próximos a 3% no pior caso. Na Figura 4.14b, pode-se ver que tanto os subdomínios quanto as interfaces tiveram uma estimativa de carga precisa, com erro médio pequeno e desvio padrão abaixo de 2%.

Percebe-se também, na Figura 4.14b, que o desvio padrão cresce quando o número de subdomínios aumenta. Isso indica que a *octree* é boa para estimar a carga, mas que, quanto menor o subdomínio, ou seja, quanto menor o número de elementos em um subdomínio, menos precisa é a estimativa de carga. Isso já era visível, inclusive, no primeiro caso, quando a camada de células da *octree* não foi removida, e que a sua presença na estimativa de carga de subdomínios pequenos, nos casos com 12 ou mais processos, é bastante prejudicial.

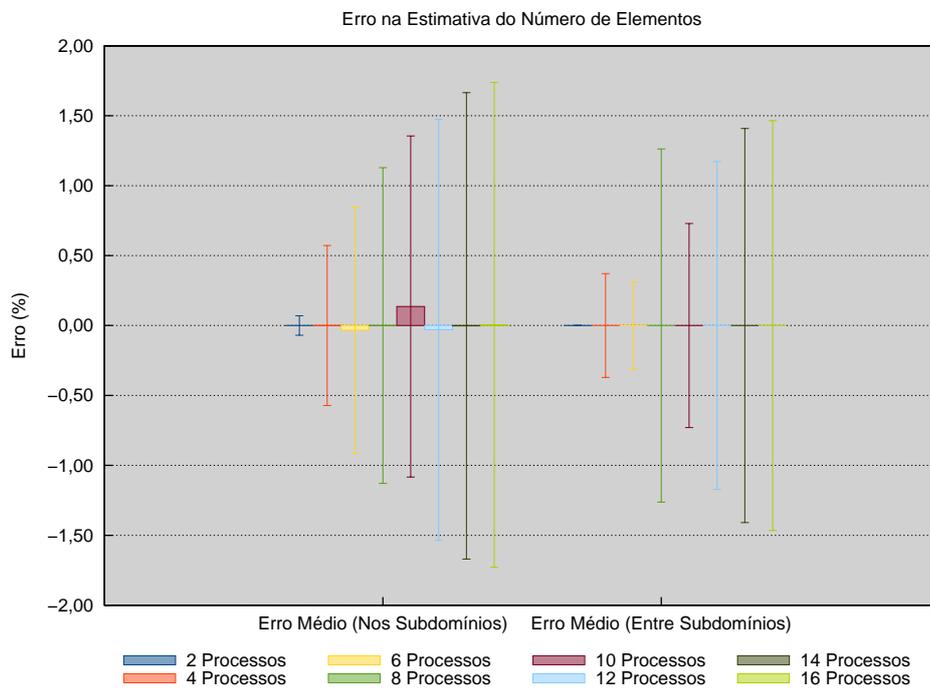
### 4.3.6 Qualidade da malha

O último tópico analisado é a qualidade da malha gerada. O propósito principal da técnica paralela não é, necessariamente, gerar uma malha de boa qualidade, mas gerar uma malha de qualidade próxima à qualidade da malha gerada sequencialmente. Assim, pode-se dizer que a paralelização da técnica sequencial não acarretou em uma piora significativa na qualidade da malha.

A qualidade de uma malha é dada pela qualidade dos seus elementos, e a métrica utilizada neste trabalho para o cálculo da qualidade de um tetraedro é definida como  $\alpha = 3R_i/R_c$ , onde  $R_i$  e  $R_c$  são os raios das esferas inscrita e circunscrita, respectivamente (LEWIS et al., 1996; PARTHASARATHY et al.,



(a) Somente os subdomínios, em todas as execuções.

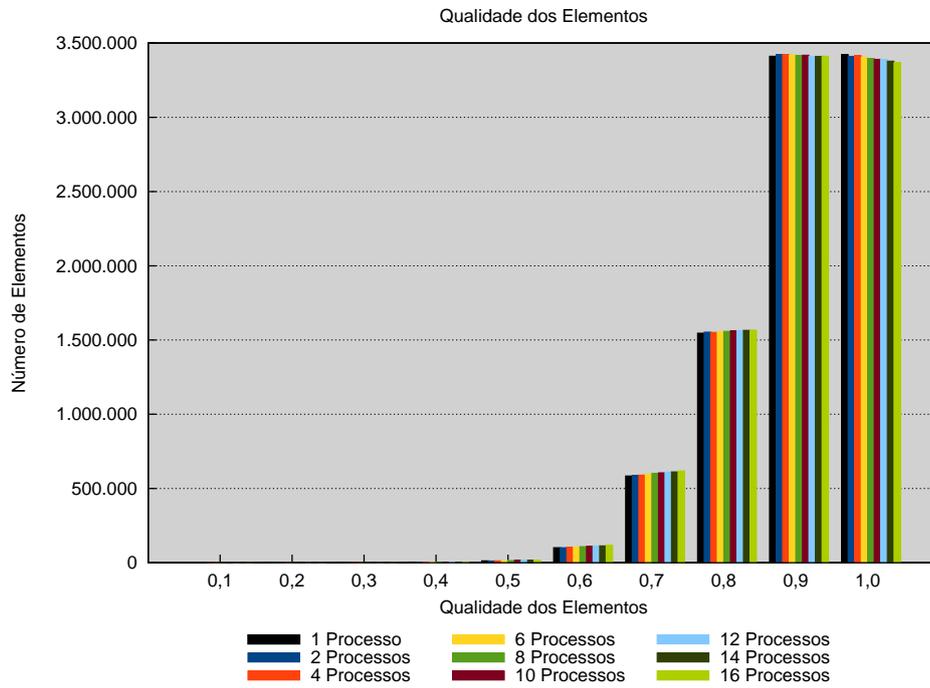


(b) 1 camada, limiar 50%.

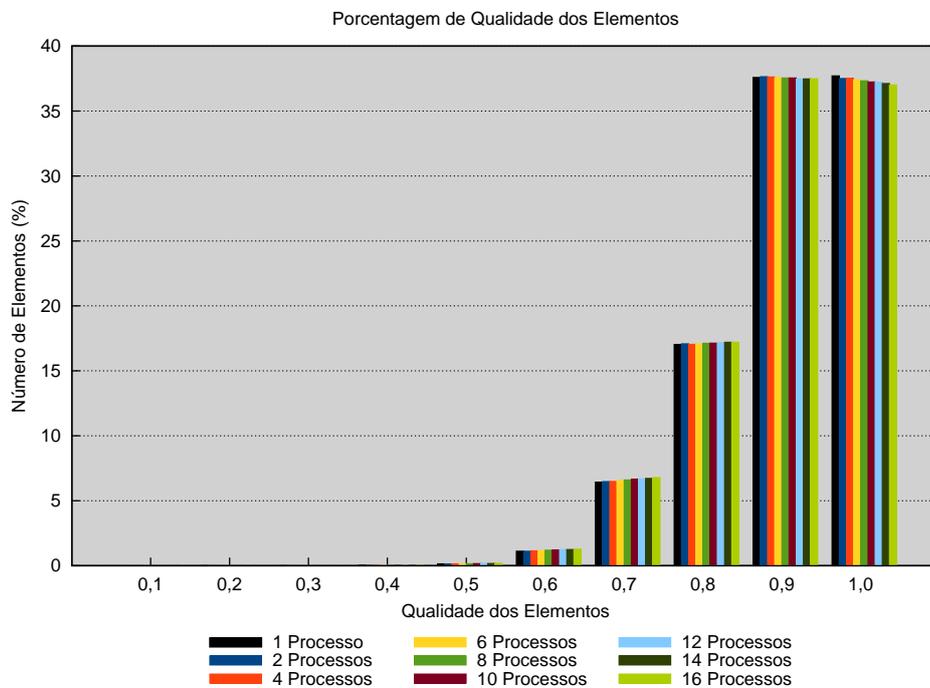
Figura 4.14 – Erro médio e desvio padrão da estimativa do número de elementos para o modelo da Viga.

1993). Esta métrica  $\alpha$  tem valor 1,0 para um tetraedro equilátero. Quanto pior a qualidade do elemento, mais próximo de 0,0 é o valor de  $\alpha$ . Pode-se dizer que os elementos com  $\alpha \leq 0,1$  são de péssima qualidade e que os elementos com  $\alpha \geq 0,7$  são de boa qualidade.

A Figura 4.15a mostra a qualidade das malhas geradas para o melhor caso do *speed-up*, e a Figura 4.15b mostra os mesmos dados, porém com relação ao tamanho da malha gerada.



(a) Qualidade das malhas geradas.



(b) Qualidade, em porcentagem, das malhas geradas.

Figura 4.15 – Viga, 1 camada, limiar 50%.

Nos gráficos da Figura 4.15, cada barra indica a quantidade de elementos em cada intervalo de

qualidade, para cada malha gerada, variando-se o número de processos. Pode-se ver que as malhas geradas em paralelo seguem a mesma tendência de qualidade da malha gerada sequencialmente. Como esta malha foi de boa qualidade, ou seja, a grande maioria dos elementos têm qualidade 0,7 ou mais, as malhas geradas em paralelo também têm boa qualidade.

Para melhor comparar as malhas geradas em paralelo com a malha gerada sequencialmente, calcula-se a diferença das qualidades como sendo a diferença, em porcentagem, entre a quantidade de elementos em um determinado intervalo de uma malha gerada em paralelo e a quantidade de elementos no mesmo intervalo da malha gerada sequencialmente. A Figura 4.16 mostra essa diferença, em cada intervalo. Quando a diferença é positiva, a malha gerada em paralelo tem mais elementos nesse intervalo, quando a diferença é negativa, a malha gerada sequencialmente tem mais elementos nesse intervalo.

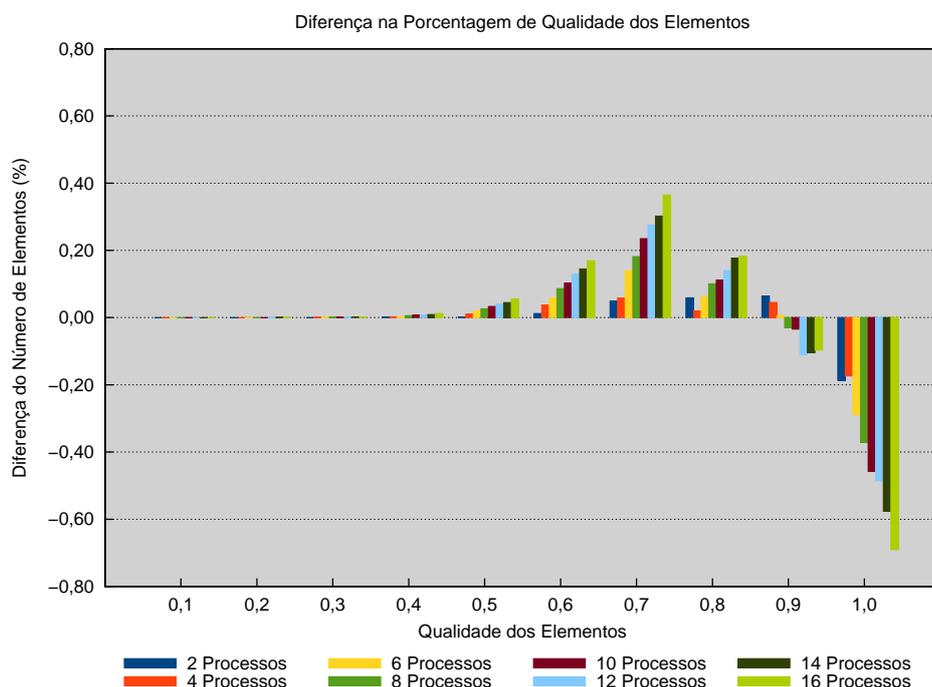


Figura 4.16 – Diferença entre a porcentagem da qualidade das malhas geradas em paralelo e a da malha gerada sequencialmente, para o modelo da Viga, 1 camada, limiar 50%.

Na Figura 4.16, vê-se claramente que alguns dos elementos que tinham qualidade  $0,9 \leq \alpha \leq 1,0$  passaram a ter qualidade  $\alpha < 0,9$ . A qualidade de grande parte desses elementos, entretanto, permaneceu  $\alpha \geq 0,5$ , o que indica que alguns elementos deixaram de ser muito bons e passaram a ser medianos. A quantidade de elementos de qualidade ruim, com  $\alpha < 0,1$ , manteve-se praticamente constante. Além disso, percebe-se que essa diferença é de menos de 1% dos elementos, em cada intervalo.

A soma dos valores absolutos dessas diferenças pode ser vista uma medida da diferença, como um todo, entre malha gerada em paralelo e a malha gerada sequencialmente. A Figura 4.17 mostra as somas dessas diferenças para todas as execuções mencionadas anteriormente. Perceba que, quando remove-se a camada de células da *octree* na estimativa da carga e conforme aumenta-se o limiar, melhora-se o posicionamento do plano de partição e, portanto, a decomposição do domínio. Isso é refletido na qualidade da malha, que fica cada vez mais próxima da malha gerada sequencialmente.

Além disso, é possível também ver que, conforme aumenta-se o número de processos, ou seja, quanto mais se decompõe o domínio, maior é a diferença da qualidade da malha gerada em paralelo para a malha gerada sequencialmente. Logo, existe um *trade-off* entre qualidade da malha e rapidez da geração, ou

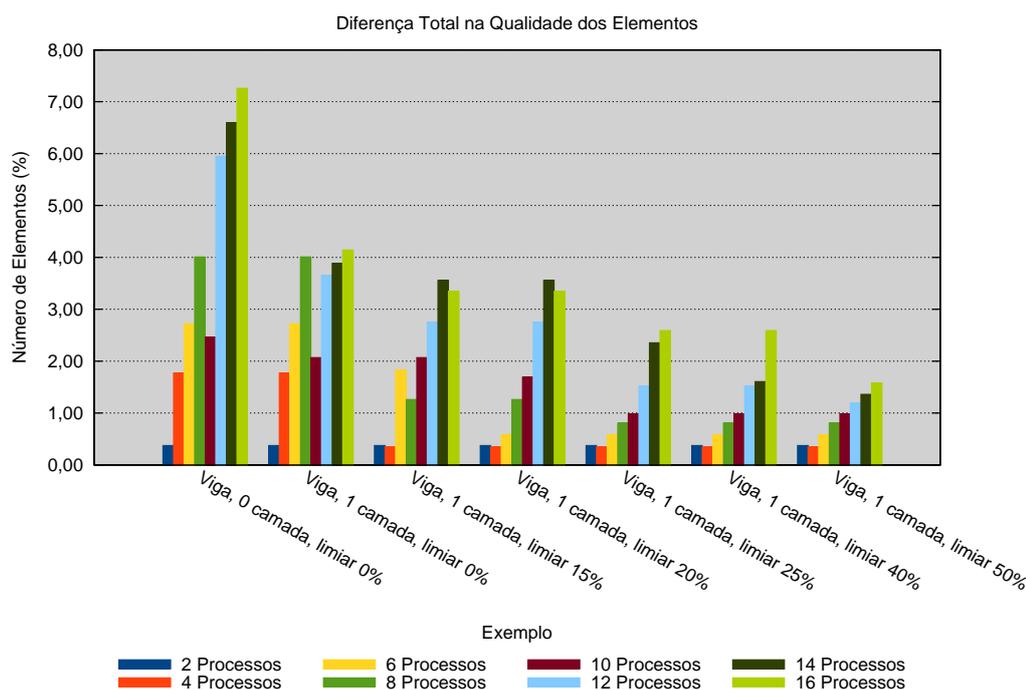


Figura 4.17 – Diferença total na qualidade das malhas geradas, para o modelo da Viga.

seja, quanto mais rápido deseja-se gerar a malha, utilizando-se mais processos, menor vai ser a qualidade da malha, por conta da decomposição feita no domínio. Entretanto, essa diferença, no pior caso do modelo da Viga, corresponde a menos de 8% dos elementos gerados. No geral, a diferença é de menos de 4% dos elementos e, no melhor caso, de menos de 2% dos elementos gerados.

### 4.3.7 Malha

A Figura 4.18 mostra a vista frontal da borda da malha gerada em paralelo para o modelo da Viga, com 16 processos, em que cada cor corresponde a um processo, quando 1 camada de células da *octree* foi removida no cálculo da estimativa de carga e com um limiar de 50%. Perceba que este modelo, nessas condições, segue exatamente o que foi ilustrado na Figura 3.29, com o processo 0, de cor vermelha, gerando a parte da malha correspondente ao seu subdomínio e a 4 interfaces.



Figura 4.18 – Malha para o modelo da Viga.

## 4.4 Modelos práticos

Esta seção apresenta os resultados obtidos para os modelos práticos, ou seja, a Engrenagem com fratura, o Rotor e a Engrenagem sem fratura. Esses modelos são ilustrados, respectivamente, nas Figuras 4.19 a 4.21.

A Figura 4.22 mostra as decomposições dos modelos práticos, com 16 processos. Pelas Figuras 4.22a e 4.22b, pode-se ver que as regiões próximas da fratura são as que contém mais subdomínios. Isso acontece

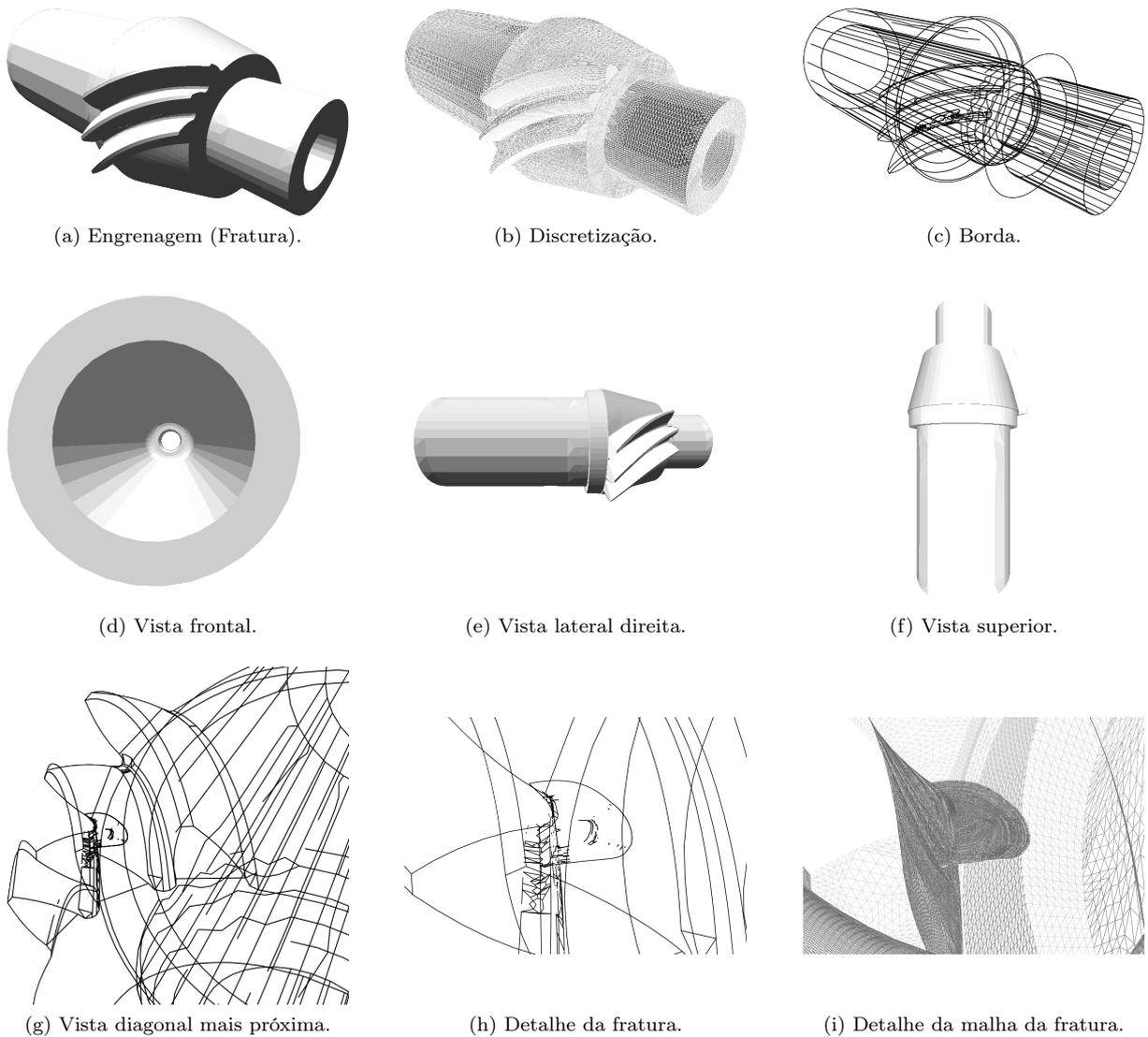


Figura 4.19 – Modelo da Engrenagem (Fratura).

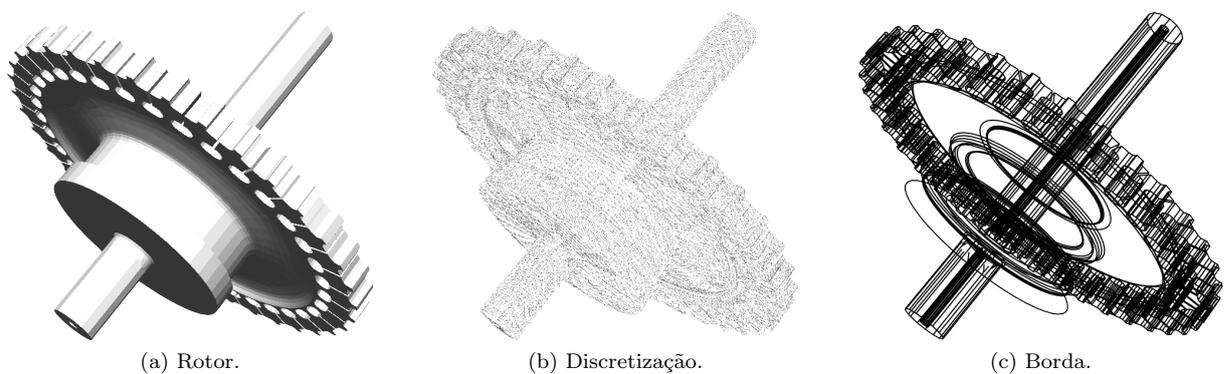


Figura 4.20 – Modelo do Rotor.

porque essas regiões são mais refinadas e, conseqüentemente, é nessas regiões onde mais elementos serão gerados. Assim, pela maneira como a estimativa de carga é feita, essas regiões são mais pesadas e, portanto, mais subdomínios serão gerados nas suas proximidades.

A Figura 4.23 mostra como a fratura é decomposta no modelo da Engrenagem (Fratura).

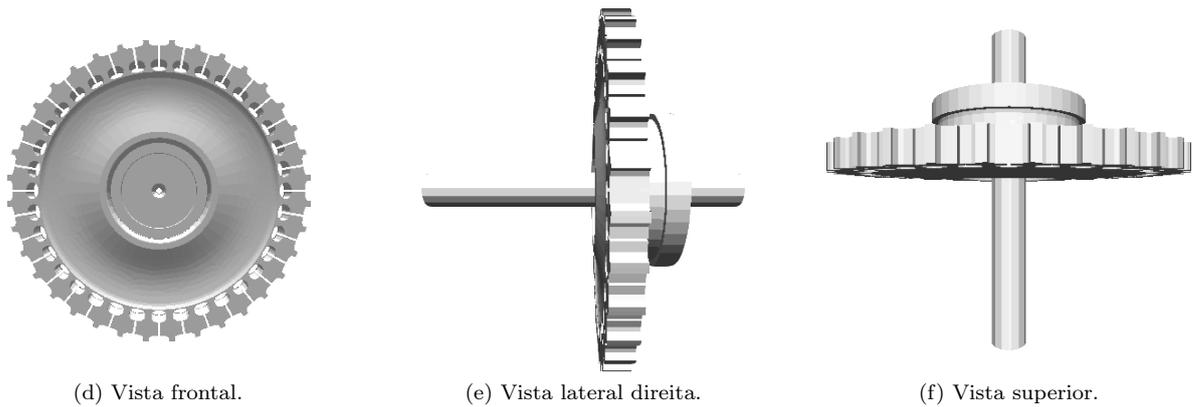


Figura 4.20 – Modelo do Rotor (continuação).

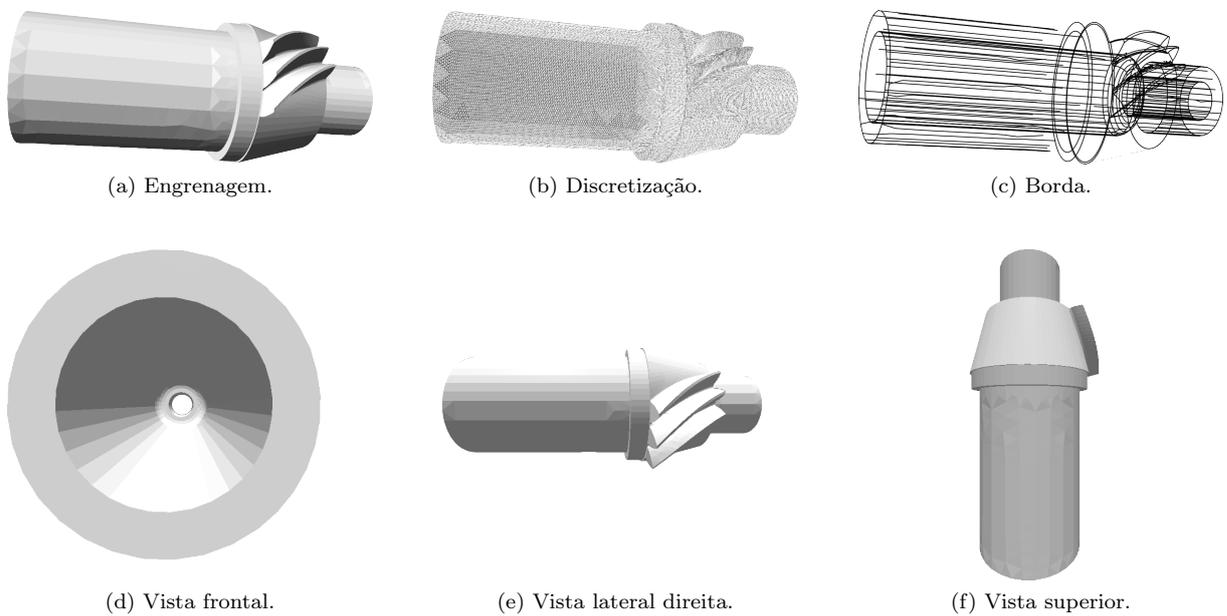


Figura 4.21 – Modelo da Engrenagem.

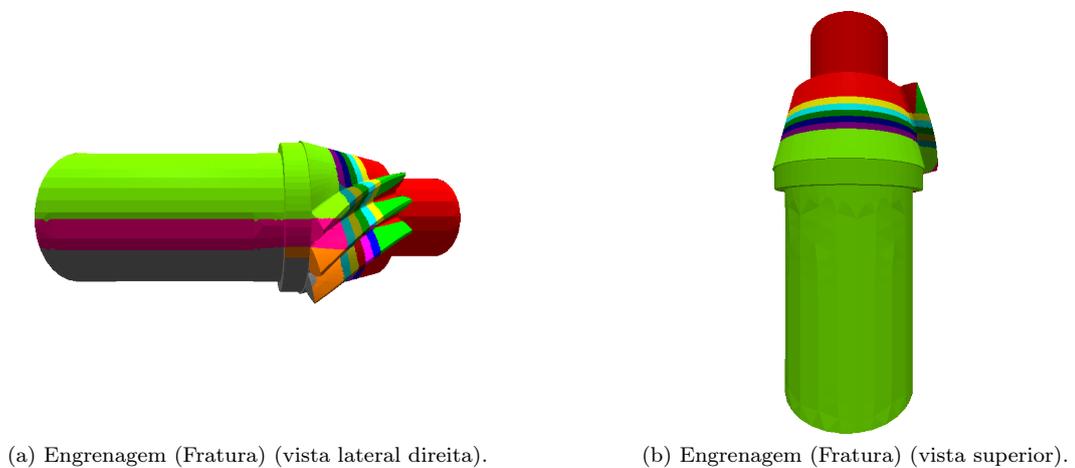


Figura 4.22 – Decomposições dos modelos práticos com 16 processos.

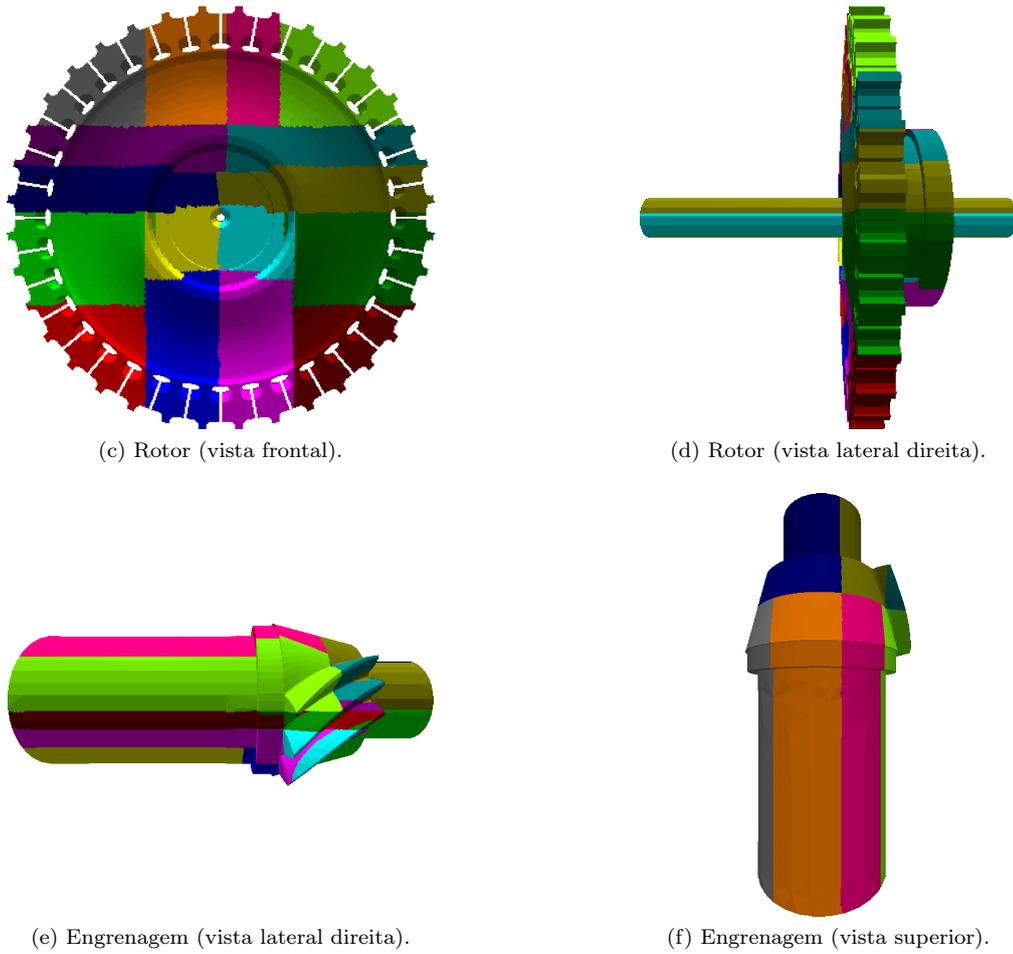


Figura 4.22 – Decomposições dos modelos práticos com 16 processos (continuação).

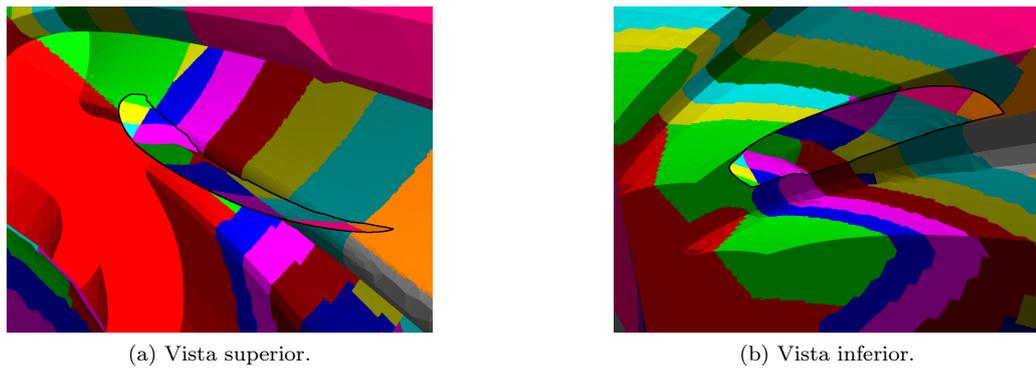


Figura 4.23 – Decomposição da fratura do modelo da Engrenagem (Fratura) com 16 processos.

A Figura 4.24 mostra o tamanho das malhas geradas pela técnica sequencial. Comparando o modelo da Engrenagem (Fratura) com o da Engrenagem, como o primeiro tem fratura, a malha desse modelo deveria ser maior. Entretanto, além de esse modelo ter sido gerado com o refinamento uniforme da *octree* desativado, o segundo modelo tem uma borda mais discretizada, gerando, assim, uma malha maior, com pouco mais de 4,5 milhões de elementos.

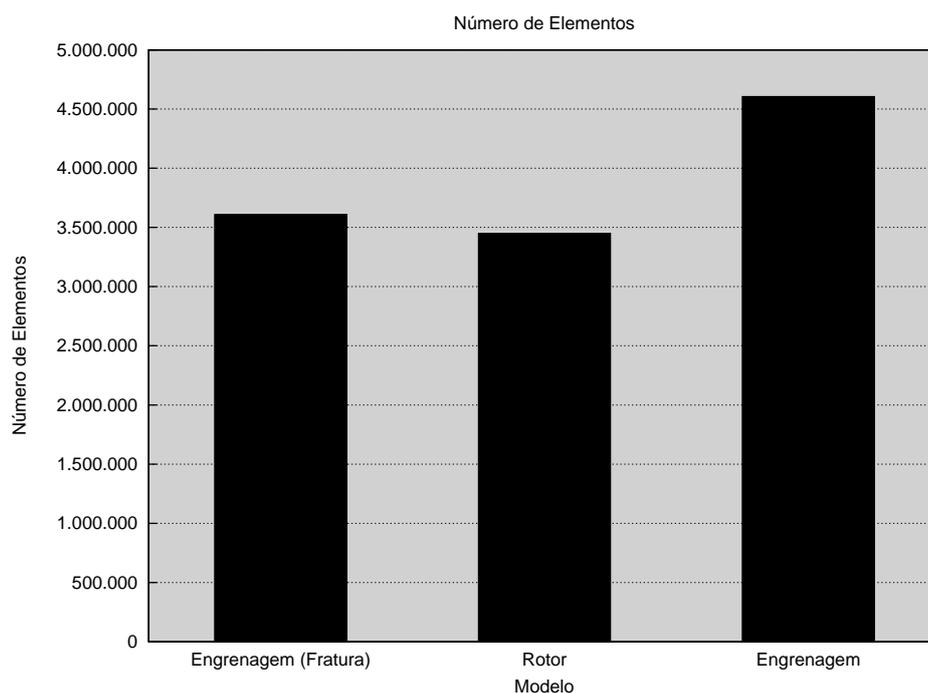


Figura 4.24 – Tamanhos das malhas dos modelos práticos.

#### 4.4.1 Tempo de execução e *speed-up*

A Figura 4.25 mostra os gráficos de tempo de execução e de *speed-up* para os modelos práticos. Para efeito de comparação, o modelo da Viga, com sua melhor configuração, também foi incluído nos dois gráficos. O gráfico da Figura 4.25a mostra que o tempo de execução está, obviamente, relacionado com o tamanho da malha gerada.

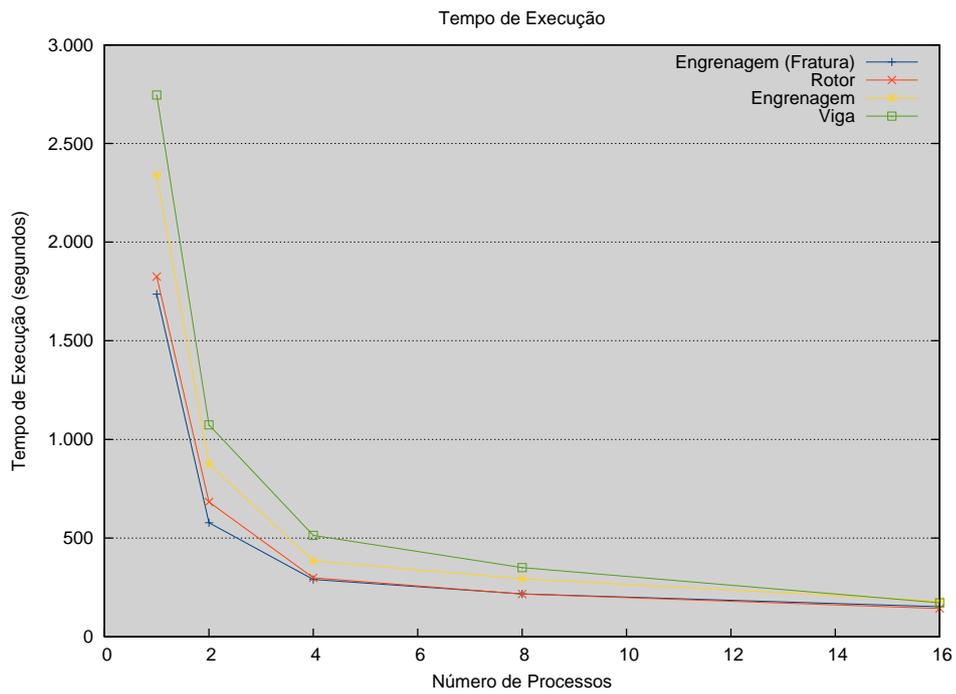
O modelo da Viga é o que gera a maior malha, com cerca de 9 milhões de elementos e, portanto, é nele que a implementação sequencial demora mais. Por não ter geometria complexa, entretanto, a diferença no tempo sequencial para os outros modelos não é tão grande.

As malhas dos modelos da Engrenagem (Fratura) e do Rotor têm, aproximadamente, o mesmo tamanho e, assim, a implementação sequencial obteve aproximadamente o mesmo tempo de execução. O modelo do Rotor, por ter uma geometria mais complexa, demorou um pouco mais que o da Engrenagem (Fratura). O modelo prático que gerou a maior malha, o da Engrenagem, foi também o que levou mais tempo para terminar de gerar a malha, sequencialmente.

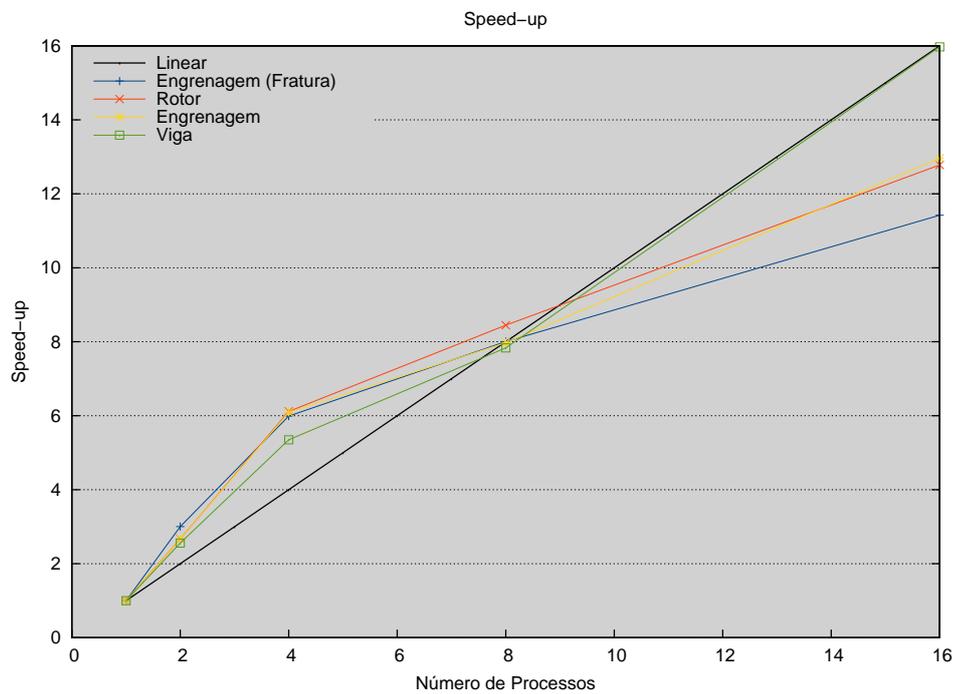
Pelo gráfico de *speed-up* da Figura 4.25b, é possível observar dois padrões. Em primeiro lugar, há uma super-linearidade com poucos processos, o que pode indicar que a implementação sequencial pode necessitar de otimizações. Em segundo lugar, mesmo no pior caso, o *speed-up* foi sempre crescente, ou seja, sempre que o número de processos aumentou, o tempo de execução diminuiu.

Os modelos da Engrenagem (Fratura) (Figuras 4.19, 4.22a e 4.22b) e da Engrenagem (Figuras 4.21, 4.22e e 4.22f) apresentam geometrias bem parecidas e, portanto, gráficos de *speed-up* bem parecidos. Além disso, a geometria de ambos lembra a do modelo da Viga, ou seja, eles são curtos em dois eixos e compridos no outro eixo, favorecendo a técnica de decomposição. Por último, por gerar uma malha maior, o modelo da Engrenagem tem um *speed-up* superior ao da Engrenagem (Fratura).

Grande parte da geometria do modelo do Rotor (Figuras 4.20, 4.22c e 4.22d) é fina no eixo Z e comprida nos eixos X e Y, implicando, intuitivamente, em planos de decomposição posicionados somente nesses dois



(a) Tempo de execução.



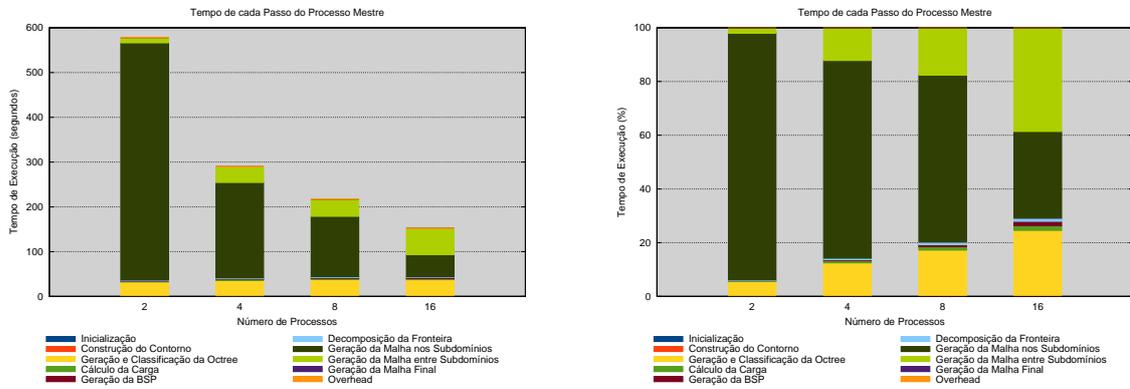
(b) Speed-up.

Figura 4.25 – Tempo de execução e *speed-up* dos modelos práticos.

últimos eixos. Isso é exatamente o que acontece, ocasionando um bom desempenho da implementação paralela.

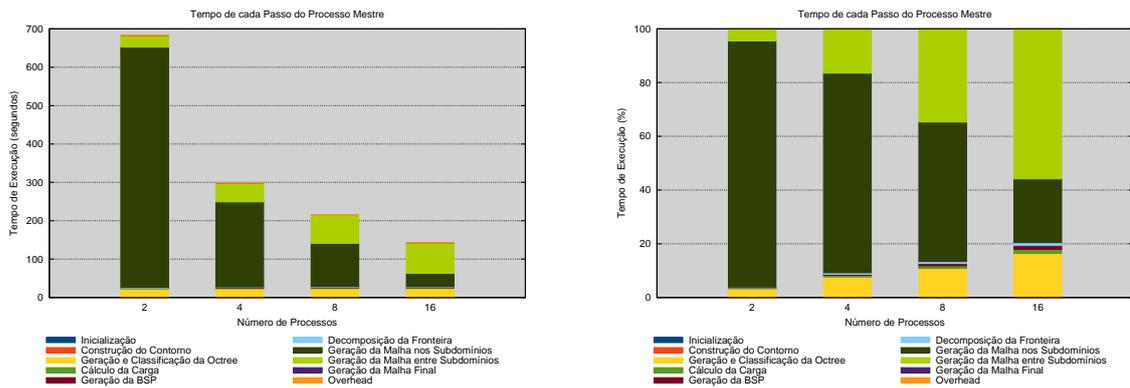
### 4.4.2 Detalhamento do tempo de execução

As duas colunas da Figura 4.26 mostram o detalhamento do tempo de execução do processo mestre, em termos absolutos e relativos, respectivamente, para os modelos práticos apresentados nesta seção.



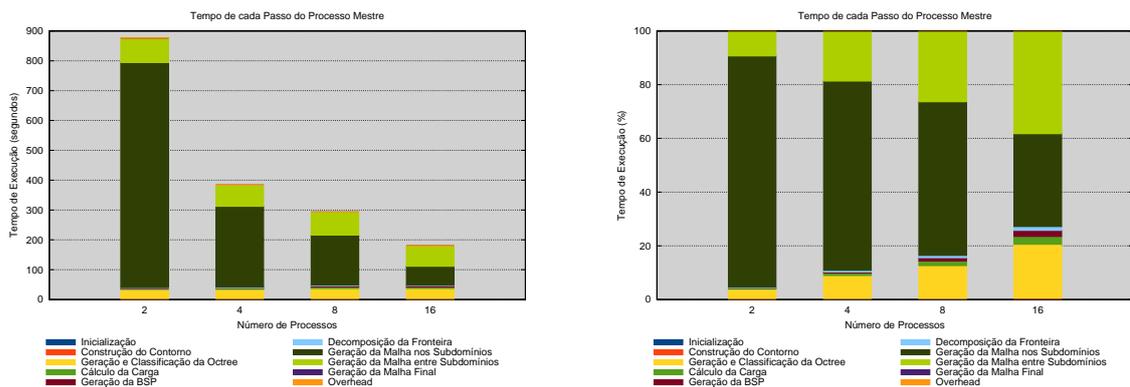
(a) Tempo absoluto da Engrenagem (Fratura).

(b) Tempo em porcentagem da Engrenagem (Fratura).



(c) Tempo absoluto do Rotor.

(d) Tempo em porcentagem do Rotor.



(e) Tempo absoluto da Engrenagem.

(f) Tempo em porcentagem da Engrenagem.

Figura 4.26 – Detalhamento do tempo de execução absoluto (coluna da esquerda) e em porcentagem (coluna da direita).

Os gráficos mostram que a geração de malhas nas interfaces é um fator limitante para o *speed-up*. Assim, uma melhor decomposição poderia levar a menores interfaces, diminuindo esse tempo e melhorando o *speed-up*. Entretanto, como a decomposição é baseada na estimativa de carga, é preciso melhorar a

estimativa de carga para melhorar a decomposição. Perceba, também, que todos os modelos estão no limite do número de processos, isto é, adicionar mais processos provavelmente aumentaria o número de interfaces e, portanto, o tempo necessário para que as suas malhas fossem geradas.

Além disso, todos esses modelos seguem a mesma tendência do modelo da Viga de que a geração e a classificação da *octree*, quando o número de processos cresce, torna-se proporcionalmente grande, podendo chegar a cerca de 25% do tempo total, e necessita ser otimizado ou paralelizado.

#### 4.4.3 Detalhamento do tempo de execução de geração de malha

As duas colunas da Figura 4.27 detalham o tempo de execução dos procedimentos de geração de malha, em termos absolutos e relativos, respectivamente. Perceba que boa parte do tempo de execução da geração das malhas de interface são gastos em comunicação, indicando que os subdomínios podem não estar bem balanceados.

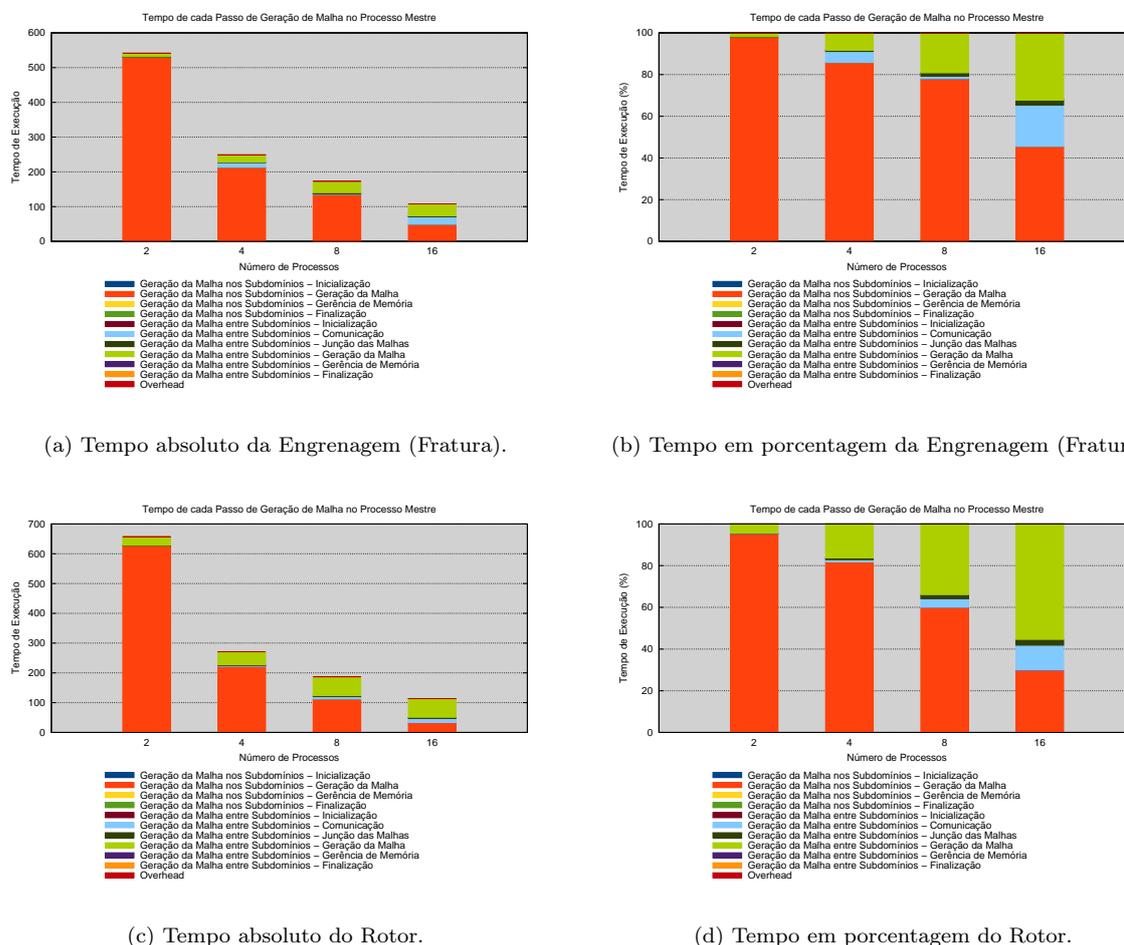
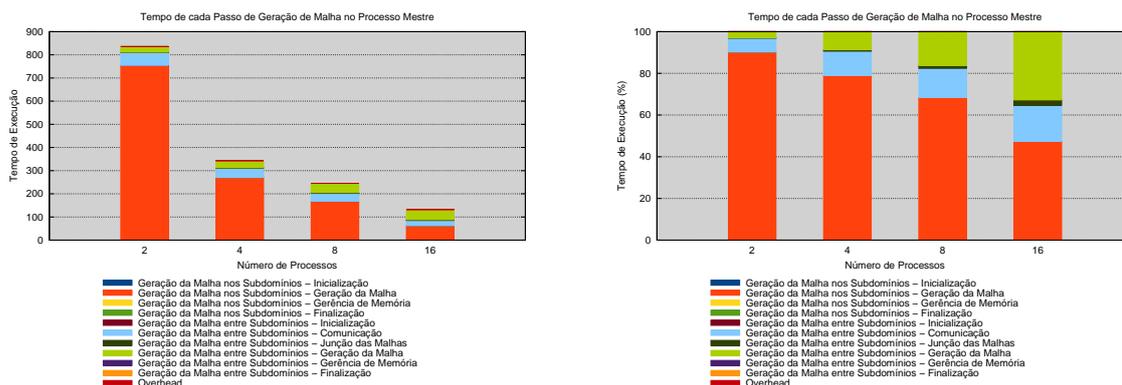


Figura 4.27 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita).

#### 4.4.4 Balanceamento da carga

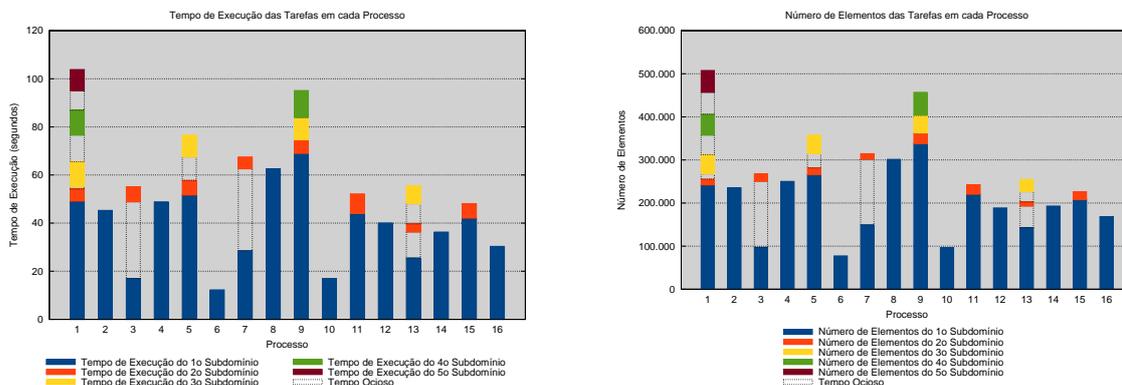
A coluna da esquerda da Figura 4.28 mostra o tempo de execução de cada subdomínio em cada processo e a coluna da direita da mesma figura mostra a quantidade de elementos gerados em cada subdomínio e em cada interface. Esses gráficos corroboram o que foi dito anteriormente, que o tempo de espera está prejudicando a implementação paralela.



(e) Tempo absoluto da Engrenagem.

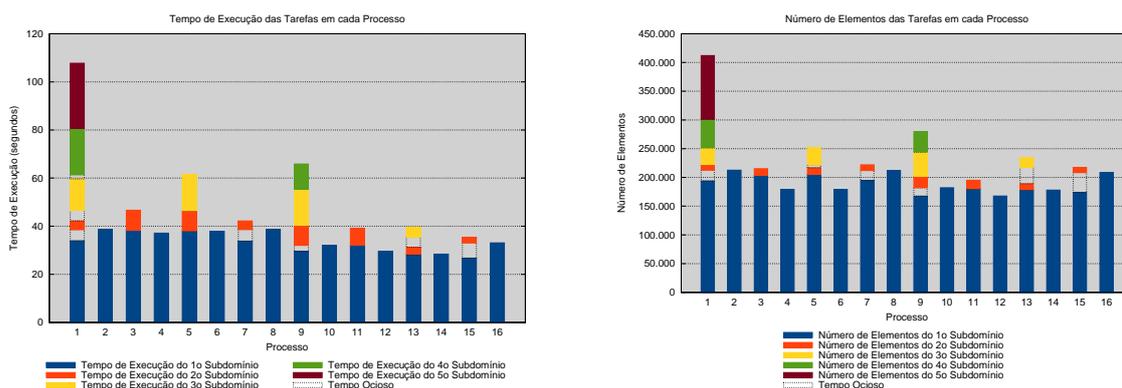
(f) Tempo em porcentagem da Engrenagem.

Figura 4.27 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita) (continuação).



(a) Tempo de execução da Engrenagem (Fratura).

(b) Número de elementos da Engrenagem (Fratura).

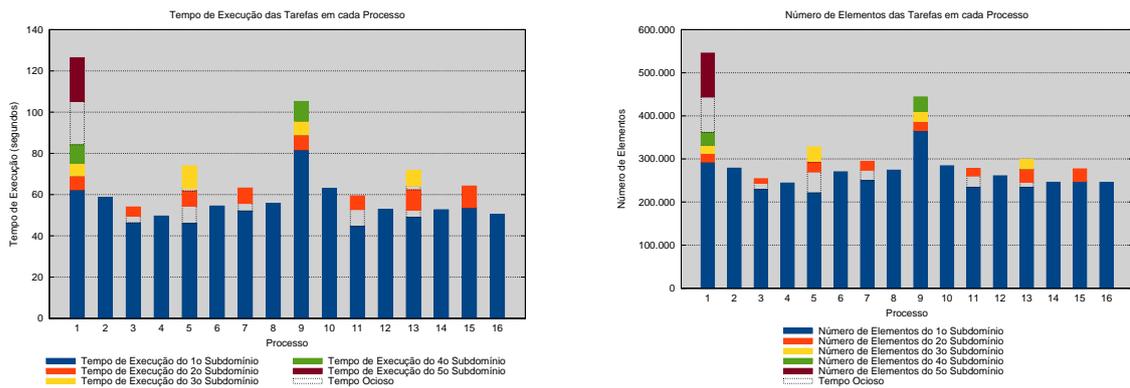


(c) Tempo de execução do Rotor.

(d) Número de elementos do Rotor.

Figura 4.28 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, com 16 processos.

A razão de o modelo da Engrenagem (Fratura) (Figuras 4.28a e 4.28b) estar mal balanceado é que o refinamento uniforme da *octree* de densidade não foi ativado. Assim, os tetraedros internos à malha podem ser maiores que o maior tetraedro na borda e, conseqüentemente, existe mais transição entre



(e) Tempo de execução da Engrenagem.

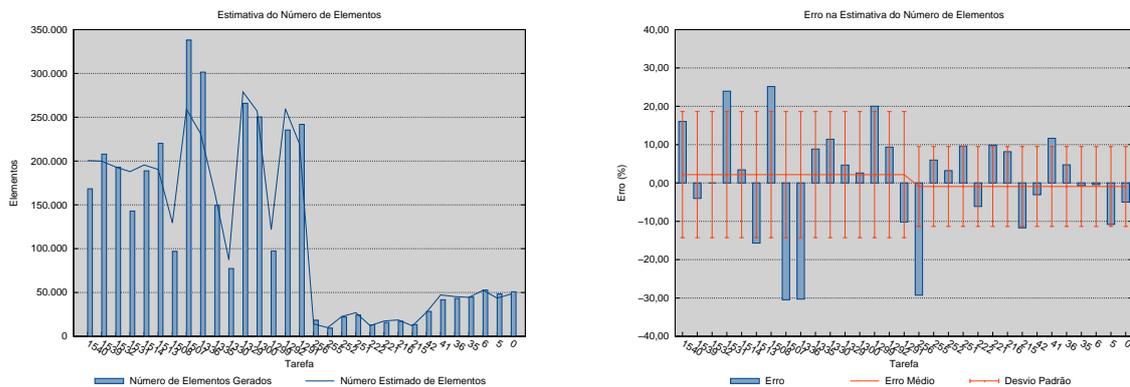
(f) Número de elementos da Engrenagem.

Figura 4.28 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, com 16 processos (continuação).

elementos pequenos e elementos grandes, fazendo com que a diferença entre níveis das células da *octree* em um subdomínio tenha maior influência na carga, algo que não está sendo levado em conta durante a estimativa. Neste modelo, entretanto, as interfaces são pequenas, fazendo com que o seu *speed-up* seja bom.

#### 4.4.5 Estimativa da carga

A coluna da esquerda da Figura 4.29 apresenta a estimativa do número de elementos em cada tarefa, seja ela um subdomínio ou uma interface, para os três modelos práticos, quando eles foram executados com 16 processos. A coluna da direita da mesma figura mostra o erro envolvido na estimativa da carga para os mesmos casos, bem como a média e o desvio padrão dos erros. Perceba que o modelo da Engrenagem (Fratura) foi o que apresentou maior erro, com um desvio padrão chegando a quase 20%, ocasionando o maior desbalanceamento deste modelo, como observado na seção passada.

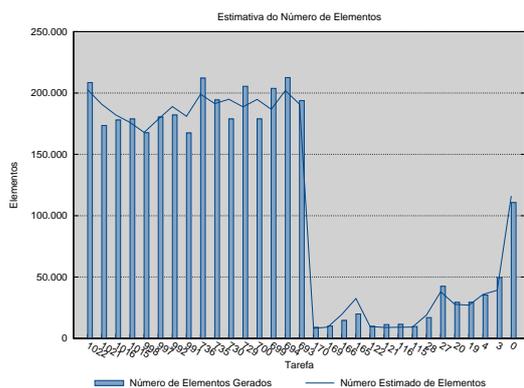


(a) Estimativa da Engrenagem (Fratura).

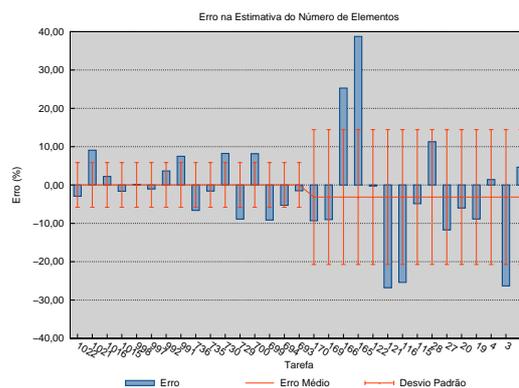
(b) Erro na estimativa da Engrenagem (Fratura).

Figura 4.29 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces (coluna da esquerda) e erro na estimativa do número de elementos de cada tarefa (coluna da direita), com 16 processos.

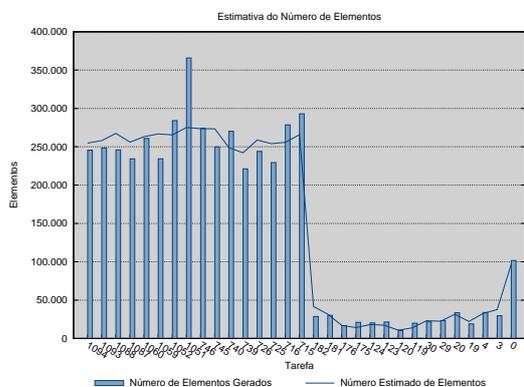
A Figura 4.30 mostra o erro médio e o desvio padrão da estimativa do número de elementos nos subdomínios para os três modelos práticos.



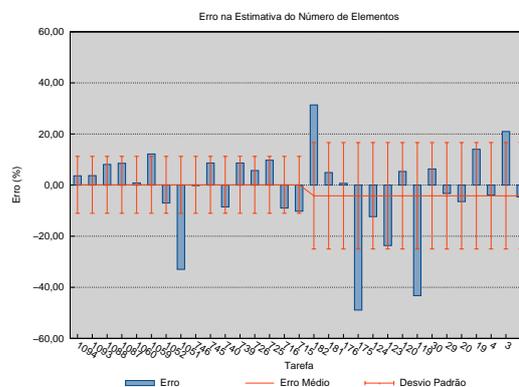
(c) Estimativa do Rotor.



(d) Erro na estimativa do Rotor.



(e) Estimativa da Engrenagem.



(f) Erro na estimativa da Engrenagem.

Figura 4.29 – Estimativa do número de elementos em cada tarefa, subdomínios ou interfaces (coluna da esquerda) e erro na estimativa do número de elementos de cada tarefa (coluna da direita), com 16 processos (continuação).

Uma tendência facilmente vista no gráfico é a mesma do modelo da Viga de que, quanto mais processos, mais planos de partição e, conseqüentemente, maior o erro na estimativa de carga.

No modelo com fratura, a *octree* é bastante refinada ao redor dela, o que pode ser ruim para a estimativa de carga, uma vez que todas as células da *octree* têm o mesmo peso, independente da sua profundidade. No mesmo modelo, além de a fratura ser grande, há uma maior transição entre elementos de tamanhos diferentes, prejudicando a estimativa, principalmente com 16 processos.

Os dois últimos modelos, que não têm fratura, foram os de melhor estimativa. Assumindo que uma boa estimativa é uma cujo erro seja, no máximo, por volta de 10%, então os três modelos, podem ser considerados com uma boa estimativa, exceto o da Engrenagem (Fratura) para 16 processos.

#### 4.4.6 Qualidade das malhas

A coluna da esquerda da Figura 4.31 mostra a qualidade das malhas geradas, em porcentagem, para os três modelos práticos. Pode-se ver que a maioria dos elementos têm boa qualidade,  $\alpha \geq 0,7$ . A coluna da direita da mesma figura apresenta a diferença dessa porcentagem entre as malhas geradas em paralelo e a malha gerada sequencialmente.

Perceba, pela coluna da direita da Figura 4.31, que a maioria dos elementos que tiveram qualidade modificada passaram de um qualidade de média para boa ( $\alpha \geq 0,5$ ) para uma qualidade de ruim para

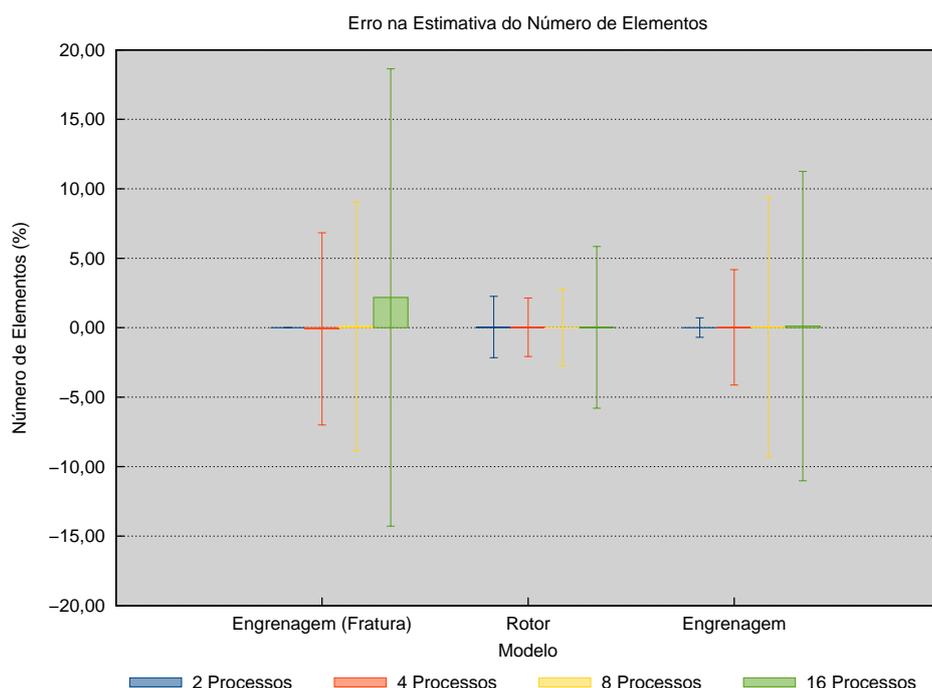
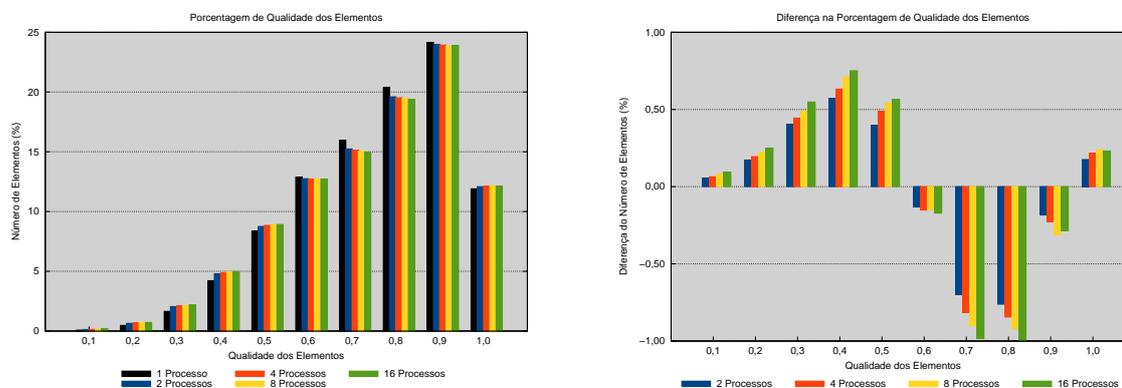


Figura 4.30 – Erro médio e desvio padrão da estimativa do número de elementos para os modelos práticos.

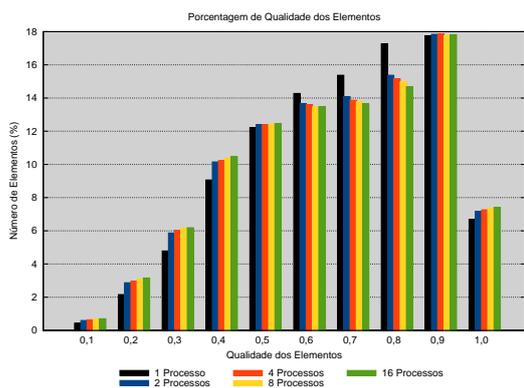


(a) Qualidade das malhas da Engrenagem (Fratura). (b) Diferença na qualidade das malhas da Engrenagem (Fratura).

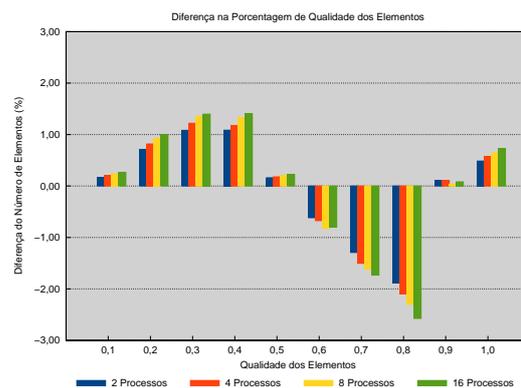
Figura 4.31 – Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença na porcentagem da qualidade das malhas geradas (coluna da direita).

média ( $\alpha \leq 0,5$ ). Entretanto, o pior caso foi o do modelo do Rotor (Figura 4.31d), cuja quantidade de elementos que deixaram de ter qualidade  $0,8 \leq \alpha < 0,9$  correspondeu a 2,5% dos elementos gerados.

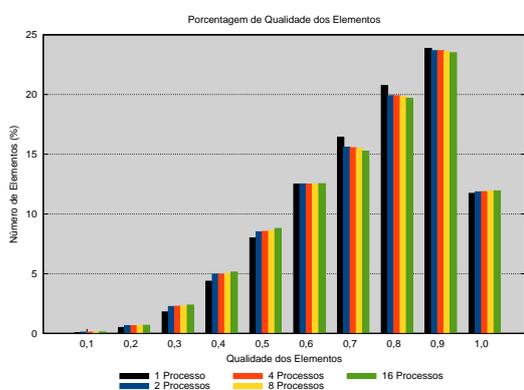
A Figura 4.32 apresenta a diferença total na porcentagem da qualidade das malhas geradas em paralelo para a malha gerada sequencialmente. O pior caso foi o do Rotor, cuja quantidade total de elementos que mudaram de qualidade correspondeu a pouco mais de 10%. Perceba que, assim como no modelo da Viga, existe a tendência de que, quanto mais planos de partição, mais diferente é a malha gerada em paralelo, havendo assim, um *trade-off* entre qualidade da malha gerada e rapidez de geração. Entretanto, vê-se que a primeira interface é a que mais influencia na diferença da qualidade da malha, pois a variação da



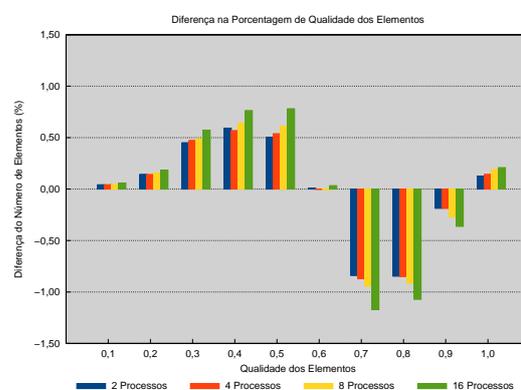
(c) Qualidade das malhas do Rotor.



(d) Diferença na qualidade das malhas do Rotor.



(e) Qualidade das malhas da Engrenagem.



(f) Diferença na qualidade das malhas da Engrenagem.

Figura 4.31 – Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença na porcentagem da qualidade das malhas geradas (coluna da direita) (continuação).

diferença com 2 processos para a diferença com 16 processos é de cerca de apenas 2,5%, no pior caso.

Por último, veja que a qualidade da malha não tem relacionamento nenhum com a estimativa de carga nem com o *speed-up*, sejam eles bons ou ruins. Isso quer dizer que a ideia de parar o avanço da fronteira quando esta se aproxima do plano de partição mantém, a menos de um pequeno erro, a qualidade da malha aproximadamente a mesma da gerada sequencialmente.

#### 4.4.7 Malhas

A Figura 4.33 mostra as bordas das malhas geradas em paralelo para os modelos práticos, com 16 processos.

### 4.5 Considerações finais

Este capítulo mostrou os resultados obtidos pela execução do algoritmo paralelo em 1 modelo teórico e em 3 modelos práticos. O modelo da Viga foi utilizado para se encontrar o valor do limiar de balanceamento entre as cargas dos subdomínios e as cargas das interfaces, na escolha do melhor eixo de posicionamento do plano de partição, durante a decomposição do domínio, e o mesmo valor do limiar foi utilizado nos outros modelos.

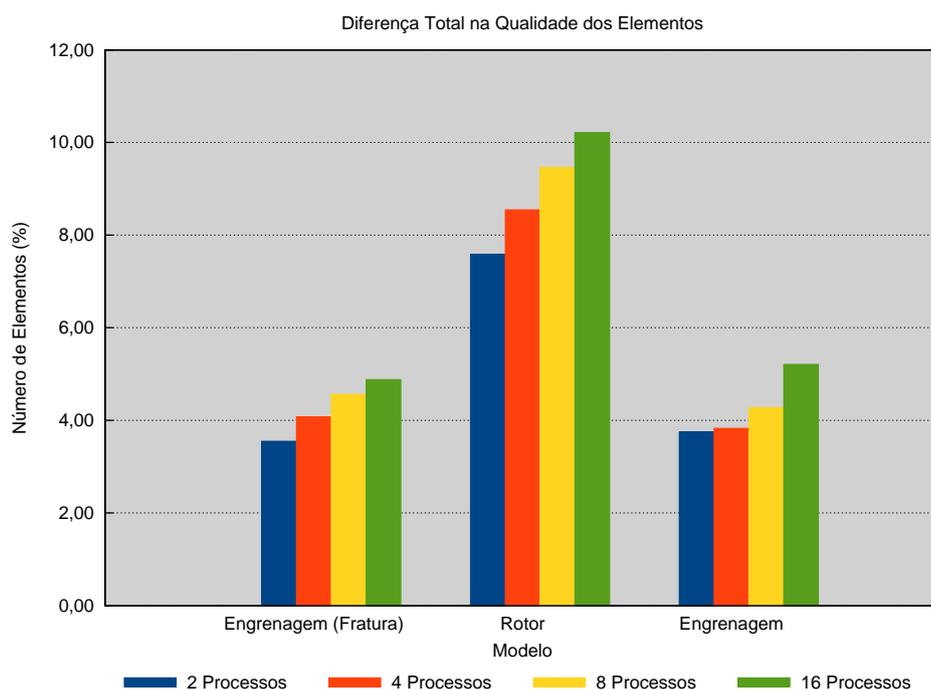


Figura 4.32 – Diferença na porcentagem da qualidade das malhas geradas.

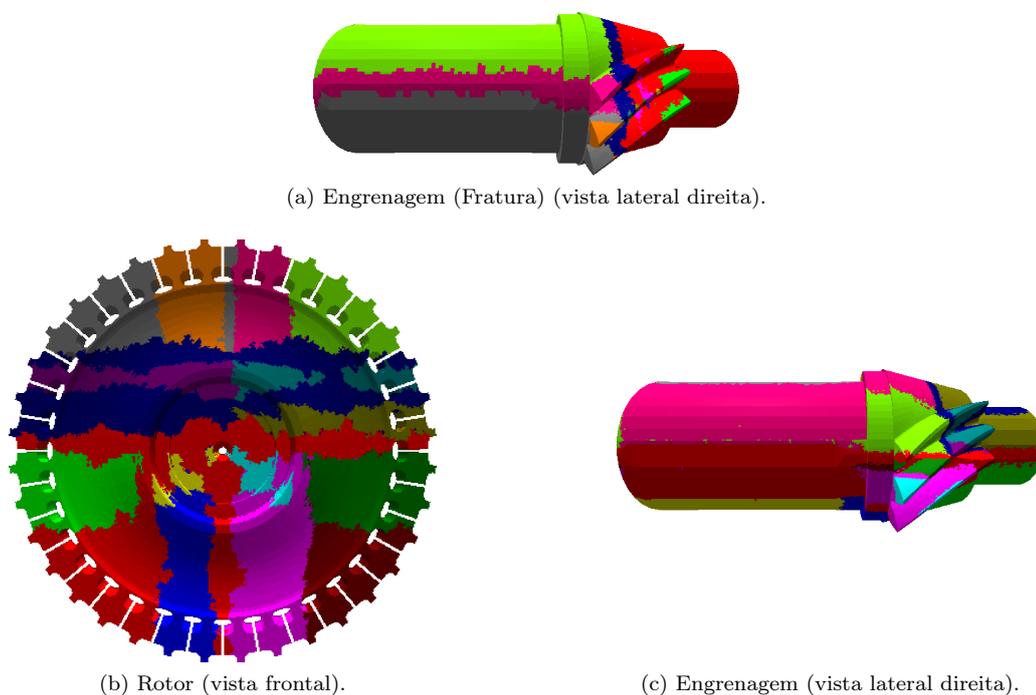


Figura 4.33 – Malhas para os modelos práticos.

Os modelos considerados com boa estimativa são aqueles cujo erro na estimativa do número de elementos é, no máximo, próximo a 10%. O mesmo limiar de 10% foi quase sempre atendido no que diz respeito à diferença na qualidade de uma malha gerada em paralelo para a malha gerada sequencialmente, ou seja, as malhas geradas em paralelo tiveram aproximadamente a mesma qualidade da malha gerada pela técnica sequencial.

O próximo capítulo apresenta as conclusões acerca deste trabalho, destacando as suas principais contribuições, identifica os pontos em que este trabalho pode melhorar e descreve algumas ideias para que isso seja feito.



## 5 CONCLUSÃO

### 5.1 Principais contribuições

Este trabalho apresentou uma técnica paralela de geração de malhas que pode ser implementada tanto em duas quanto em três dimensões, e funciona tanto para computadores de memória compartilhada quanto de memória distribuída. A ideia principal da técnica é decompor o domínio utilizando uma estrutura de árvore binária, de forma que todas as folhas desta árvore tenham aproximadamente a mesma carga. Após gerar a malha nas folhas da árvore, em paralelo, as malhas de interface são geradas, conectando os subdomínios. Isso caracteriza a técnica como de decomposição contínua de domínio com geração de interfaces *a posteriori*.

Por utilizar decomposição contínua do domínio com geração de interfaces *a posteriori*, a técnica paralela não tem uma influência negativa na geração da malha, e herda as características presentes no gerador de malhas sequencial. O algoritmo sequencial que foi paralelizado é de avanço de fronteira, um tipo de técnica que trata naturalmente a existência de buracos no interior do modelo. Além disso, o algoritmo sequencial também trata fraturas no interior ou na superfície do modelo, que também podem ser utilizadas para representar superfícies internas de modelos com várias regiões.

Outras duas características herdadas pela técnica paralela da técnica sequencial são de respeitar a fronteira dada como entrada e de fornecer boa transição entre regiões menos refinadas e regiões mais refinadas da malha. A última característica herdada foi a da qualidade da malha. Apesar de a decomposição do domínio afetar um pouco a qualidade da malha, ela é comparável à da malha gerada sequencialmente.

Neste trabalho, a decomposição do modelo e, conseqüentemente, o balanceamento da carga, depende da estrutura utilizada para a estimativa da carga, que foi uma *octree*. Como os modelos são alinhados aos eixos e, portanto, à *octree*, a estimativa foi bem feita, dentro de um erro de aproximadamente 10%. Esse alinhamento é importante, pois um desalinhamento pode comprometer a estimativa da carga e, conseqüentemente, a boa distribuição de carga entre os subdomínios.

O último fator relevante para um bom *speed-up* é o tamanho das interfaces. Como as interfaces são geradas *a posteriori*, se a carga de uma interface corresponder a uma boa porcentagem da carga dos lados decompostos por ela, o *speed-up* pode ficar prejudicado. Assim, para modelos compridos em um ou dois eixos e curto no(s) outro(s), a técnica tem um bom desempenho, desde que não haja plano de partição paralela a um dos eixos compridos.

Finalmente, apesar de utilizar uma árvore binária para decompor o domínio, por conta do fator de proporcionalidade utilizado na decomposição, qualquer número de processos pode ser utilizado, mesmo que não seja potência de dois.

### 5.2 Trabalhos futuros

Vários pontos ainda podem ser melhorados neste trabalho. A primeira otimização diz respeito à geração, aos refinamentos e à classificação da *octree*, que são feitos sequencialmente. Apesar de não ser o passo que mais gasta tempo, na técnica paralela, conforme o número de processos aumenta, esse tempo torna-se proporcionalmente grande, podendo chegar a cerca de 25% do tempo total de execução. Assim, paralelizar esse passo torna-se essencial para a geração de malhas maiores e utilizando mais que 16 processos. Além disso, como a estrutura da *octree* é a mesma em todos os processos, ela pode ocupar muito espaço na memória, espaço este que poderia ser utilizado para guardar a malha em vez de uma estrutura auxiliar. A paralelização da *octree* também resolveria este problema, distribuindo-a pelos processos.

A segunda melhoria seria na estimativa da carga. Cada célula da *octree* tem a mesma contribuição para a estimativa de carga, e uma estimativa ponderada pela altura em que a célula se encontra na árvore parece ser um melhor caminho. Entretanto, são necessários estudos mais aprofundados da estimativa de carga para se achar esse peso dependente do nível.

Outra medida a ser tomada que deve favorecer a estimativa da carga é sempre garantir o alinhamento da *octree* e da BSP ao modelo. Analogamente, pode-se aplicar alguma transformação linear no modelo para deixá-lo alinhado aos eixos globais, gerar a malha, e aplicar a transformação inversa à malha para levá-la à orientação original. Tal transformação pode ser uma mudança de base utilizando os eixos principais do modelo, encontrados como os autovetores unitários da matriz de covariância do fecho convexo do modelo (GOTTSCHALK et al., 1996).

No que diz respeito à decomposição, poderia-se relaxar a restrição de que os planos de partição não devem cruzar as células cheias da *octree*. Com isso, bastaria pegar, por exemplo, a mediana dos centroides das células da *octree*, como em De Cougny et al. (1994), em vez de descer a estrutura da *octree* para encontrar o melhor posicionamento do plano de partição.

Outra melhoria na decomposição é permitir que eixos quaisquer possam ser utilizados no posicionamento do plano de partição, podendo-se utilizar eixos diagonais em vez de somente os três eixos globais. Com isso, os eixos principais poderiam ser utilizados para decompor o domínio, e poderia-se comparar qual eixo decompõe melhor o domínio.

Algo que é comumente encontrado em técnicas paralelas é a chamada *over-decomposition*, ou seja, permitir que um mesmo processo seja responsável por vários subdomínios. Isso pode ser feito continuando-se a subdivisão da árvore BSP depois que é encontrado o subdomínio de um determinado processo ou de uma determinada *thread*.

Um fator que diminui a generalidade da técnica paralela é o fato de que, na geração de uma malha de interface, além da fronteira dada como entrada, é dada também uma malha volumétrica inicial, correspondente a uma camada de elementos adjacentes à fronteira, proveniente dos dois nós filhos, na estrutura da BSP. Isso não é comum em implementações de avanço de fronteira, em que somente a fronteira é dada como entrada. Assim, para aumentar a generalidade, é possível apagar essa malha volumétrica, encontrar a fronteira resultante, e utilizar essa fronteira como entrada para o algoritmo de avanço de fronteira.

Ainda no que diz respeito às interfaces, para melhorar o *speed-up*, pode-se gerar malhas de superfície de interface *a priori*, e utilizá-las como entrada para os subdomínios. Espera-se que isso aumente o *speed-up*, já que o tempo de geração das malhas de interface diminui. Entretanto, espera-se que a qualidade da malha também diminua, uma vez que essas malhas de interface geradas *a priori* podem introduzir artefatos no interior da malha. Gerar as malhas de interface *a priori* também aumenta a generalidade da técnica paralela, uma vez que qualquer algoritmo sequencial pode ser utilizado, sem nenhuma modificação.

Para aplicações paralelas de Métodos de Elementos Finitos, é interessante que a malha já chegue particionada, evitando um passo intermediário de particionamento depois da geração da malha. A malha gerada pela técnica descrita neste trabalho, ao seu fim, já está particionada, entretanto, este particionamento pode não ser bom, por que o processo mestre termina com a maior parte da malha. Esta malha pode ser reparticionada entre os diversos processos, utilizando os próprios planos de partição da decomposição para decidir para qual processo um elemento deve ser enviado.

Apesar de a implementação ser independente do tipo de memória, ela foi testada apenas em computadores de memória distribuída, e seu comportamento em memória compartilhada ainda deve ser analisado. Devido à limitação da memória, o propósito da versão de memória compartilhada é o de gerar malhas menores que as geradas em computadores de memória distribuída. Além disso, por conta da não existência de comunicação entre processos, acredita-se que esta versão pode se comportar melhor que a de memória distribuída. Entretanto, em testes preliminares realizados no início da implementação, o gargalo da versão de memória compartilhada era a alocação de memória, que pode ser muito intensa.

Uma melhoria que aumenta a robustez da técnica, tanto da sequencial quanto da paralela, é a utilização de aritmética exata (SHEWCHUK, 1997) para os testes de proximidade e de interseção de faces. Com isso, a técnica deixaria de falhar devido a erros numéricos de precisão ou de truncamento.

Para aumentar a confiabilidade dos resultados de *speed-up* da técnica, esta deve ser executada várias vezes para uma mesma configuração de um mesmo modelo, eliminando amostras atípicas (*outliers*) de tempo de execução e calculando-se uma média das medições feitas.

Por último, deseja-se incorporar o gerador paralelo de malhas em um método paralelo de elementos finitos, e analisar não somente a precisão dos resultados como também a rapidez da simulação.



## Apêndice



# APÊNDICE A – ANÁLISE DA TÉCNICA PARALELA

## A.1 Introdução

Este apêndice faz uma análise do uso da BSP para decompor o domínio de forma a distribuir uniformemente o número de elementos entre os subdomínios. Inicialmente, a análise é feita para o caso bidimensional e, em seguida, essa mesma análise é estendida para três dimensões.

A análise é feita para o pior caso de geração de malha, que vem a ser um simples modelo uniforme de um quadrado ou um cubo. Isso acontece por que estes modelos maximizam a área ou o volume, e a *quadtrees/octrees* de estimativa de carga é uma árvore completa onde todos as folhas estão no mesmo nível, ou seja, têm o mesmo tamanho, e são células cheias, isto é, são utilizadas para o cálculo da estimativa de carga.

## A.2 Complexidade

É difícil calcular a complexidade de algoritmos de avanço de fronteira, uma vez que ela depende da quantidade de elementos de fronteira existentes na malha final em vez do tamanho da entrada, ou seja, da quantidade de arestas (em 2D) ou polígonos (em 3D) da fronteira. Uma complexidade computacional de  $O(N \log N)$  foi reportada na literatura, sendo  $N$  o número de elementos gerados (LÖHNER; PARIKH, 1988; BONET; PERAIRE, 1991). Foi também reportado que uma complexidade de  $O(N^p \log N)$ , com  $p$  um pouco maior do que 1, é uma medida mais realista (JIN; TANNER, 1993; MÖLLER; HANSBO, 1995).

Nesse sentido, como a complexidade muitas vezes depende da implementação do algoritmo de avanço de fronteira, ela é geralmente calculada por execuções exaustivas, resultando em um tempo de execução empírico  $T(N) = CN^p \log N$  onde  $N$  é o número de elementos da malha gerada. Neste trabalho, essa complexidade foi usada, com  $0 < C \leq 1$  e  $1 \leq p \leq 2$  (CAVALCANTE-NETO et al., 2001).

Como visto anteriormente, a técnica descrita neste trabalho tem duas fases de geração de malha, uma que gera malha nos subdomínios e outra que trabalha nas interfaces entre dois subdomínios (Seção 3.3.3). Assumindo que todos os subdomínios têm, aproximadamente, a mesma estimativa de carga, eles geram aproximadamente o mesmo número de elementos, e as mesmas constantes  $C$  e  $p$  podem ser utilizadas para eles, uma vez que a mesma implementação da mesma técnica sequencial de avanço de fronteira é utilizada em todos os subdomínios.

Nas interfaces, entretanto, o número de elementos varia com o tamanho da interface, que pode depender do nível da árvore da BSP em que está a interface. Além disso, na geração de malha da interface, existe uma quantidade de elementos dados como entrada, fazendo com que a etapa de melhoria gaste um tempo maior do que gastaria se fosse aplicada somente na malha gerada na interface.

Portanto, as constantes  $C$  e  $p$  das interfaces, apesar de iguais entre si, podem ter valores diferentes das constantes utilizadas para os subdomínios, as folhas da BSP. Assim,  $C_s$  e  $p_s$  denotam as constantes para a geração de malha em um subdomínio e  $C_i$  e  $p_i$  denotam as mesmas constantes para a geração de malha em uma interface.

Seja  $P$  o número de processos na arquitetura paralela, onde somente uma *thread* é utilizada por processo. Então, a BSP é criada com  $P$  folhas e sua altura é  $\lceil \log_2 P \rceil + 1$ , o que significa que existem  $\lceil \log_2 P \rceil$  níveis de geração de malha de interface. Todos os  $P$  subdomínios são executados simultaneamente, bem como todas as tarefas de geração de malha de interface em cada nível.

Assumindo que a estimativa de carga é bem feita, como todos os subdomínios têm aproximadamente a mesma estimativa, tem-se que  $N_s = N/P$ , onde  $N_s$  é o número de elementos de um subdomínio. Se  $N_l$  é o maior número de elementos gerados em uma interface no nível  $l \geq 1$  da BSP, então  $N_l$  pode variar de acordo com nível  $l$ .

Portanto, o tempo de execução do algoritmo paralelo apresentado neste trabalho é dado pela Equação A.1:

$$\begin{aligned} T_P(N) &= T(N_s) + \sum_{l=1}^{\lceil \log_2 P \rceil} T(N_l) \\ &= C_s N_s^{p_s} \log N_s + \sum_{l=1}^{\lceil \log_2 P \rceil} T(C_i N_l^{p_i} \log N_l). \end{aligned} \quad (\text{A.1})$$

### A.2.1 Caso bidimensional

Para o caso bidimensional, já que  $N$  é o número de elementos em um domínio planar,  $\sqrt{N}$  pode ser considerado o número de elementos em uma região linear. Uma interface tem um comportamento linear, combinado com uma largura de 2 camadas de região livre (1 camada de região livre de cada subdomínio adjacente), mais 2 camadas de elementos (1 camada de cada subdomínio adjacente).

No modelo do quadrado uniforme, a subdivisão da BSP é sempre feita alternando-se os eixos X e Y, e uma interface no eixo X tem o mesmo tamanho da interface no eixo Y do nível anterior. Quando a subdivisão passa do eixo X para o eixo Y, o tamanho da interface é dividido por 2. Logo, o tamanho da interface depende do valor de  $\lfloor \frac{l}{2} \rfloor$ , e o número de elementos de uma interface no nível  $l$  é aproximadamente  $N_l = 4 \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}$ .

O tempo de execução do algoritmo paralelo apresentado neste trabalho, em duas dimensões, é dado na Equação A.2:

$$\begin{aligned} T_P(N) &= T(N_s) + \sum_{l=1}^{\lceil \log_2 P \rceil} T(N_l) \\ &= C_s \left(\frac{N}{P}\right)^{p_s} \log\left(\frac{N}{P}\right) + \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ C_i \left(4 \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right)^{p_i} \log\left(4 \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right) \right] \\ &\leq c_s \left(\frac{N}{P}\right)^{p_s} \log\left(\frac{N}{P}\right) + c_i \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \left(\frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right)^{p_i} \log\left(\frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right) \right]. \end{aligned} \quad (\text{A.3})$$

Na Equação A.3, assume-se que  $\log\left(\frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right) \geq 1$ . O pior caso para a constante  $C$  é quando  $C_s = C_i = 1$ . Para  $p$ , o caso mais comum é  $p_s = p_i \cong 1$ . Nesses casos, a complexidade da técnica paralela torna-se como na Equação A.4:

$$T_P(N) \leq \frac{N}{P} \log\left(\frac{N}{P}\right) + \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \log\left(\frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}}\right) \right]. \quad (\text{A.4})$$

Como  $P \geq 2$ ,  $\log_2 P \geq 1$  e, portanto,  $\lceil \log_2 P \rceil \geq 1$ . O somatório da Equação A.4 pode ser escrito

como na Equação A.5:

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \log \left( \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \right) \right] &= \sqrt{N} \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} \left( \log \sqrt{N} - \log 2^{\lfloor \frac{l}{2} \rfloor} \right) \right] \\
 &= \sqrt{N} \left[ \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\log \sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} - \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\log 2^{\lfloor \frac{l}{2} \rfloor}}{2^{\lfloor \frac{l}{2} \rfloor}} \right] \\
 &= \sqrt{N} \left[ \log \sqrt{N} \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} - \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} \right]. \tag{A.5}
 \end{aligned}$$

Dois casos são possíveis para  $\lceil \log_2 P \rceil$ , ele pode ser ímpar ( $\lceil \log_2 P \rceil = 2k + 1$ ) ou par ( $\lceil \log_2 P \rceil = 2k$ ). Para os dois casos, o primeiro somatório da Equação A.5 pode ser reescrito como nas Equações A.6 e A.7:

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} &= \sum_{l=1}^{2k+1} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} \\
 &= \frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k} + \frac{1}{2^k} \\
 &= 1 + 2 \left[ \sum_{i=1}^k \frac{1}{2^i} \right] = 1 + 2 \left[ 1 - \frac{1}{2^k} \right] = 1 + 2 - \frac{2}{2^k} = 3 - \frac{2}{2^k}, \tag{A.6}
 \end{aligned}$$

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} &= \sum_{l=1}^{2k} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} \\
 &= \sum_{l=1}^{2k-1+1} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} = \sum_{l=1}^{2k-1} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} + \frac{1}{2^{\lfloor \frac{2k}{2} \rfloor}} = \left[ 3 - \frac{2}{2^{k-1}} \right] + \frac{1}{2^k} \\
 &= 3 - \frac{4}{2^k} + \frac{1}{2^k} = 3 - \frac{3}{2^k}. \tag{A.7}
 \end{aligned}$$

Substituindo de volta o valor de  $k$  por  $\frac{\lceil \log_2 P \rceil - 1}{2}$  ou  $\frac{\lceil \log_2 P \rceil}{2}$ , respectivamente, para os casos ímpar e par, as Equações A.6 e A.7 tornam-se as Equações A.8 e A.9:

$$3 - \frac{2}{2^k} = 3 - \frac{2}{2^{\frac{\lceil \log_2 P \rceil - 1}{2}}} = 3 - \frac{2}{\sqrt{2^{\lceil \log_2 P \rceil - 1}}} = 3 - \frac{2}{\sqrt{\frac{2^{\lceil \log_2 P \rceil}}{2}}} = 3 - \frac{2\sqrt{2}}{\sqrt{2^{\lceil \log_2 P \rceil}}}, \tag{A.8}$$

$$3 - \frac{3}{2^k} = 3 - \frac{3}{2^{\frac{\lceil \log_2 P \rceil}{2}}} = 3 - \frac{3}{\sqrt{2^{\lceil \log_2 P \rceil}}}. \tag{A.9}$$

Logo, como  $3 > 2\sqrt{2}$ , a Equação A.10 é verdadeira:

$$\sum_{l=1}^{\lceil \log_2 P \rceil} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} \leq 3 - \frac{2\sqrt{2}}{\sqrt{2^{\lceil \log_2 P \rceil}}}. \tag{A.10}$$

Similarmente, o segundo somatório da Equação A.5 pode ser reescrito como as Equações A.11 e A.12, dependendo se  $\lceil \log P \rceil$  é ímpar ou par, respectivamente:

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} &= \sum_{l=1}^{2k+1} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} \\
 &= \frac{0}{2^0} + \frac{1}{2^1} + \frac{1}{2^1} + \frac{2}{2^2} + \frac{2}{2^2} + \cdots + \frac{k}{2^k} + \frac{k}{2^k} \\
 &= 0 + 2 \left[ \sum_{i=1}^k \frac{i}{2^i} \right] = 2 \left[ 2 - \frac{k+2}{2^k} \right] = 4 - \frac{2k+4}{2^k}, \tag{A.11}
 \end{aligned}$$

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} &= \sum_{l=1}^{2k} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} \\
 &= \sum_{l=1}^{2k-1+1} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} = \sum_{l=1}^{2k-1} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} + \frac{\lfloor \frac{2k}{2} \rfloor}{2^{\lfloor \frac{2k}{2} \rfloor}} = \left[ 4 - \frac{2(k-1)+4}{2^{k-1}} \right] + \frac{k}{2^k} \\
 &= 4 - \frac{2k+2}{2^{k-1}} + \frac{k}{2^k} = 4 - \frac{4k+4}{2^k} + \frac{k}{2^k} = 4 - \frac{3k+4}{2^k}. \tag{A.12}
 \end{aligned}$$

Substituindo de volta o valor de  $k$  por  $\frac{\lceil \log_2 P \rceil - 1}{2}$  ou  $\frac{\lceil \log_2 P \rceil}{2}$ , respectivamente, para os casos ímpar e par, as Equações A.11 e A.12 tornam-se as Equações A.13 e A.14:

$$\begin{aligned}
 4 - \frac{2k+4}{2^k} &= 4 - \frac{2 \frac{\lceil \log P \rceil - 1}{2} + 4}{2^{\frac{\lceil \log P \rceil - 1}{2}}} = 4 - \frac{\lceil \log P \rceil - 1 + 4}{\sqrt{2^{\lceil \log P \rceil - 1}}} = 4 - \frac{\lceil \log P \rceil + 3}{\sqrt{\frac{2^{\lceil \log P \rceil}}{2}}} \\
 &= 4 - \sqrt{2} \frac{\lceil \log P \rceil + 3}{\sqrt{2^{\lceil \log P \rceil}}} = 4 - \sqrt{2} \frac{\lceil \log P \rceil}{\sqrt{2^{\lceil \log P \rceil}}} - \frac{3\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}}, \tag{A.13}
 \end{aligned}$$

$$4 - \frac{3k+4}{2^k} = 4 - \frac{3 \frac{\lceil \log P \rceil}{2} + 4}{2^{\frac{\lceil \log P \rceil}{2}}} = 4 - \frac{3\lceil \log P \rceil + 8}{2\sqrt{2^{\lceil \log P \rceil}}} = 4 - \frac{3}{2} \frac{\lceil \log P \rceil}{\sqrt{2^{\lceil \log P \rceil}}} - \frac{4}{\sqrt{2^{\lceil \log P \rceil}}}. \tag{A.14}$$

Logo, como  $\frac{3}{2} > \sqrt{2}$  e  $3\sqrt{2} > 4$ , a Equação A.15 é verdadeira:

$$- \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} \leq - \left( 4 - \frac{3}{2} \frac{\lceil \log P \rceil}{\sqrt{2^{\lceil \log P \rceil}}} - \frac{3\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right). \tag{A.15}$$

Substituindo as Equações A.10 e A.15 na Equação A.5, o somatório torna-se como na Equação A.16:

$$\begin{aligned}
 \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \log \left( \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \right) \right] &= \sqrt{N} \left[ \log \sqrt{N} \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{1}{2^{\lfloor \frac{l}{2} \rfloor}} - \sum_{l=1}^{\lceil \log_2 P \rceil} \frac{\lfloor \frac{l}{2} \rfloor}{2^{\lfloor \frac{l}{2} \rfloor}} \right] \\
 &\leq \sqrt{N} \left[ \frac{\log N}{2} \left( 3 - \frac{2\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right) \right. \\
 &\quad \left. - \left( 4 - \frac{3}{2} \frac{\lceil \log P \rceil}{\sqrt{2^{\lceil \log P \rceil}}} - \frac{3\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right) \right] \\
 &\leq \sqrt{N} \left[ \frac{3 \log N}{2} - \frac{2\sqrt{2} \log N}{2\sqrt{2^{\lceil \log P \rceil}}} - 4 + \frac{3\lceil \log P \rceil}{2\sqrt{2^{\lceil \log P \rceil}}} + \frac{3\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right] \\
 &\leq \sqrt{N} \left[ \frac{3 \log N}{2} - 4 + \frac{3\lceil \log P \rceil - 2\sqrt{2} \log N + 6\sqrt{2}}{2\sqrt{2^{\lceil \log P \rceil}}} \right] \\
 &\leq \frac{\sqrt{N}}{2} \left[ 3 \log N - 8 + \frac{3\lceil \log P \rceil - 2\sqrt{2} \log N + 6\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right]. \tag{A.16}
 \end{aligned}$$

Portanto, a complexidade (Equação A.4) para o caso bidimensional da técnica paralela torna-se a Equação A.17:

$$\begin{aligned}
 T_P(N) &\leq \frac{N}{P} \log \left( \frac{N}{P} \right) + \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \log \left( \frac{\sqrt{N}}{2^{\lfloor \frac{l}{2} \rfloor}} \right) \right] \\
 &\leq \frac{N}{P} \log \left( \frac{N}{P} \right) + \frac{\sqrt{N}}{2} \left[ 3 \log N - 8 + \frac{3\lceil \log P \rceil - 2\sqrt{2} \log N + 6\sqrt{2}}{\sqrt{2^{\lceil \log P \rceil}}} \right]. \tag{A.17}
 \end{aligned}$$

Uma vez que a complexidade está sendo analisada, as constantes não têm grande importância, e algumas delas podem ser descartadas. Além disso, para valores de  $P$  que são potência de 2, isto é, o

número de processos sempre duplica,  $\lceil \log P \rceil = \log P$ , e a complexidade se torna como na Equação A.18:

$$\begin{aligned}
T_P(N) &\leq \frac{N}{P} \log \left( \frac{N}{P} \right) + \sqrt{N} \left[ \log N + \frac{\log P - \log N}{\sqrt{2^{\log P}}} \right] \\
&= \frac{N}{P} \log \left( \frac{N}{P} \right) + \sqrt{N} \left[ \log N + \frac{\log P - \log N}{\sqrt{P}} \right] \\
&= \frac{N}{P} \log \left( \frac{N}{P} \right) + \sqrt{N} \log N - \sqrt{N} \frac{\log N - \log P}{\sqrt{P}} \\
&= \frac{N}{P} \log \left( \frac{N}{P} \right) + \sqrt{N} \log N - \frac{\sqrt{N}}{\sqrt{P}} \log \left( \frac{N}{P} \right) \\
&= \left( \frac{N}{P} - \sqrt{\frac{N}{P}} \right) \log \left( \frac{N}{P} \right) + \sqrt{N} \log N.
\end{aligned} \tag{A.18}$$

### A.2.2 Caso tridimensional

Em três dimensões, como  $N$  é o número de elementos em um domínio volumétrico,  $\sqrt[3]{N}$  é o número de elementos em uma região linear, e o número de elementos em uma região planar é  $\sqrt{N^2}$ .

O modelo do cubo uniforme subdivide primeiro no eixo X, depois no eixo Y e, finalmente, no eixo Z, recomeçando o ciclo. Na passagem de nível do eixo X para o eixo Y, a interface é subdividida ao meio, e o mesmo acontece na passagem do eixo Y para o eixo Z. No recomeço do ciclo, da passagem do eixo Z para o eixo X, entretanto, o tamanho da interface permanece o mesmo. Assim, a estimativa de carga também depende do nível  $l$ , no valor de  $l - \lceil l/3 \rceil$ , e pode ser calculada como  $N_l = 4 \frac{\sqrt[3]{N^2}}{2^{l - \lceil l/3 \rceil}}$

Portanto, o tempo de execução pode ser representado com na Equação A.19:

$$\begin{aligned}
T_P(N) &= T \left( \frac{N}{P} \right) + \sum_{l=1}^{\lceil \log_2 P \rceil} T \left( 4 \frac{\sqrt[3]{N^2}}{2^{l - \lceil l/3 \rceil}} \right) \\
&\leq c_s \left( \frac{N}{P} \right)^{p_s} \log \left( \frac{N}{P} \right) + c_i \sum_{l=1}^{\lceil \log_2 P \rceil} \left[ \left( \frac{\sqrt[3]{N^2}}{2^{l - \lceil l/3 \rceil}} \right)^{p_i} \log \left( \frac{\sqrt[3]{N^2}}{2^{l - \lceil l/3 \rceil}} \right) \right].
\end{aligned} \tag{A.19}$$

Fazendo-se as mesmas contas da Seção A.2.1, desta vez para o caso tridimensional, considerando-se quando  $\lceil \log P \rceil$  for  $3k$ ,  $3k + 1$  ou  $3k + 2$ , unificando-se os casos e, finalmente, fazendo  $P$  ser sempre potência de 2, a complexidade torna-se a Equação A.20:

$$T_P(N) \leq \left( \frac{N}{P} - \sqrt[3]{\frac{N^2}{P^2}} \right) \log \left( \frac{N}{P} \right) + \sqrt[3]{N^2} \log N. \tag{A.20}$$

### A.3 Super-linearidade

Um algoritmo paralelo atinge super-linearidade quando  $T_P(N) \leq \frac{T(N)}{P}$ , o que significa que o algoritmo paralelo, quando executado com  $P$  processos, é pelo menos  $P$  vezes mais rápido que o algoritmo sequencial. Assim, um condição de super-linearidade para o caso bidimensional é apresentado na Equação

ção A.21:

$$\begin{aligned}
 T_P(N) &\leq \frac{T(N)}{P} \\
 \frac{N}{P} \log \left( \frac{N}{P} \right) + \sqrt{N} \log N - \frac{\sqrt{N}}{\sqrt{P}} \log \left( \frac{N}{P} \right) &\leq \frac{N \log N}{P} \\
 \frac{N}{P} \log N - \frac{N}{P} \log P + \sqrt{N} \log N - \frac{\sqrt{N}}{\sqrt{P}} \log N + \frac{\sqrt{N}}{\sqrt{P}} \log P &\leq \frac{N \log N}{P} \\
 -\frac{N}{P} \log P + \sqrt{N} \log N - \frac{\sqrt{N}}{\sqrt{P}} \log N + \frac{\sqrt{N}}{\sqrt{P}} \log P &\leq 0 \\
 \sqrt{N} \frac{\log P}{\sqrt{P}} \left( 1 - \frac{\sqrt{N}}{\sqrt{P}} \right) + \sqrt{N} \log N \left( 1 - \frac{1}{\sqrt{P}} \right) &\leq 0 \\
 \frac{\log P}{\sqrt{P}} \left( \frac{\sqrt{P} - \sqrt{N}}{\sqrt{P}} \right) + \log N \left( \frac{\sqrt{P} - 1}{\sqrt{P}} \right) &\leq 0 \\
 \frac{\log P}{\sqrt{P}} (\sqrt{P} - \sqrt{N}) + \log N (\sqrt{P} - 1) &\leq 0 \\
 \log N (\sqrt{P} - 1) &\leq \frac{\log P}{\sqrt{P}} (\sqrt{N} - \sqrt{P}) \\
 (\sqrt{P} - 1) \frac{\sqrt{P}}{\log P} &\leq \frac{\sqrt{N} - \sqrt{P}}{\log N}. \tag{A.21}
 \end{aligned}$$

Como  $\log N \in O(\sqrt{N})$ ,  $\log N \leq \sqrt{N}$  tem-se que  $\frac{1}{\sqrt{N}} \leq \frac{1}{\log N}$ . Assim, a Equação A.21 pode ser continuada como na Equação A.22:

$$\begin{aligned}
 T_P(N) &\leq \frac{T(N)}{P} \\
 (\sqrt{P} - 1) \frac{\sqrt{P}}{\log P} &\leq \frac{\sqrt{N} - \sqrt{P}}{\log N} \\
 (\sqrt{P} - 1) \frac{\sqrt{P}}{\log P} &\leq \frac{\sqrt{N} - \sqrt{P}}{\sqrt{N}} \leq \frac{\sqrt{N} - \sqrt{P}}{\log N} \\
 (\sqrt{P} - 1) \frac{\sqrt{P}}{\log P} &\leq 1 - \frac{\sqrt{P}}{\sqrt{N}} \\
 \frac{\sqrt{P}}{\sqrt{N}} &\leq 1 - (\sqrt{P} - 1) \frac{\sqrt{P}}{\log P} \\
 \frac{\sqrt{P}}{\sqrt{N}} &\leq \frac{\log P - P + \sqrt{P}}{\log P} \\
 \frac{\sqrt{P} \log P}{\log P - P + \sqrt{P}} &\leq \sqrt{N} \\
 \frac{P \log^2 P}{(\log P - P + \sqrt{P})^2} &\leq N. \tag{A.22}
 \end{aligned}$$

A Equação A.23 mostra uma condição de super-linearidade para o caso 3D:

$$\frac{P \log^3 P}{(\log P - P + \sqrt[3]{P^2})^3} \leq N. \tag{A.23}$$

#### A.4 Escalabilidade

É conhecido que programas paralelos tendem a degradar quando o número de processos cresce muito (GRAMA et al., 2003), geralmente devido ao tempo de comunicação, que aumenta, enquanto que o tempo

de computação não diminui de maneira compensatória. No caso da técnica descrita neste trabalho, outra condição para que o tempo de execução aumente é quando o tempo de execução nas interfaces é muito grande relativamente ao tempo de execução nos subdomínios.

É vantajoso, portanto, saber o número de processos para o qual o tempo de execução começa a aumentar em vez de diminuir, ou seja, o número de processos que faz com que a derivada do tempo de execução se torne zero. As Equações A.24 a A.27 mostram os cálculos para quando isso acontece, no caso bidimensional:

$$\begin{aligned}
 \frac{d}{dP} T_P(N) &= 0 \\
 \frac{d}{dP} \left[ \frac{N}{P} \log \frac{N}{P} + \sqrt{N} \log N - \frac{\sqrt{N}}{\sqrt{P}} \log \frac{N}{P} \right] &= 0 \\
 N \frac{d}{dP} \left[ \frac{1}{P} \log \frac{N}{P} \right] - \sqrt{N} \frac{d}{dP} \left[ \frac{1}{\sqrt{P}} \log \frac{N}{P} \right] &= 0.
 \end{aligned} \tag{A.24}$$

As duas derivadas da Equação A.24 podem ser melhor descritas nas Equações A.25 e A.26:

$$\begin{aligned}
 \frac{d}{dP} \left[ \frac{1}{P} \log \frac{N}{P} \right] &= \frac{d}{dP} [P^{-1}] \log \frac{N}{P} + \frac{1}{P} \frac{d}{dP} \left[ \log \frac{N}{P} \right] \\
 &= -1P^{-2} \log \frac{N}{P} + \frac{1}{P} \frac{1}{\frac{N}{P} \ln 2} \frac{d}{dP} \left[ \frac{N}{P} \right] \\
 &= \frac{-1}{P^2} \log \frac{N}{P} + \frac{1}{N \ln 2} N \frac{-1}{P^2} = -\frac{\log \frac{N}{P}}{P^2} - \frac{1}{P^2 \ln 2},
 \end{aligned} \tag{A.25}$$

$$\begin{aligned}
 \frac{d}{dP} \left[ \frac{1}{\sqrt{P}} \log \frac{N}{P} \right] &= \frac{d}{dP} [P^{-\frac{1}{2}}] \log \frac{N}{P} + \frac{1}{\sqrt{P}} \frac{d}{dP} \left[ \log \frac{N}{P} \right] \\
 &= \frac{-1}{2} \frac{1}{P\sqrt{P}} \log \frac{N}{P} + \frac{1}{\sqrt{P}} \frac{1}{\frac{N}{P} \ln 2} \frac{d}{dP} \left[ \frac{N}{P} \right] \\
 &= \frac{-1}{2P\sqrt{P}} \log \frac{N}{P} + \frac{1}{\frac{N}{\sqrt{P}} \ln 2} N \frac{-1}{P^2} = -\frac{\log \frac{N}{P}}{2P\sqrt{P}} - \frac{\sqrt{P}}{P^2 \ln 2}.
 \end{aligned} \tag{A.26}$$

Substituindo as Equações A.25 e A.26, a Equação A.24 torna-se a Equação A.27, que não é, de fato, muito útil, uma vez que ela não mostra um valor explícito para o número de processos  $P$  como uma função do tamanho da saída  $N$ :

$$\begin{aligned}
 N \frac{d}{dP} \left[ \frac{1}{P} \log \frac{N}{P} \right] - \sqrt{N} \frac{d}{dP} \left[ \frac{1}{\sqrt{P}} \log \frac{N}{P} \right] &= 0 \\
 N \left[ -\frac{\log \frac{N}{P}}{P^2} - \frac{1}{P^2 \ln 2} \right] - \sqrt{N} \left[ -\frac{\log \frac{N}{P}}{2P\sqrt{P}} - \frac{\sqrt{P}}{P^2 \ln 2} \right] &= 0 \\
 \sqrt{N} \left[ \frac{\log \frac{N}{P}}{2P\sqrt{P}} + \frac{\sqrt{P}}{P^2 \ln 2} \right] &= N \left[ \frac{\log \frac{N}{P}}{P^2} + \frac{1}{P^2 \ln 2} \right] \\
 \frac{\sqrt{P} \ln 2 \log \frac{N}{P} + 2\sqrt{P}}{2P^2 \ln 2} &= \sqrt{N} \frac{\ln 2 \log \frac{N}{P} + 1}{P^2 \ln 2} \\
 \frac{\sqrt{P}}{2} \left( \ln 2 \log \frac{N}{P} + 2 \right) &= \sqrt{N} \left( \ln 2 \log \frac{N}{P} + 1 \right) \\
 \frac{\ln 2 \log \frac{N}{P} + 2}{\ln 2 \log \frac{N}{P} + 1} &= \frac{2\sqrt{N}}{\sqrt{P}}.
 \end{aligned} \tag{A.27}$$

Analogamente, a Equação A.28 mostra uma condição de escalabilidade para o caso tridimensional:

$$\frac{\ln 2 \log \frac{N}{P} + 3}{\ln 2 \log \frac{N}{P} + 1} = \frac{3 \sqrt[3]{N}}{2 \sqrt[3]{P}}. \tag{A.28}$$

## A.5 Considerações finais

Embora essas equações apresentem condições de super-linearidade e escalabilidade, existem diversas constantes implicitamente envolvidas na função de complexidade do algoritmo serial e da proporcionalidade na estimativa de carga, que varia com o tamanho da malha gerada, da geometria e da discretização do modelo. Além disso, as equações não contabilizam porções sequenciais no algoritmo, como a geração e a classificação da *octree* de estimativa de carga, e nem o tempo de comunicação entre processos, presente na etapa da geração das malhas de interface, que também é uma função do tamanho da malha de interface e do número de processos. Uma vez que essas constantes são difíceis de se obter como função da entrada, essas equações e condições apresentadas nesta seção podem não ter utilidade prática.

Por último, as equações são para um algoritmo de avanço de fronteira que têm complexidade  $O(N \log N)$  no tamanho da saída, o que é o caso geral. Para o pior caso,  $O(N^2 \log N)$ , os cálculos das equações tornam-se muito mais problemáticos e tediosos, e estão omitidos neste trabalho.

## REFERÊNCIAS

- ANGEL, E. *Interactive Computer Graphics - A Top-Down Approach Using OpenGL*. 5th. ed. [S.l.]: Addison Wesley, 2008. Citado em 62.
- BATISTA, V. H. F.; MILLMAN, D. L.; PION, S.; SINGLER, J. Parallel geometric algorithms for multi-core computers. *Computational Geometry: Theory and Applications*, v. 43, n. 8, p. 663–677, 2010. Citado em 44 e 50.
- BERN, M. W.; EPPSTEIN, D.; TENG, S.-H. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, v. 9, n. 6, p. 517–532, 1999. Citado em 23, 36 e 50.
- BISWAS, R.; DAS, S. K.; HARVEY, D. J.; OLIKER, L. Parallel dynamic load balancing strategies for adaptive irregular applications. *Applied Mathematical Modelling*, v. 25, n. 2, p. 109–122, 2000. Citado em 30.
- BONET, J.; PERAIRE, J. An alternating digital tree (ADT) algorithm for 3d geometric searching and intersection problems. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, Ltd, v. 13, n. 1, p. 1–17, 1991. ISSN 1097-0207. Citado em 118.
- CAMATA, J. J.; COUTINHO, A. L. G. A. Parallel linear octree meshing with immersed surfaces. In: *Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing*. Petropolis, Rio de Janeiro, Brazil: IEEE Computer Society Press, 2010. p. 151–158. ISBN 978-1-4244-8287-0. ISSN 1550-6533. Citado em 45 e 50.
- CAMATA, J. J.; COUTINHO, A. L. G. A. Parallel implementation and performance analysis of a linear octree finite element mesh generation scheme. *Concurrency and Computation: Practice and Experience*, v. 25, n. 6, p. 826–842, 2013. ISSN 1532-0634. Citado em 45 e 50.
- CARVALHO, P. C. P.; FIGUEIREDO, L. H. de. *Introdução à Geometria Computacional*. 1st. ed. [S.l.]: 18º Colóquio Brasileiro de Matemática, IMPA - Instituto Nacional de Matemática Pura e Aplicada, 1991. Citado em 29 e 62.
- CAVALCANTE-NETO, J. B. *Simulação Auto-Adaptativa Baseada em Enumeração Espacial Recursiva de Modelos Bidimensionais de Elementos Finitos*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 1994. Citado em 52.
- CAVALCANTE-NETO, J. B. *Geração de Malha e Estimativa de Erro para Modelos Tridimensionais de Elementos Finitos com Trincas*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 1998. Citado em 52.
- CAVALCANTE-NETO, J. B.; MARTHA, L. F.; WAWRZYNEK, P. A.; INGRAFFEA, A. R. A back-tracking procedure for optimization of simplex meshes. *Communications in Numerical Methods in Engineering*, v. 21, n. 12, p. 711–722, 2005. ISSN 1069-8299. Citado em 56.
- CAVALCANTE-NETO, J. B.; WAWRZYNEK, P. A.; CARVALHO, M. T. M. de; MARTHA, L. F.; INGRAFFEA, A. R. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, v. 17, n. 1, p. 75–91, 2001. Citado em 23, 52 e 118.
- CHEN, J.; ZHAO, D.; HUANG, Z.; ZHENG, Y.; WANG, D. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, v. 92, n. 8, p. 671–693, 2012. ISSN 1097-0207. Citado em 12, 48, 50 e 51.
- CHERNIKOV, A. N.; CHRISOCHOIDES, N. P. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In: *Proceedings of the 14th International Meshing Roundtable*. San Diego, United States: Sandia National Laboratory, 2005. Citado em 12, 40, 44 e 50.
- CHERNIKOV, A. N.; CHRISOCHOIDES, N. P. Parallel guaranteed quality Delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing*, v. 28, n. 5, p. 1907–1926, 2006. Citado em 40 e 50.

- CHRISOCHOIDES, N. P. *A Survey of Parallel Mesh Generation Methods*. [S.l.], 2005. Citado em 29.
- CHRISOCHOIDES, N. P.; CHERNIKOV, A. N.; FEDOROV, A.; KOT, A.; LINARDAKIS, L.; FOTEINOS, P. Towards exascale parallel Delaunay mesh generation. In: *Proceedings of the 18th International Meshing Roundtable*. Salt Lake City, United States: Springer-Verlag, 2009. Citado em 43 e 50.
- CHRISOCHOIDES, N. P.; NAVE, D. Simultaneous mesh generation and partitioning for Delaunay meshes. *Mathematics and Computers in Simulation*, v. 54, n. 4-5, p. 321–339, 2000. Citado em 12, 37, 44 e 50.
- COUPEZ, T.; DIGONNET, H.; DUCLOUX, R. Parallel meshing and remeshing. *Applied Mathematical Modelling*, Elsevier Science Inc, v. 25, n. 2, p. 153–175, 2000. Citado em 12, 37 e 50.
- DE COUGNY, H. L.; SHEPHARD, M. S. Parallel volume meshing using face removals and hierarchical repartitioning. *Computer Methods in Applied Mechanics and Engineering*, v. 174, n. 3-4, p. 275–298, 1999. Citado em 12, 31, 35, 50 e 51.
- DE COUGNY, H. L.; SHEPHARD, M. S.; ÖZTURAN, C. Parallel three-dimensional mesh generation. *Computing Systems in Engineering*, v. 5, n. 4-6, p. 311–323, 1994. ISSN 0956-0521. 3rd National Symposium on Large-Scale Structural Analysis for High-Performance Computers and Workstations. Citado em 33, 50 e 113.
- DE COUGNY, H. L.; SHEPHARD, M. S.; ÖZTURAN, C. Parallel three-dimensional mesh generation on distributed memory MIMD computers. *Engineering with Computers*, Springer-Verlag, v. 12, n. 2, p. 94–106, 1996. ISSN 0177-0667. Citado em 12, 33, 50 e 51.
- ELGINDY, H. An optimal speed-up parallel algorithm for triangulating simplicial point sets in space. *International Journal of Parallel Programming*, v. 15, n. 5, p. 389–398, 1986. Citado em 32 e 50.
- Free Software Foundation, Inc. *GCC, The GNU Compiler Collection*. 2015. Disponível em: <<http://gcc.gnu.org>>. Citado em 78.
- FREITAS, M. O. *Geração em Paralelo de Malhas Triangulares por Avanço de Fronteira com Particionamento por Decomposição Espacial Recursiva*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2010. Citado em 23.
- FREITAS, M. O.; CAVALCANTE-NETO, J. B.; VIDAL, C. A.; MARTHA, L. F.; INGRAFFEA, A. R.; WAWRZYNEK, P. A. A parallel technique for two-dimensional mesh generation for arbitrary regions with cracks. In: *Proceedings of the 32th Iberian Latin American Congress on Computational Methods in Engineering*. Ouro Preto, Brazil: Associação Brasileira de Mecânica Computacional, 2011. p. 1–20. Citado em 23.
- FREITAS, M. O.; WAWRZYNEK, P. A.; CAVALCANTE-NETO, J. B.; VIDAL, C. A.; MARTHA, L. F.; INGRAFFEA, A. R. A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. *Advances in Engineering Software*, v. 59, p. 38–52, 2013. ISSN 0965-9978. Citado em 13, 23, 49, 50, 51 e 59.
- GLOBISH, G. PARMESH – A parallel mesh generator. *Parallel Computing*, v. 25, p. 509–524, 1995. Citado em 31, 34 e 50.
- GLUT, B.; JURCZYK, T. Domain decomposition techniques for parallel generation of tetrahedral meshes. In: BUBAK, M.; ALBADA, D. van; DONGARRA, J.; SLOOT, P. (Ed.). *Proceedings of the International Conference on Computational Science 2008*. [S.l.]: Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5101). p. 641–650. ISBN 978-3-540-69383-3. Citado em 12, 42, 43, 50 e 51.
- GOTTSCHALK, S.; LIN, M. C.; MANOCHA, D. OBB-Tree: A hierarchical structure for rapid interference detection. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New Orleans, Louisiana, United States: Association for Computing Machinery, 1996. p. 171–180. ISBN 0-89791-746-4. Citado em 113.
- GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V. *Introduction to Parallel Computing*. 2nd. ed. [S.l.]: Addison Wesley, 2003. Citado em 123.

- GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. New York, New York, United States: Association for Computing Machinery, 1984. p. 47–57. ISBN 0-89791-128-8. Citado em 56.
- HEARN, D.; BAKER, M. P. *Computer Graphics - C Version*. 2nd. ed. [S.l.]: Prentice Hall, 1996. Citado em 62.
- HJELLE, Ø.; DÆHLEN, M. *Triangulation and Applications - Mathematics and Visualizations*. 1st. ed. [S.l.]: Springer, 2006. Citado em 22, 27 e 28.
- HODGSON, D. C.; JIMACK, P. K. Efficient parallel generation of partitioned, unstructured meshes. *Advances in Engineering Software*, v. 27, n. 1-2, p. 59–70, 1996. Citado em 12, 34, 35 e 50.
- IKITS, M.; MAGALLON, M. *The OpenGL Extension Wrangler Library*. 2015. Disponível em: <<http://glew.sourceforge.net>>. Citado em 78.
- ITO, Y.; SHIH, A. M.; ERUKALA, A. K.; SONI, B. K.; CHERNIKOV, A. N.; CHRISOCHOIDES, N. P.; NAKAHASHI, K. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, v. 75, n. 5-6, p. 200–209, 2007. Citado em 12, 29, 42 e 50.
- IVANOV, E. G.; ANDRÄ, H.; KUDRYAVTSEV, A. N. Domain decomposition approach for automatic parallel generation of tetrahedral grids. *Computational Methods in Applied Mathematics*, v. 6, n. 2, p. 178–193, 2006. Citado em 40, 50 e 51.
- JIN, H.; TANNER, R. I. Generation of unstructured tetrahedral meshes by advancing front technique. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, Ltd, v. 36, n. 11, p. 1805–1823, 1993. ISSN 1097-0207. Citado em 27 e 118.
- KARYPIS, G.; KUMAR, V. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*. [S.l.], 1998. Citado em 30.
- KARYPIS, G.; SCHLOEGEL, K.; KUMAR, V. *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.2*. [S.l.], 2011. Citado em 30.
- KHAN, A. I.; TOPPING, B. H. V. Parallel adaptive mesh generation. *Computing Systems in Engineering*, v. 2, n. 1, p. 75–101, 1991. ISSN 0956-0521. Citado em 32 e 50.
- KILGARD, M. *GLUT - The OpenGL Utility Toolkit*. 2015. Disponível em: <<https://www.opengl.org/resources/libraries/glut>>. Citado em 78.
- KOHOUT, J.; KOLINGEROVÁ, I.; ŽÁRA, J. Parallel Delaunay triangulation in  $E^2$  and  $E^3$  for computers with shared memory. *Parallel Computing*, v. 31, n. 5, p. 491–522, 2005. Citado em 12, 39 e 50.
- LÄMMER, L.; BURGHARDT, M. Parallel generation of triangular and quadrilateral meshes. *Advances in Engineering Software*, v. 31, n. 12, p. 929–936, 2000. Citado em 12, 30, 36, 50 e 51.
- LARWOOD, B. G.; WEATHERILL, N. P.; HASSAN, O.; MORGAN, K. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, v. 58, p. 177–188, 2003. Citado em 12, 38, 39, 40, 50 e 51.
- LEWIS, R. W.; ZHENG, Y.; GETHIN, D. T. Three-dimensional unstructured mesh generation: Part 3 - volume meshes. *Computer Methods in Applied Mechanics and Engineering*, v. 134, p. 285–310, 1996. Citado em 92 e 93.
- LINARDAKIS, L.; CHRISOCHOIDES, N. P. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing*, SIAM - Society for Industrial and Applied Mathematics, v. 27, n. 4, p. 1394–1423, 2006. Citado em 12, 41, 43 e 50.
- LO, S. H. Parallel Delaunay triangulation - Application to two dimensions. *Finite Elements in Analysis and Design*, v. 62, n. 0, p. 37–48, 2012. ISSN 0168-874X. Citado em 12, 46, 47 e 50.
- LO, S. H. Parallel Delaunay triangulation in three dimensions. *Computer Methods in Applied Mechanics and Engineering*, v. 237–240, n. 0, p. 88–106, 2012. ISSN 0045-7825. Citado em 47 e 50.

- LÖHNER, R. A parallel advancing front grid generation scheme. *International Journal for Numerical Methods in Engineering*, v. 51, n. 6, p. 663–678, 2001. Citado em 12, 23, 29, 31, 38, 50 e 51.
- LÖHNER, R. A 2nd generation parallel advancing front grid generator. In: *Proceedings of the 21st International Meshing Roundtable*. San Jose, California, United States: Sandia National Laboratory, 2012. p. 457–474. Citado em 12, 47, 48, 50 e 51.
- LÖHNER, R. Recent advances in parallel advancing front grid generation. *Archives of Computational Methods in Engineering*, Springer Netherlands, v. 21, n. 2, p. 127–140, 2014. ISSN 1134-3060. Citado em 47, 50 e 51.
- LÖHNER, R.; PARIKH, P. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, v. 8, p. 1135–1149, 1988. Citado em 118.
- MERKS, E. An optimal parallel algorithm for triangulating a set of points in the plane. *International Journal of Parallel Programming*, v. 15, n. 5, p. 399–411, 1986. Citado em 32 e 50.
- MIRANDA, A. C. de O.; CAVALCANTE-NETO, J. B.; MARTHA, L. F. An algorithm for two-dimensional mesh generation for arbitrary regions with cracks. In: *SIBGRAPI '99: Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing*. [S.l.]: IEEE Computer Society, 1999. p. 29–38. Citado em 23 e 52.
- MIRANDA, A. C. de O.; LIRA, W. W. M.; NETO, J. B. C.; SOUSA, R. A.; MARTHA, L. F. A three-dimensional adaptive mesh generation approach using geometric modeling with multi-regions and parametric surfaces. *Journal of Computing and Information Science in Engineering*, American Society of Mechanical Engineers, v. 13, n. 2, p. 021002–1–021002–13, 2013. ISSN 1530-9827. Citado em 61.
- MÖLLER, P.; HANSBO, P. On advancing front mesh generation in three dimensions. *International Journal for Numerical Methods in Engineering*, John Wiley & Sons, Ltd, v. 38, n. 21, p. 3551–3568, 1995. ISSN 1097-0207. Citado em 27 e 118.
- MORETTI, C. O. *Um Sistema Computacional Paralelo Aplicado à Simulação de Propagação Tridimensional de Fraturas*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2001. Citado em 30, 38 e 50.
- MPI Forum. *The Message Passing Interface (MPI) Standard*. 2015. Disponível em: <<http://www.mcs.anl.gov/research/projects/mpi>>. Citado em 78.
- OKUSANYA, T.; PERAIRE, J. Parallel unstructured mesh generation. In: *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*. [S.l.]: Mississippi State University, 1996. p. 719–729. Citado em 12, 34 e 50.
- OpenCV Dev Team. *Open Source Computer Vision*. 2015. Disponível em: <<http://opencv.org>>. Citado em 78.
- OpenMP Architecture Review Board. *The OpenMP API Specification for Parallel Programming*. 2015. Disponível em: <<http://www.openmp.org>>. Citado em 78.
- PANITANARAK, T.; SHONTZ, S. M. MDEC: MeTiS-based domain decomposition for parallel 2D mesh generation. *Procedia Computer Science*, v. 4, n. 0, p. 302–311, 2011. ISSN 1877-0509. Proceedings of the International Conference on Computational Science 2011. Citado em 12, 45, 46 e 50.
- PARTHASARATHY, V. N.; GRAICHEN, C. M.; HATHAWAY, A. F. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design*, v. 15, p. 255–261, 1993. Citado em 92 e 93.
- PÉBAY, P. P.; STEPHENSON, M. B.; FORTIER, L. A.; OWEN, S. J.; MELANDER, D. J. pCAMAL: An embarrassingly parallel hexahedral mesh generator. In: *Proceedings of the 16th International Meshing Roundtable*. Washington, United States: Sandia National Laboratory, 2007. Citado em 42 e 50.
- PIRZADEH, S. Z.; ZAGARIS, G. Domain decomposition by the advancing-partition method for parallel unstructured grid generation. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. [S.l.]: AIAA - American Institute of Aeronautics and Astronautics, 2009. Citado em 43.

- RIVARA, M.-C.; CALDERON, C.; FEDOROV, A.; CHRISOCHOIDES, N. P. Parallel decoupled terminal-edge bisection method for 3D mesh generation. *Engineering with Computers*, v. 22, p. 111–119, 2006. Citado em 12, 41, 42 e 50.
- RUPPERT, J. A new and simple algorithm for quality 2-dimensional mesh generation. In: *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*. Texas, United States: SIAM - Society for Industrial and Applied Mathematics, 1999. p. 83–62. Citado em 28.
- SHEWCHUK, J. R. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, v. 18, n. 3, p. 305–363, 1997. Citado em 114.
- SIMONOVSKI, I.; CIZELJ, L. Automatic parallel generation of finite element meshes for complex spatial structures. *Computational Materials Science*, v. 50, n. 5, p. 1606–1618, 2011. ISSN 0927-0256. Citado em 45 e 50.
- TENG, Y. A.; SULLIVAN, F.; BEICHL, I.; PUPPO, E. A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation. In: *Proceedings of the 1993 Conference on High Performance Network and Computing*. Oregon, United States: Association for Computing Machinery, 1993. p. 112–121. Citado em 32 e 50.
- The C++ Standards Committee. *JTC1/SC22/WG21 - The C++ Standards Committee - ISO C++*. 2015. Disponível em: <<http://www.open-std.org/jtc1/sc22/wg21>>. Citado em 78.
- The Ohio State University. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE*. 2015. Disponível em: <<http://mvapich.cse.ohio-state.edu>>. Citado em 78.
- The OpenGL Architecture Review Board. *OpenGL - The Industry's Foundation for High Performance Graphics*. 2015. Disponível em: <<http://www.opengl.org>>. Citado em 78.
- The University of Texas at Austin. *Stampede - Texas Advanced Computing Center*. 2015. Disponível em: <<https://www.tacc.utexas.edu/stampede>>. Citado em 78.
- TOPPING, B. H. V.; CHENG, B. Parallel and distributed adaptive quadrilateral mesh generation. *Computers & Structures*, v. 73, n. 1-5, p. 519–536, 1999. Citado em 12, 36 e 50.
- TOPPING, B. H. V.; KHAN, A. I. Subdomain generation for non-convex parallel finite element domains. *Advances in Engineering Software*, v. 25, n. 2-3, p. 253–266, 1996. ISSN 0965-9978. Citado em 30.
- WARNER, D.; INGRAFFEA, A. *Cornell Fracture Group*. 2015. Disponível em: <<http://www.cfg.cornell.edu>>. Citado em 78.
- WILSON, J. K.; TOPPING, B. H. V. Parallel adaptive tetrahedral mesh generation by the advancing front technique. *Computers & Structures*, v. 68, n. 1-3, p. 57–78, 1998. Citado em 29, 35 e 50.
- WU, H.; GUAN, X.; GONG, J. ParaStream: A parallel streaming Delaunay triangulation algorithm for LiDAR points on multicore architectures. *Computers & Geosciences*, Elsevier Ltd, v. 37, n. 9, p. 1355–1363, 2011. ISSN 0098-3004. Citado em 12, 45, 46, 50 e 51.
- WU, P.; HOUSTIS, E. N. Parallel adaptive mesh generation and decomposition. *Engineering with Computers*, Springer London, v. 12, n. 3-4, p. 155–167, 1996. Citado em 34, 50 e 51.
- wxTeam. *wxWidgets - Cross-Platform GUI Library*. 2015. Disponível em: <<http://www.wxwidgets.org>>. Citado em 78.
- YILMAZ, Y.; ÖZTURAN, C.; TOSUN, O.; ÖZER, A. H.; SONER, S. *Parallel Mesh Generation, Migration and Partitioning for the Elmer Application*. [S.l.], 2010. 1-12 p. Citado em 12, 44, 50 e 51.
- ZAGARIS, G.; PIRZADEH, S. Z.; CHRISOCHOIDES, N. P. A framework for parallel unstructured grid generation for practical aerodynamic simulations. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. [S.l.]: AIAA - American Institute of Aeronautics and Astronautics, 2009. Citado em 12, 31, 43, 50 e 51.