Markos Oliveira Freitas

Geração em Paralelo de Malhas Triangulares por Avanço de Fronteira com Particionamento por Decomposição Espacial Recursiva

Fortaleza

2010

Markos Oliveira Freitas

Geração em Paralelo de Malhas Triangulares por Avanço de Fronteira com Particionamento por Decomposição Espacial Recursiva

Dissertação apresentada ao Departamento de Computação da Universidade Federal do Ceará como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador:
Joaquim Bento Cavalcante Neto

Coorientador:
Creto Augusto Vidal

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO
CRAB - COMPUTAÇÃO GRÁFICA, REALIDADE VIRTUAL E ANIMAÇÃO

Fortaleza

2010

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 namorada, Karol, pelo apoio e carinho durante esses anos.

Agradeço aos professores orientadores, Joaquim Bento e Creto Vidal, por terem acreditado no meu trabalho e me guiado na minha jornada acadêmica. Ao professor Luiz Fernando Martha, pelas observações e contribuições que engrandeceram este trabalho.

Aos colegas de labuta Marco, Ricardo, André Ribeiro, Yuri Lenon, Roberto, Teófilo, Martha, Rafael Ivo, Jonas, Caio, Suzana, Rubens, Rafael Siqueira, Charles, Eduardo, Pablo, Felipe, Francicléber, André Ramos, Airton, e aos outros nomes que tenha esquecido, pelos momentos divertidos durante as intermináveis horas enfurnadas 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, Leonardo Frota, Leonardo Tavares, Ingryd, Vandick e Gabriela, pelas amizades que se formaram e perduram até hoje. Agradeço ainda aos amigos presentes à minha defesa, entre eles Patrícia, Gisele, Lucas, Ben Rainir, João Neto, Laise, Lilian e Iuri.

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. Ao Laboratório de Pesquisa em Computação (LIA) e ao Grupo de Pesquisa em Paralelismo, Grafos e Algoritmos (ParGO), por terem cedido o *cluster* para a realização dos testes necessários. Agradeço ao pessoal da administração do LIA, Daniel, Fernando, Débora, e, em especial, a Daniel Matos, que, mesmo durante as férias, mostrou-se disposto a ajudar sempre que preciso.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio 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 bidimensionais triangulares utilizando computação paralela, com processadores de memória compartilhada, memória distribuída ou memória híbrida. Inicialmente, o domínio dado como entrada, ou seja, a borda de um objeto juntamente com o seu interior, é subdividido recursivamente, utilizando uma *quadtree*. As células-folha dessa *quadtree* são distribuídas entre os diversos processadores, que ficam responsáveis pela geração dos triângulos nessas regiões, bem como a otimização dessas submalhas. Posteriormente, a nova fronteira é encontrada e cada célula da *quadtree* é deslocada de metade de seu tamanho em um eixo cartesiano. Então, a nova fronteira é particionada entre as células deslocadas, que são novamente distribuídas entre os processadores, onde mais submalhas são geradas. O deslocamento da *quadtree* é repetido em várias direções e sentidos, executando alternadamente com a geração das submalhas, até que todas as células tenham sido deslocadas em todas as direções e não seja mais possível gerar malha. Então, um processador fica responsável por finalizar a geração da malha e otimizar a parte da malha que não foi otimizada pelos outros processadores. Esta técnica mostrou-se mais rápida que a técnica sequencial, além de ter mantido a qualidade da malha próxima da malha gerada sequencialmente.

PALAVRAS-CHAVE: Geração em paralelo de malhas, triangulação, geometria computacional, computação de alto desempenho.

Abstract

This work describes a technique for generating two-dimensional triangular meshes using parallel computing, with shared, distributed or hybrid memory. Initially, the input domain, i.e., the boundary of an object and its interior, is recursively subdivided, using a quadtree. This quadtree's leaf cells are distributed among the processors, that are responsible for generating the triangles, as well as optimizing their submeshes. After that, the new front is found and each quadtree cell is shifted by half its size in the direction of a Cartesian axis. Then, the new front is partitioned among the shifted cells, which are again distributed among the processors. The quadtree shifting is repeated in other directions, executing alternately with the submeshes generation, until every cell has been shifted in all directions and no more mesh can be generated. Then, one processor is responsible for finalizing the mesh generation and improving the part of the mesh that was not improved by the other processors. This technique is much faster than its serial counterpart and produces meshes with the same quality as the ones generated by the serial version.

KEYWORDS: *Parallel mesh generation, triangulation, computational geometry, high performance computing.*

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

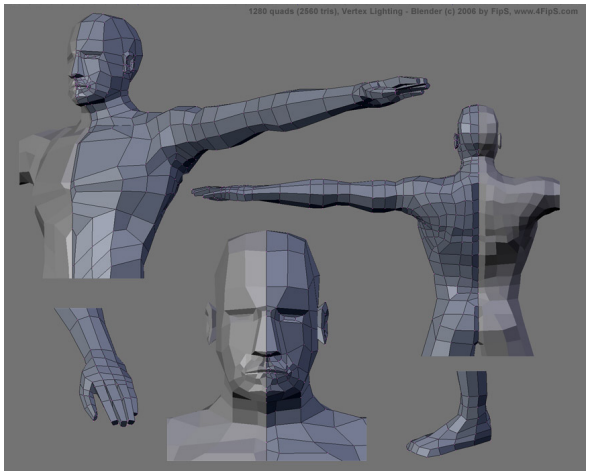
1.1 Motivação

Malhas e, em particular, triangulações, são amplamente utilizadas como uma base para representar geometrias e outras informações que aparecem em uma larga variedade de aplicações [Hjelle e Dæhlen 2006]. Podem ser utilizadas em:

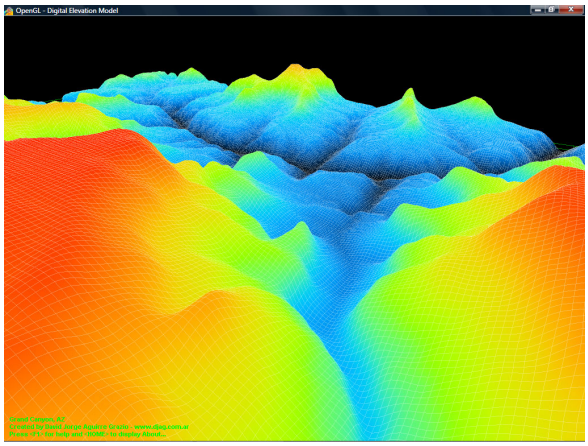
- Computação gráfica (CG) e visualização, modelando avatares, objetos e ambientes tridimensionais para aplicações de realidade virtual (Figura 1.1a).
- Sistemas de informações geográficas (SIG, ou GIS, do inglês *geographical information systems*), representando terrenos e relações entre objetos geográficos (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 de elementos finitos (MEF, ou *finite element method*, FEM).

A fim de representar mais fielmente os objetos reais, é essencial a utilização de malhas maiores, ou seja, com mais polígonos ou poliedros, permitindo que mais detalhes sejam vistos ou analisados. Com isso, a geração manual dessas malhas, além de demorada, exige uma certa capacidade artística de quem estiver modelando. Portanto, a geração automática dessas malhas tornou-se necessária.

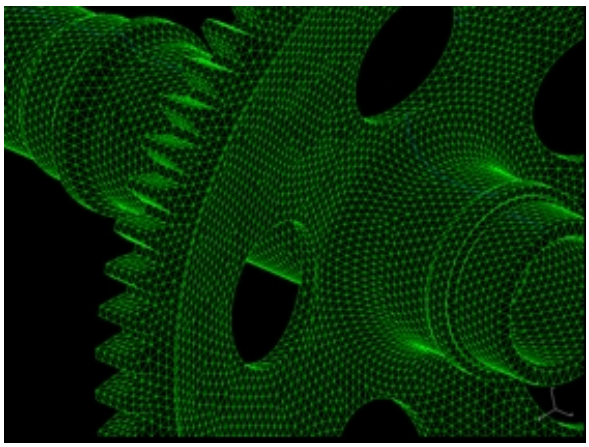
Assim, com o uso de malhas geradas automaticamente, tornou-se possível, por exemplo, simular computacionalmente fenômenos físicos com maior precisão. Entretanto, a utilização de uma única unidade de processamento pode fazer com que essas simulações demorem muito. Logo, com o propósito de diminuir o tempo de execução das simulações, passou-se a utilizar



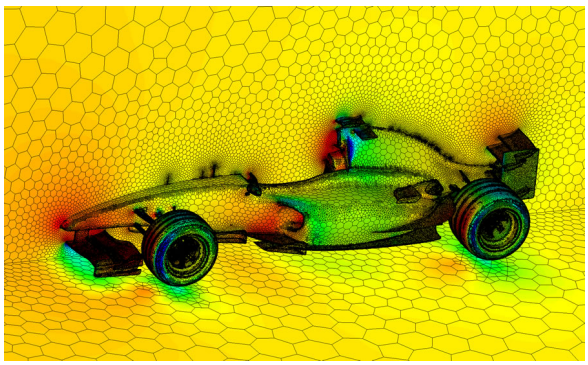
(a) Malha de computação gráfica (modelo humano). Fonte: `www.blenderartists.org`.



(b) Malha de sistemas de informação geográfica (*Grand Canyon*, EUA). Fonte: `www.mcrenox.com.ar`.



(c) Malha de projetos assistidos por computadores (engrenagem). Fonte: `www.stephensondesign.com`.



(d) Malha de engenharia (carro de Fórmula 1). Fonte: `www.ansys.com`.

Figura 1.1: Exemplos de uso de malhas.

técnicas de computação de alto desempenho, ou seja, a utilização de diversos processadores trabalhando conjuntamente para resolver uma instância de um problema.

Entretanto, em algumas simulações computacionais, 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 triangulares. Essa técnica deve satisfazer quatro requisitos:

1. Respeitar a fronteira de um objeto (discretizada em segmentos), dada como entrada.
2. Gerar malha de boa qualidade, a fim de prover resultados precisos em simulações de fenômenos físicos.
3. Fornecer uma boa transição entre regiões muito refinadas, com muitos triângulos pequenos, e regiões pouco refinadas, com poucos triângulos grandes.
4. Tratar fronteiras com trincas, ou fraturas, e gerar malhas respeitando-as, para que a malha seja utilizada em simulações de propagação de fraturas.

A técnica paralela tem como base uma técnica sequencial existente, que satisfaz os quatro critérios acima relacionados [Miranda et al. 1999, Cavalcante-Neto et al. 2001]. Além disso, a técnica paralela pode ser utilizada tanto em computadores de memória compartilhada quanto em computadores de memória distribuída, como em computadores que combinam os dois tipos de memória. Essa técnica também é suficientemente genérica para poder ser estendida para três dimensões.

A ideia por trás da técnica paralela é gerar uma decomposição espacial recursiva, ou seja, uma *quadtree*, que particione o domínio. Cada célula-folha é uma partição, cuja malha interna pode ser gerada de forma independente das outras partições. Assim, as malhas internas podem ser geradas e melhoradas simultaneamente. Para fazer a fronteira avançar de uma partição para outra, cada célula sofre um deslocamento de metade do seu tamanho em uma direção cartesiana, permitindo a geração de mais malha. Depois, cada célula volta para sua posição original, podendo gerar mais malha, para então a célula ser deslocada em outra direção.

Esse procedimento de deslocamento em uma direção e geração de malha acontece alternadamente, até que todas as células tenham se deslocado em todas as direções e não seja mais possível gerar malha. Então, um único processador fica responsável pela finalização da malha e pela melhoria da parte da malha que não foi previamente melhorada. Assim, uma malha para aquele domínio é completamente gerada.

1.3 Organização

Este trabalho está organizado em seis capítulos. O Capítulo 2 apresenta os principais conceitos de Geometria Computacional e de Computação de Alto Desempenho, necessários à compreensão dos capítulos restantes. O Capítulo 3 relata algumas técnicas existentes de geração em paralelo de malhas.

O Capítulo 4 apresenta a técnica paralela de geração de malhas proposta. Este capítulo explica, inicialmente, a técnica sequencial em que se baseou a técnica paralela. Posteriormente, a técnica proposta é apresentada em detalhes. Isso é importante, porque a técnica paralela faz uso da sequencial em cada processador.

O Capítulo 5 traz alguns resultados obtidos com a utilização da técnica proposta, mostrando que a técnica de fato gera malhas com qualidade em tempo reduzido, quando comparado com a técnica sequencial. No Capítulo 6, são apresentadas as conclusões a cerca do trabalho e algumas sugestões de trabalhos futuros.

2 Conceitos e Definições

2.1 Introdução

Este capítulo visa fornecer ao leitor alguns conceitos referentes às áreas de pesquisa envolvidas neste trabalho. Na Seção 2.2, são apresentados os conceitos sobre geometria computacional e malhas bem como sobre computação de alto desempenho, além de fazer relações entre malhas e paralelismo. Na Seção 2.3, são descritas algumas técnicas de geração de malha, como avanço de fronteira e triangulação de Delaunay, e alguns métodos utilizados para paralelizá-las. A última seção, 2.4, traz as considerações finais do capítulo.

2.2 Conceitos

2.2.1 Geometria Computacional

A geometria computacional é a área da computação que estuda estruturas de dados e algoritmos corretos e ótimos para a solução de problemas geométricos. Entre os problemas estudados, estão a construção de fechos convexos e a geração de malhas, descritos nesta Seção. Muitas das definições e resultados apresentados foram adaptados de [Carvalho e Figueiredo 1991], [Frey e George 2000] e [Owen 1998].

Fecho Convexo

Definição 2.1 *Seja d um número inteiro não-negativo. Sejam* $C \subset \mathbb{R}^d$ *e* $p \in \mathbb{R}^d$*. Então, p é dito ser combinação convexa dos pontos de C se*

$$p = \sum_{i=1}^{|C|} p_i \lambda_i, \; com \; \sum_{i=1}^{|C|} \lambda_i = 1.$$

Definição 2.2 *Seja d um número inteiro não-negativo. Então, $C \subseteq \mathbb{R}^d$ é dito convexo se, para quaisquer dois pontos p_1 e p_2 de C e para $0 \leq \lambda \leq 1$, $\lambda \in \mathbb{R}$, o ponto $p = \lambda p_1 + (1-\lambda)p_2 \in C$. Isto é, se combinações convexas de elementos de C também pertencem a C.*

Definição 2.3 *Um ponto $p \in C \subseteq \mathbb{R}^d$ é chamado de ponto extremo de C se C é um conjunto convexo e p não pode ser expresso como uma combinação convexa dos outros pontos de C.*

Definição 2.4 *Seja $C \subset \mathbb{R}^d$ finito, com d um inteiro não-negativo. Então, o fecho convexo de C é o conjunto de todas as combinações convexas dos elementos de C.*

O fecho convexo de um conjunto finito de pontos é inteiramente caracterizado pelos seus pontos extremos, uma vez que todos os pontos internos podem ser escritos como combinações convexas destes. Dessa forma, podemos pensar no fecho convexo de um conjunto finito de pontos como o menor objeto geométrico convexo que engloba todos os pontos, onde menor diz respeito ao tamanho da região interna desse objeto (Figura 2.1).

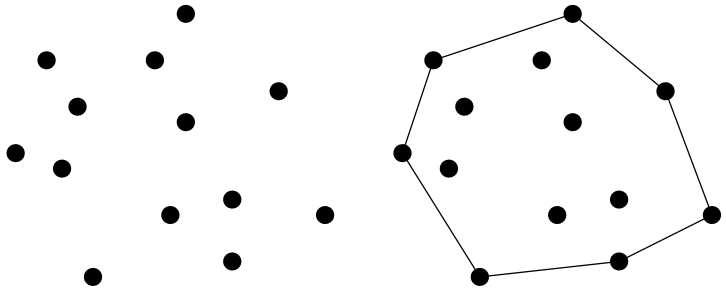


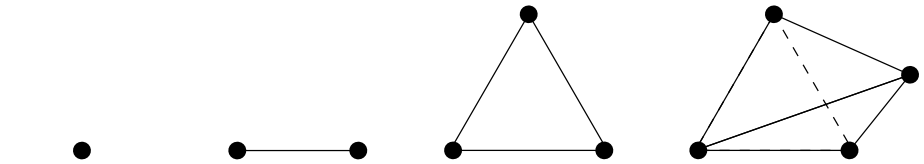
Figura 2.1: Conjunto de pontos e seu fecho convexo.

Complexos Simpliciais

Definição 2.5 *Seja d um número inteiro não-negativo. Considere $d+1$ pontos $p_j = (p_{i,j})_{i=1}^d \in \mathbb{R}^d$, $1 \leq j \leq d+1$, nem todos no mesmo hiperplano, isto é, tais que a matriz*

$$A = \begin{pmatrix} p_{1,1} & \cdots & p_{1,d+1} \\ \cdots & \cdots & \cdots \\ p_{d,1} & \cdots & p_{d,d+1} \\ 1 & 1 & 1 \end{pmatrix}$$

de ordem $d+1$ é invertível. Então, um d-simplex K, cujos vértices são os p_j, é o fecho convexo desses pontos. Todo ponto $p \in \mathbb{R}^d$ pode ser escrito como uma combinação dos $d+1$ pontos p_j, e onde os λ_i são únicos para p e são chamados de coordenadas baricêntricas do ponto p com respeito aos pontos p_j.



(a) 0-*simplex*. (b) 1-*simplex*. (c) 2-*simplex*. (d) 3-*simplex*.

Figura 2.2: Exemplos de *simplices*.

Informalmente, um *d-simplex* é o menor objeto geométrico que pode ser descrito em d dimensões, mas não em dimensões menores. Assim, um ponto é um 0-*simplex* (Figura 2.2a), um segmento de reta é um *simplex* de dimensão 1 (Figura 2.2b), um triângulo é um *simplex* bidimensional (Figura 2.2c), e um tetraedro é um *simplex* em três dimensões (Figura 2.2d).

Observe ainda que a combinação a que se refere a Definição 2.5 não necessariamente é convexa (Definição 2.1), ou seja, $\sum_{i=1}^{d} \lambda_i$ pode ser diferente de 1.

O plural de *simplex* aqui utilizado pode ser tanto *simplexes* quanto *simplices*, palavras emprestadas da língua inglesa. Em geral, são considerados mais usualmente *simplices* até de três dimensões, por que acima disso não fazem sentido em computação gráfica e não têm usabilidade comum em engenharia.

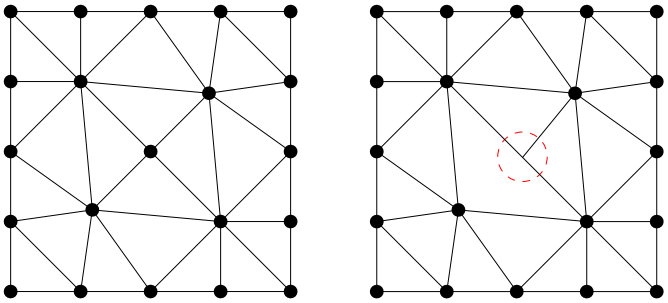
Definição 2.6 *Seja $S \subset \mathbb{R}^d$, $d \in \mathbb{N}$, um conjunto finito de pontos. Então, conv(S), o fecho convexo de S, denota um domínio Ω em $\mathbb{R}^d$. Então, um complexo simplicial C de Ω é um conjunto de simplices tal que:*

1. *O conjunto de 0-simplices em C é o conjunto de pontos S,*
2. *$\Omega = \bigcup_{K \in C} K$, onde K é um d-simplex,*
3. *O interior de cada elemento K em C é não vazio,*
4. *A interseção do interior de dois d-simplices de C é vazia.*

Em duas dimensões, um complexo simplicial é chamado de triangulação, e em três dimensões, de tetraedralização. Algumas vezes, complexos simpliciais em mais de duas dimensões também são chamadas de triangulação, para efeito de generalização.

É comum, ainda, relaxar-se a definição de complexo simplicial para permitir a existência de novos pontos, além daqueles dados em S. Tais pontos são chamados de pontos de Steiner (por conta do matemático suíço Jakob Steiner), e são frequentemente utilizados em triangulações e tetraedralizações.

Definição 2.7 *Seja C um complexo simplicial de dimensão d. Então, C é chamado de complexo simplicial conforme (ou conformante) se a interseção de quaisquer dois simplices de C é vazia ou é exatamente um simplex de dimensão menor que d.*



(a) Conforme. (b) Não-conforme.

Figura 2.3: Triangulações conforme e não-conforme.

Geralmente, triangulações e tetraedralizações são conformes (Figura 2.3a). Quando um complexo simplicial não é conforme, ele é chamado de não-conforme (Figura 2.3b). Perceba, na Figura 2.3b, que a região destacada indica a não-conformidade, onde a interseção entre dois triângulos não corresponde a uma aresta ou a um vértice, mas a uma parte de uma aresta.

Definição 2.8 *Sejam C um complexo simplicial para $S \subset \mathbb{R}^d$ e R um conjunto de simplices de dimensão menor que d tais que todos os* 0*-simplices de R pertencem a S. Então, C é dito restrito com relação a R, ou simplesmente restrito, se todos os simplices de R estão presentes em C. R é chamado de conjunto de restrições do complexo simplicial.*

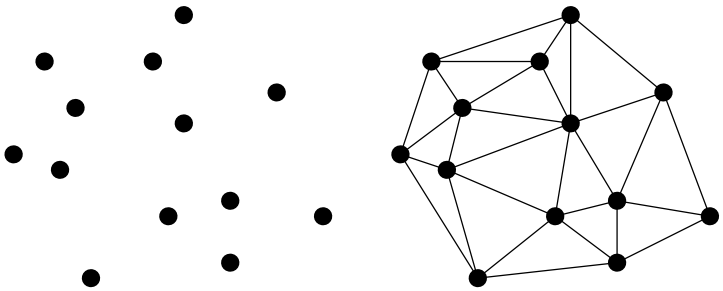
Geralmente, os complexos simpliciais restritos (Figura 2.4b) são também conformes. Em duas dimensões, as restrições são segmentos de retas (arestas) e, em três dimensões, são arestas e triângulos. Essas restrições podem ser interiores ou de borda (Figura 2.4c). No caso das restrições de borda, não podem existir *simplices* fora da região definida por elas, e o complexo simplicial não mais diz respeito ao fecho convexo do conjunto de pontos.

Malhas

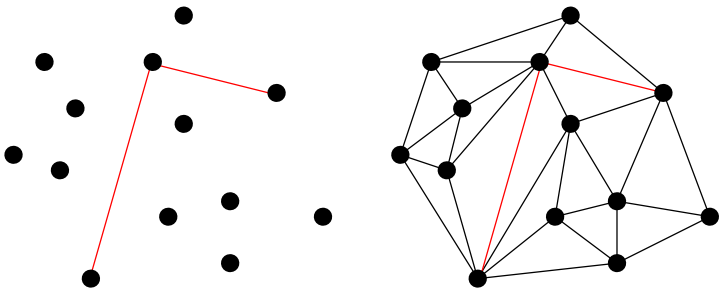
Definição 2.9 *Seja Ω um domínio limitado (por exemplo, pelo seu fecho convexo). Então, M é uma malha de Ω se:*

1. $\Omega = \bigcup_{K \in M} K$,
2. *O interior de cada elemento K em M é não vazio,*

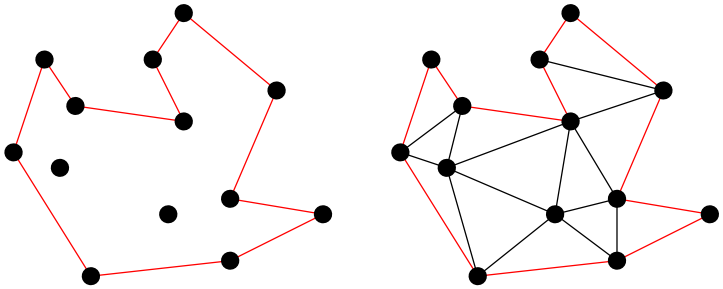
3. A interseção do interior de dois elementos de M é vazia.



(a) Sem restrições.



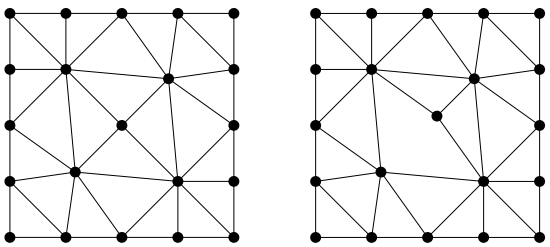
(b) Com restrições internas (arestas vermelhas).



(c) Com restrições de borda (arestas vermelhas).

Figura 2.4: Triangulações sem e com restrições.

Perceba que, em uma malha, os elementos podem ser objetos geométricos quaisquer, não mais restritos a *simplices* (Figura 2.5).



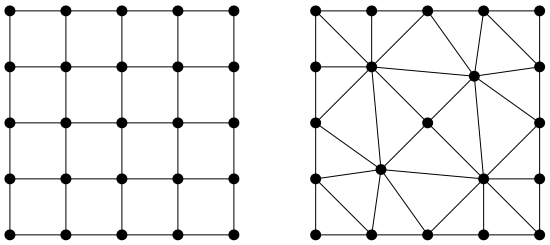
(a) Triangulação. (b) Malha qualquer.

Figura 2.5: Triangulação e malha qualquer.

Definição 2.10 *Seja M uma malha de dimensão d. Então, M é chamado de malha conforme (ou conformante) se a interseção de quaisquer dois elementos de M é vazia ou é exatamente um objeto geométrico de dimensão menor que d.*

Geralmente, as malhas utilizadas em aplicações reais são conformes. Podemos, ainda, definir malhas restritas, analogamente a complexos simpliciais restritos. Obviamente, triangulações, tetraedralizações e complexos simpliciais mais gerais também são malhas.

Definição 2.11 *Uma malha é chamada de estruturada se todos os seus vértices internos têm o mesmo número de elementos adjacentes. Caso contrário, ela é chamada de não-estruturada.*



(a) Estruturada. (b) Não-estruturada.

Figura 2.6: Malhas estruturada e não-estruturada.

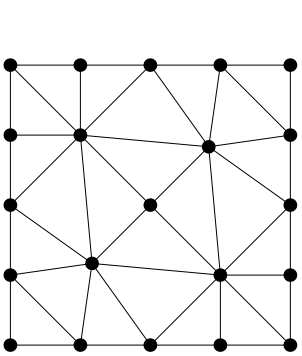
Como consequencia da Definição 2.11, em uma malha estruturada bidimensional (Figura 2.6a), todos os pontos têm a mesma quantidade de pontos adjacentes, ou seja, a mesma conectividade, exceto pelos pontos do contorno. Em uma malha não-estruturada (Figura 2.6b), um ponto interno da malha pode ter qualquer conectividade. Geralmente, utiliza-se quadriláteros e hexaedros para malhas estruturadas e triângulos e tetraedros para malhas não-estruturadas. Uma malha estruturada também pode ser chamada de grade, apesar de alguns autores usarem os termos grade e malha como sinônimos.

Definição 2.12 *Uma malha é chamada de híbrida se nem todos os seus elementos têm a mesma quantidade de vértices.*

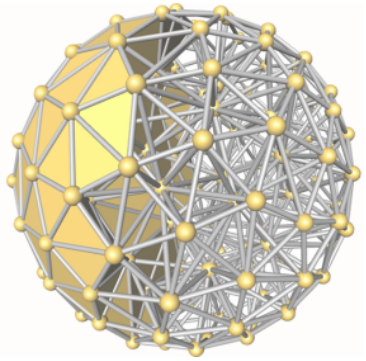
Em uma malha híbrida (Figura 2.5b), seus elementos podem ser quaisquer, desde que sejam de mesma dimensão. Em uma malha não-híbrida (Figura 2.5a), todos os seus elementos são de um mesmo tipo.

Além das malhas bidimensionais e tridimensionais, na prática, também são utilizadas malhas de superfícies. Tais malhas são compostas de elementos de uma determinada dimensão $\mathbb{R}^d$,

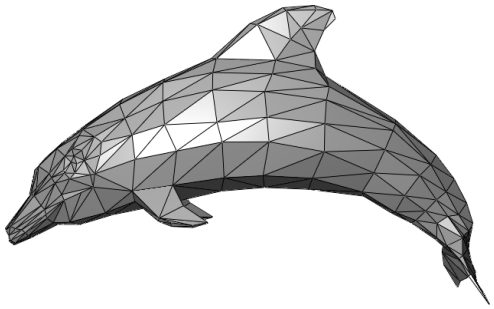
porém com coordenadas em $\mathbb{R}^{d+1}$. Elas são utilizadas para representar a superfície, ou a casca, de um objeto, em vez do seu interior, como foram apresentadas as malhas nesta seção (Figura 2.7).



(a) Bidimensional.



(b) Tridimensional. Fonte: `www.cgal.org`.



(c) De superfície. Fonte: `en.wikipedia.org`.

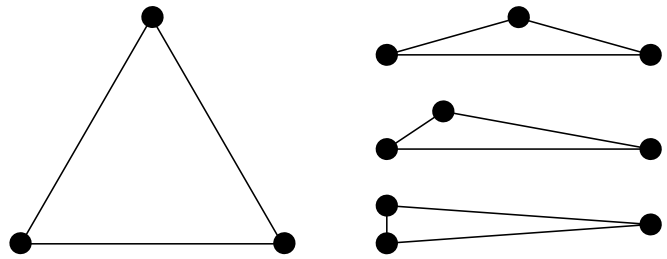
Figura 2.7: Exemplos de malhas.

Métodos de Elementos Finitos

Uma das aplicações mais importantes que utilizam malhas é o método de elementos finitos, ou simplesmente MEF. O MEF é um método numérico de resolução de sistemas de equações diferenciais que visa predizer o comportamento de objetos sob a ação de certos fenômenos físicos, tais como pressão, calor, fluxo de fluidos ou vibrações acústicas, entre outros. Para tal finalidade, um objeto é discretizado, ou seja, uma malha é gerada para este objeto, e suas propriedades físicas são definidas nos vértices (ou nós) da malha. O MEF pode ser reduzido a um método de resolução de sistemas de equações. Os resultados da aplicação do MEF sobre uma malha indicam, geralmente, um campo de respostas (deslocamento, temperatura) com valores discretos nos seus vértices.

Uma simulação numérica é a aplicação iterativa do MEF a um determinado problema. Essa simulação pode ser adaptativa, ou seja, pode-se refinar a malha onde necessário a cada iteração.

Uma boa malha leva a uma boa precisão numérica, o que leva a resultados precisos e mais realistas [Frey e George 2000]. A qualidade de uma malha depende diretamente da qualidade de seus elementos, que pode ser medida através de suas razões de aspecto (*aspect ratio*), entre outras medidas. Pode-se dizer que um triângulo ou tetraedro é bom quando ele é quase equilátero, caso contrário, ele pode ser ruim (Figura 2.8). Se uma malha tiver uma grande quantidade de elementos ruins, é possível que o MEF não convirja, podendo resultar em um campo de respostas imprecisas ou irreais.



(a) Triângulo bom (equilátero).

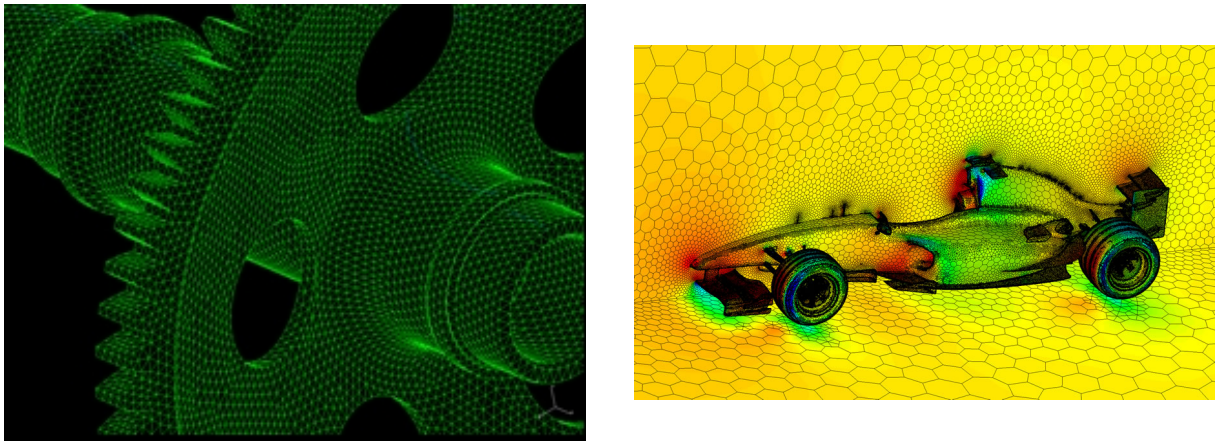
(b) Triângulos ruins.

Figura 2.8: Triângulo bom e triângulos ruins.

Em aplicações de MEFs, uma malha não precisa ter elementos de tamanho uniforme. Onde a discrepância de resultados é maior, utiliza-se uma grande quantidade de elementos pequenos, e onde a diferença não é tão grande, utiliza-se menos elementos, mas de tamanho maior. Essa não-uniformidade é justificada pois a precisão dos resultados depende inclusive do tamanho dos elementos (por isso, usa-se elementos menores onde mais se precisa) e o tempo de execução dos MEFs depende da quantidade de elementos da malha (por isso, usa-se menos elementos onde não se precisa tanto). Exemplos de malhas uniforme e não-uniforme podem ser vistos nas Figuras 2.9a e 2.9b, respectivamente. Perceba que, na Figura 2.9b, a região vermelha é mais refinada, pois necessita de resultados mais precisos, e a região amarela, mais afastada, tem elementos maiores.

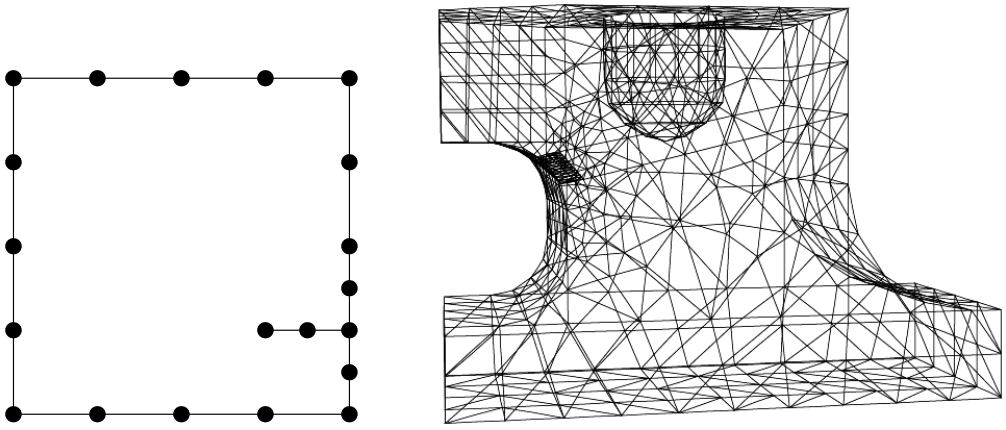
Em algumas aplicações de MEFs, analisa-se a propagação de trincas e/ou fraturas em objetos reais. Para tais aplicações, é necessário modelar-se a fratura na malha (Figura 2.10), que é geralmente idealizada como uma região sem volume. As superfícies que representam os dois lados da fratura são distintas, mas geometricamente coincidentes. Isso significa que pontos em lados opostos da fratura podem ter coordenadas coincidentes [Cavalcante-Neto et al. 2001].

Por necessitar de malhas refinadas, é comum que MEFs usem malhas grandes, com milhares ou milhões de elementos. Por influenciar no tempo de execução dos MEFs, a geração dessas malhas deve ser feita de forma rápida. Com o desenvolvimento e o barateamento de



(a) Malha uniforme. (b) Malha não-uniforme.

Figura 2.9: Malhas uniforme e não-uniforme (repetido da Figura 1.1).



(a) Fratura bidimensional. (b) Fratura tridimensional.

Figura 2.10: Exemplo de bordas com fratura.

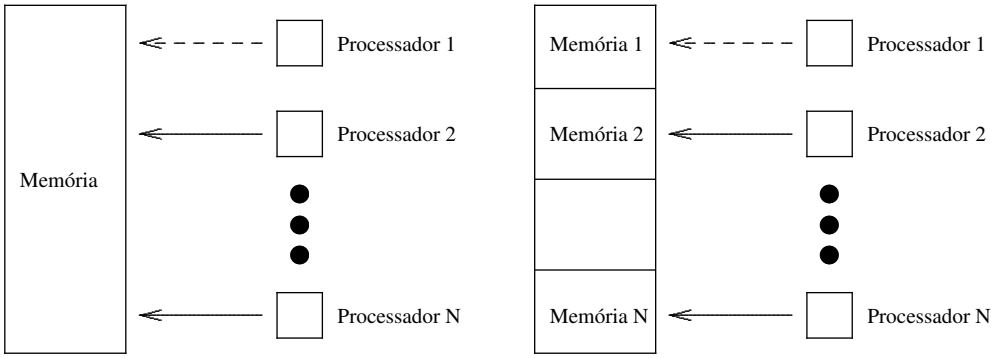
arquiteturas computacionais paralelas (com vários processadores), fazem-se necessárias pesquisa e elaboração de técnicas de geração em ambientes computacionais de alto desempenho de malhas.

2.2.2 Computação de Alto Desempenho

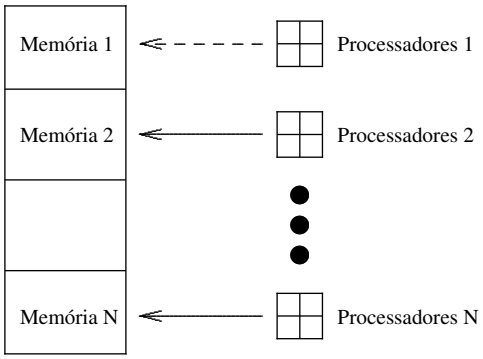
A computação de alto desempenho (ou *high performance computing*, HPC) é a área da computação que estuda o uso colaborativo de diversos processadores para a solução de um mesma instância de um problema. Com o barateamento dessa tecnologia, diversos computadores de mais de um processador têm-se popularizado nos últimos anos. Alguns conceitos aqui apresentados podem ser observados em [Grama et al. 2003].

Modelos de Arquiteturas

As arquiteturas paralelas podem ser classificadas em compartilhadas ou distribuídas. Em arquiteturas compartilhadas (Figura 2.11a), todos os processadores de um computador compartilham de um mesmo espaço de memória. Já em arquiteturas distribuídas, cada processador tem acesso exclusivo ao seu espaço de memória (Figura 2.11b). Há ainda as arquiteturas mistas, ou híbridas, em que diversos computadores de memória compartilhada são organizados em um grande computador (Figura 2.11c) de memória distribuída.



(a) Compartilhada. (b) Distribuída.



(c) Mista ou híbrida.

Figura 2.11: Arquiteturas paralelas.

Os computadores paralelos mais comuns, disponíveis em computadores de mesa (*desktops*) hoje em dia, são os de memória compartilhada. Dessa forma, todos os processadores (ou núcleos de um mesmo processador, Figura 2.12) têm fácil acesso a todas as estruturas de dados de um certo programa. É interessante, nesses programas, que um mesmo dado não possa ser modificado por mais de um processador ao mesmo tempo, para não ocasionar perda e/ou invalidez de dados na memória.

Existem ainda os processadores da Intel® com tecnologia *hyper-threading* (ou simplesmente HT). Nestes processadores, cada núcleo tem duas *pipelines*, uma para o processamento de números inteiros e outra para o processamento de números de ponto flutuante. Dessa ma-

neira, o sistema operacional enxerga o dobro de núcleos do que realmente tem o processador.

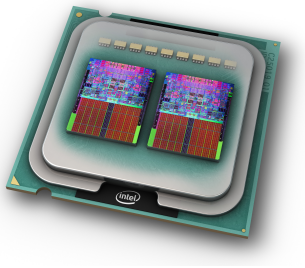


Figura 2.12: Arquitetura de memória compartilhada - processador Intel® com quatro processadores. Fonte: `www.intel.com`.

Outra infra-estrutura que tem-se barateado com o tempo são os computadores de memória distribuída. Nessas arquiteturas, cada processador tem o seu próprio espaço de memória exclusivo. Assim, se um processador tiver que acessar dados em outro processador, é necessário haver comunicação inter-processos, onde mensagens são transferidas entre os dois para que os dados sejam enviados. Fazem parte dessa arquitetura os *clusters* (Figura 2.13), que também podem ser formados por diversos computadores de mesa conectados por uma rede local, e as grades computacionais, que são diversos computadores, em diferentes locais do mundo, interligados pela internet. Se alguns desses computadores tiverem processadores multi-núcleo, tem-se a arquitetura híbrida, que tem-se tornado muito comum ultimamente, principalmente pela facilidade de aquisição e manutenção.

Figura 2.13: Arquitetura de memória distribuída - *cluster* SGI. Fonte: `www.sgi.com`.

Taxonomia

Deixando de lado o *hardware* e indo mais a fundo na teoria de HPC, as arquiteturas de computadores podem ser classificadas como [Flynn 1972]:

- SISD - *Single Instruction Single Data*. Um computador sequencial tradicional, onde somente um dado é processado por vez (Figura 2.14a),
- SIMD - *Single Instruction Multiple Data*. Um computador onde a mesma instrução é aplicada em diferentes dados ao mesmo tempo (Figura 2.14b),
- MISD - *Multiple Instruction Single Data*. Um computador onde diversas instruções são aplicadas a um mesmo dado ao mesmo tempo, uma arquitetura incomum (Figura 2.14c),
- MIMD - *Multiple Instruction Multiple Data*. Um computador onde diversas instruções são aplicadas a diversos dados (Figura 2.14d).

Esse conceito pode ser estendido para o nível de programas, onde algoritmos inteiros são aplicados aos dados, em vez de apenas instruções. Os casos mais comuns são os de SPMD (um mesmo programa para diversos dados) e MPMD (vários programas para diversos dados).

Decomposição e Agrupamento

Na construção de um programa concorrente, é necessário fazer-se a decomposição do problema, para que cada 'pedaço' do problema seja executado em um processador. Comumente, essa decomposição é de dados ou de funcionalidades. Na decomposição de dados, os diversos dados são subdivididos em instâncias menores, que são resolvidas ao mesmo tempo. Posteriormente, os resultados são combinados para se obter o resultado final. Os algoritmos de divisão e conquista são exemplos simples deste tipo de decomposição.

Na decomposição de funcionalidades, cada processador é responsável por uma parte da execução em todos os dados. Exemplos desse tipo de decomposição são os programas em *pipeline*, onde um processador faz operações em alguns dados e passa os resultados para o próximo processador, que faz outras operações.

Após a decomposição do programa em 'pedaços', chamados de unidades de paralelismo, partições ou subdomínios, estes são agrupados para satisfazer a quantidade real de processadores, pois pode haver mais unidades que processadores físicos. Geralmente, o tamanho desses agrupamentos influencia o tempo de execução total do programa.

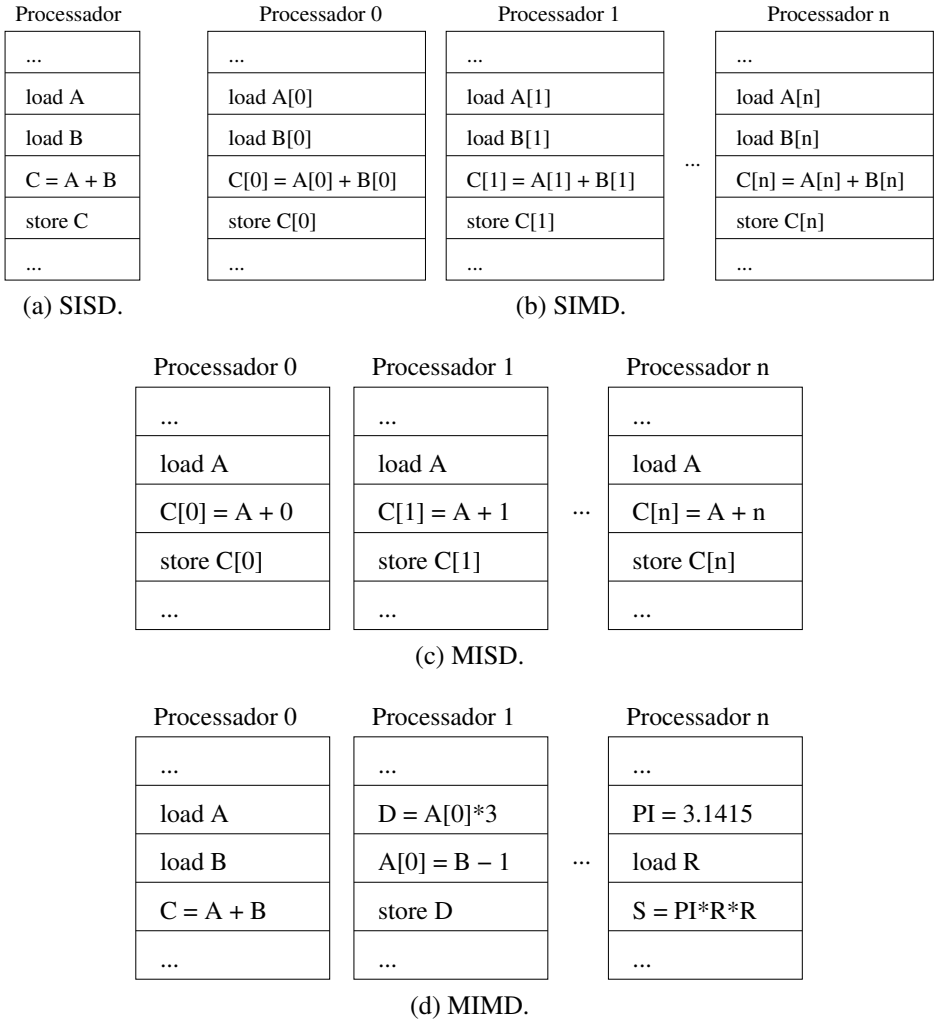


Figura 2.14: Taxonomia de [Flynn 1972].

Balanceamento de Carga

Cada partição, na grande maioria das vezes, tem uma carga diferente, que diz respeito à quantidade de computação a ser processada por aquela partição. O problema do balanceamento de carga diz respeito à divisão das N unidades de paralelismo entre os P processadores.

Idealmente, a carga total deve ser dividida igualmente entre todos os processadores, para que estes acabem suas execuções (aproximadamente) ao mesmo tempo. Se a carga estiver muito desbalanceada, muitos processadores poderão esperar, ociosos, que os últimos processadores terminem, degradando o desempenho final do programa, e, em um caso extremo, aproximando seu tempo de execução a um algoritmo sequencial.

Para tentar resolver o problema do balanceamento de carga, que é NP-completo [Lawlor et al. 2006], existem diversos algoritmos na literatura. Os tipos de balanceamento (nomenclaturas utilizadas neste trabalho) são o estático e o dinâmico. No balanceamento estático, o conjunto

de partições de um processador é imutável, ou seja, cada processador trata somente dos subdomínios inicialmente atribuídos a ele. Já no balanceamento dinâmico, este conjunto pode ser modificado com o tempo. O balanceamento dinâmico pode-se dar por frequentes reparticionamentos do domínio (no caso de particionamento de dados) ou por migração de partições (ou parte delas) para outros processadores.

Escalabilidade e *Speed-up*

A escalabilidade de um algoritmo paralelo diz respeito ao seu comportamento conforme se aumenta o número de processadores. Um algoritmo é dito escalável se ele pode ser utilizado em uma quantidade grande de processadores sem sofrer muita degradação. Em um algoritmo escalável, idealmente, P processadores fazem com que o tempo de execução decaia a (aproximadamente, ou a um fator constante de) $T(n)/P$, onde $T(n)$ é o tempo de execução do algoritmo com 1 processador.

A escalabilidade de um algoritmo está relacionada a sua aceleração (o termo inglês *speed-up* é mais comum), que é definida como a razão entre o tempo de execução do melhor algoritmo sequencial pelo tempo de execução do algoritmo paralelo, montando-se um gráfico (Figura 2.15) de *speed-up* pela quantidade de processadores. Este gráfico indica qual o ganho de performance de uma dada aplicação sobre uma implementação sequencial. O *speed-up* é uma medida que visa capturar o benefício relativo de resolver um problema em paralelo.

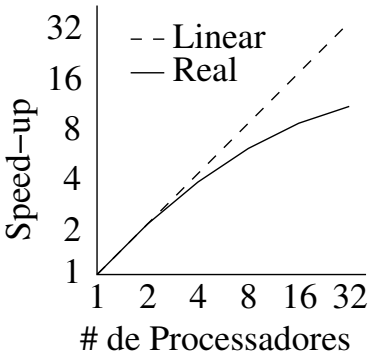


Figura 2.15: Exemplo de gráfico de *speed-up*.

Neste tipo de gráfico, é possível ver quantas vezes mais rápido o algoritmo ficou quando foi utilizada uma determinada quantidade de processadores. Em um *speed-up* linear, p processadores fazem o algoritmo ser p vezes mais rápido. Em um *speed-up* super-linear, p processadores fazem o algoritmo ser mais de p vezes mais rápido. Geralmente, o gráfico de *speed-up* é logarítmico pois, conforme aumenta-se a quantidade de processadores, aumenta-se a comunicação entre eles, cujo tempo acaba sobressaindo o de computação do programa.

A escalabilidade de um algoritmo pode estar relacionado com seu grau de acoplamento. Em um algoritmo desacoplado, a comunicação entre dois processos é inexistente ou muito pequena. Algoritmos desacoplados são, geralmente, escaláveis. Um algoritmo também pode ser classificado como parcialmente acoplado, quando a comunicação não compromete tanto sua escalabilidade, ou muito acoplado, quando a comunicação é muito intensa, podendo degradar sua escalabilidade.

Malhas e Paralelismo

Existem, basicamente, dois problemas que relacionam paralelismo com malhas [Frey e George 2000]. O primeiro problema é o de particionar uma malha previamente gerada entre os processadores, onde, possivelmente, serão aplicados métodos de elementos finitos. O outro problema é o de gerar uma malha em paralelo, problema este que é o estudado nesse trabalho.

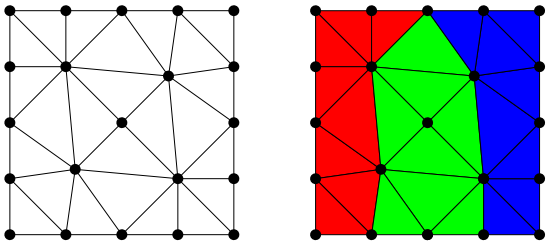
Particionamento de Malhas Como dito anteriormente, os MEFs podem ser reduzidos a sistemas de equações. É sabido [Golub e Loan 1996] que os sistemas de equações lineares $Ax = b$ podem ser resolvidos concorrentemente, particionando-se a matriz de dados A e o vetor resultado b. Particionar a matriz A de um MEF, para os matemáticos, pode ser encarado como particionar uma malha em vários subdomínios, para os engenheiros.

Um bom particionamento da malha leva a um menor tempo de execução do algoritmo. Este bom particionamento diz respeito a alguns requisitos, como a minimização da disparidade entre as cargas de dois subdomínios (o número de elementos ou de vértices), para um bom balanceamento da carga, e a minimização da quantidade de elementos entre dois subdomínios, uma vez que esses elementos determinam a quantidade de comunicação entre os subdomínios.

No particionamento de malhas, duas abordagens são estudadas na literatura. Em uma delas, o particionamento de elementos, elementos adjacentes são agrupados para formar um subdomínio (Figuras 2.16 e 2.17). Na outra abordagem, são os vértices da malha que devem ser particionados.

Uma vez que uma malha pode ser encarada como um grafo, retirando-se as informações geométricas da malha, a primeira abordagem de particionamento pode ser reduzida à segunda, utilizando-se o grafo dual da malha. No grafo dual, existe um vértice para cada elemento da malha, e uma aresta para cada adjacência entre elementos. Geralmente, são atribuídos pesos aos vértices e às arestas, indicando a carga do elemento e o custo de comunicação entre elementos, respectivamente. O problema de particionamento de grafos e, portanto, o de particionamento

de malhas, é NP-completo [Biswas et al. 2000, Lämmer e Burghardt 2000].



(a) Malha inicial. (b) Malha particionada.

Figura 2.16: Problema de particionamento de malhas, onde cada agrupamento de elementos de uma cor específica é uma partição.

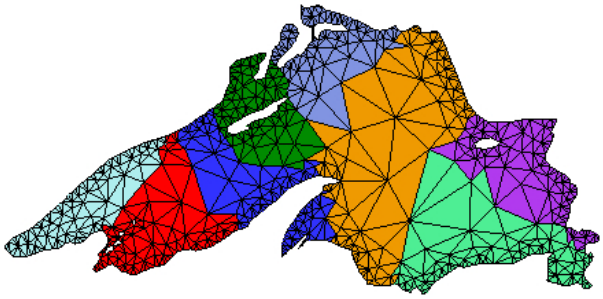
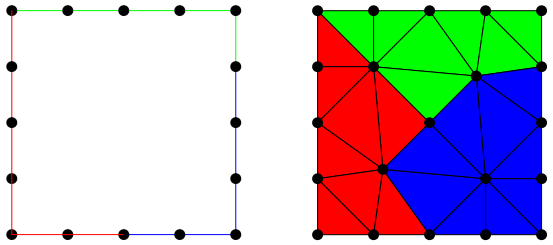


Figura 2.17: Malha particionada (*Lake Superior*, EUA e Canadá. Fonte: `www.cs.cmu.edu/~quake/triangle.html`).

Geração em Paralelo de Malhas Devido à crescente quantidade de problemas de tamanhos cada vez maiores, fizeram-se necessários o estudo e a implementação de MEFs em computadores paralelos. Em aplicações onde a malha é refinada a partir de resultados anteriores, a utilização de geradores sequenciais de malhas, além de ser um gargalo, seria um desperdício de infra-estrutura. Dessa demanda, surgiram os geradores de malha em paralelo.

Na geração em paralelo de malhas (Figura 2.18), existem duas formas de particionar o domínio. Na primeira forma, uma malha grosseira da região é rapidamente gerada, sequencialmente, e dividida entre os processadores. Essa forma, chamada de particionamento explícito no presente trabalho, envolve ainda o problema de particionamento da malha. A segunda forma de particionar o domínio envolve dividir a região a partir de funções, segmentos, eixos inerciais, ou estruturas auxiliares, por isso chamada de particionamento implícito neste trabalho. Cada subdomínio é enviado a um processador, onde a malha será gerada.

Tanto no particionamento explícito quanto no implícito, um processador não pode gerar malha na sub-região de outro processador. Portanto, é necessário haver comunicação entre os processos quando se trata de gerar malha entre dois subdomínios ou em suas bordas.



(a) Região inicial particionada.

(b) Malha gerada em paralelo.

Figura 2.18: Problema de geração de malhas em paralelo, onde cada agrupamento de elementos de uma cor específica foi gerada por um processador.

2.3 Geração de Malhas

Esta Seção descreve brevemente algumas técnicas de geração de malha, com alguns resultados práticos ou teóricos, juntamente com alguns conceitos de paralelização delas. As técnicas aqui discutidas se enquadram em três categorias:

Avanço de fronteira, onde a malha deve ser criada a partir de restrições que formem o contorno da região,

Delaunay, onde tenta-se maximizar o menor ângulo dos triângulos gerados,

Arbitrária, como são chamadas as outras técnicas de geração de malhas.

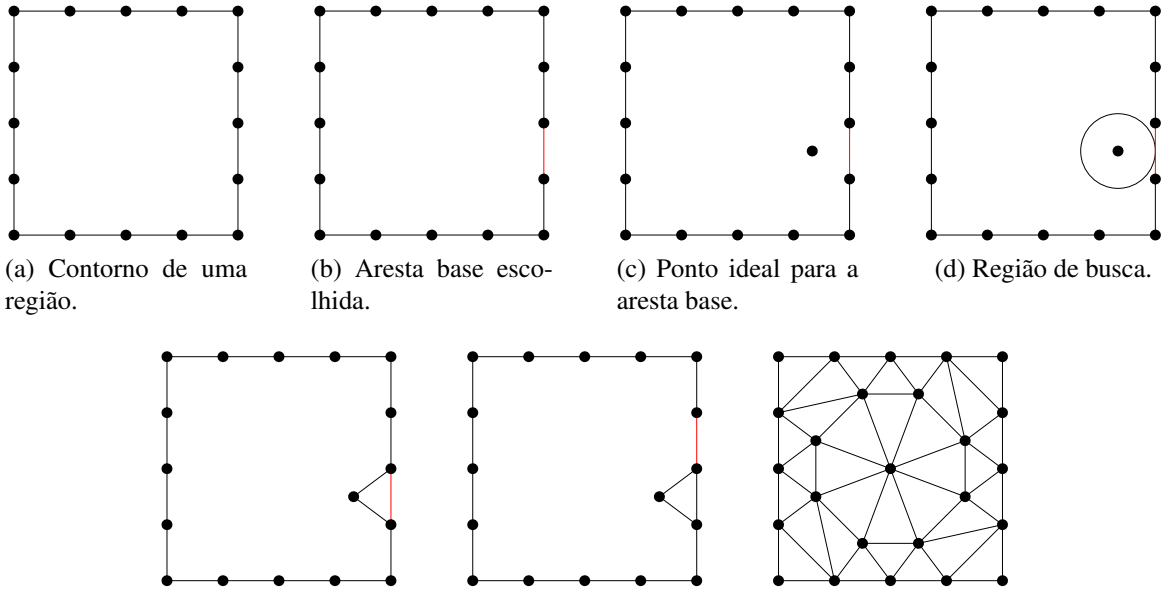
2.3.1 Avanço de Fronteira

Avanço de Fronteira Sequencial

Os algoritmos de avanço de fronteira (ou AFT, do inglês *advancing front technique*) partem de um contorno especificado da região a ser preenchida (Figura 2.19a). 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 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 de Steiner, ou seja, insere novos pontos, que não pertencem à entrada. Um algoritmo

nessa categoria procede da seguinte forma, no caso 2D triangular (Figura 2.19), enquanto houver arestas na fronteira:



(a) Contorno de uma região. (b) Aresta base escolhida. (c) Ponto ideal para a aresta base. (d) Região de busca.

(e) Formação do novo triângulo. (f) Nova aresta base escolhida. (g) Malha final.

Figura 2.19: Avanço de fronteira.

1. Escolha e remova uma aresta da fronteira, a aresta base (Figura 2.19b),
2. Encontre um ponto ideal para a formação de um novo triângulo com a aresta base (Figura 2.19c),
3. Crie uma região de busca em torno desse ponto ideal (Figura 2.19d),
4. Selecione o ponto dentro dessa região de busca cujo triângulo (entre esse ponto e a aresta base) seja válido e seja o de melhor qualidade,
   (a) Caso não exista nenhum ponto nessa região, verifique se o triângulo formado com o ponto ideal é válido,
   (b) Se for válido, adicione esse ponto à lista de pontos. 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 ponto selecionado e adicione-o à malha (Figura 2.19e),
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.

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, provendo estabilidade e precisão na aplicação de métodos numéricos (como os MEFs) sobre a malha gerada.

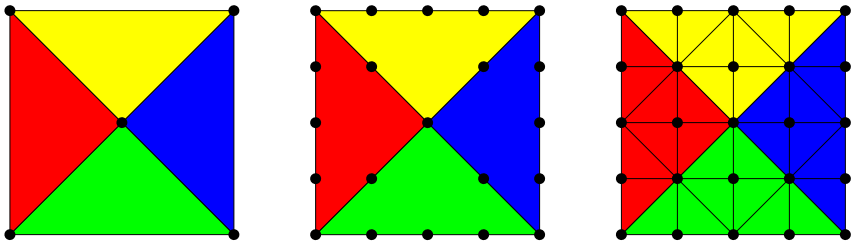
Entretanto, em algumas vezes, os elementos mais internos à malha nem sempre são de boa qualidade, pois a região torna-se menor à medida que a fronteira avança. Para tratar esses casos, geralmente uma técnica de suavização ou otimização é aplicada na malha resultante do algoritmo. Existem, ainda, alguns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

É muito difícil calcular-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 $O(CN^p \log N)$, onde N é o número de elementos da saída, $0 \leq C \leq 1$ e $1 \leq p \leq 2$ [Cavalcante-Neto et al. 2001].

Avanço de Fronteira Paralelo

Como foi dito anteriormente, o avanço de fronteira sequencial gera um elemento por vez. Portanto, a ideia envolvida na paralelização deste tipo de algoritmo é gerar vários elementos ao mesmo tempo. Assim, divide-se a região em sub-regiões, que são preenchidas concorrentemente. Como dito anteriormente, essa divisão pode ser explícita ou implícita.

Particionamento Explícito Os algoritmos paralelos de avanço de fronteira por particionamento explícito geram uma malha grosseira para a região, a partir do contorno dado, utilizando um algoritmo sequencial. Tem-se, então, duas abordagens comumente usadas. Na primeira delas (Figura 2.20), cada triângulo é considerado como uma sub-região [Wilson e Topping 1998]. Na segunda (Figura 2.21), cada sub-região é formada pelo agrupamento de triângulos próximos, cujos elementos internos são removidos [Ito et al. 2007]. Uma vez definidas as sub-regiões, suas bordas são refinadas (tanto as interfaces entre duas sub-regiões quanto o contorno dado como entrada). Então, um algoritmo de avanço de fronteira sequencial é aplicado em cada sub-região em um processador diferente.

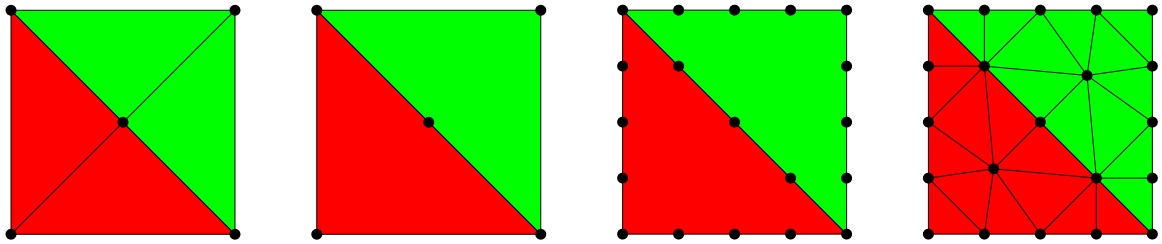


(a) Particionamento inicial em 4 subdomínios.

(b) Refinamento das bordas (contorno e interfaces).

(c) Malha final gerada em paralelo.

Figura 2.20: Geração em paralelo por particionamento explícito sem agrupamento.



(a) Particionamento inicial em 2 subdomínios.

(b) Agrupamento dos triângulos.

(c) Refinamento das bordas (contorno e interfaces).

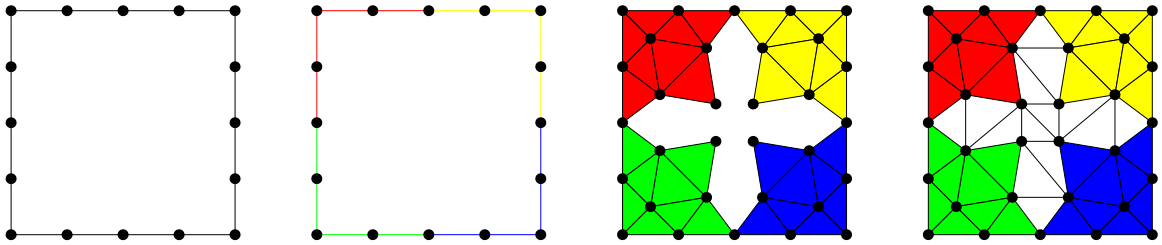
(d) Malha final gerada em paralelo.

Figura 2.21: Geração em paralelo por particionamento explícito com agrupamento.

Esse tipo de algoritmo tem a vantagem de não necessitar de nenhuma modificação nos algoritmos de avanço de fronteira, podendo-se utilizar algoritmos estáveis e bem testados. Além do mais, não é necessária muita comunicação entre as sub-regiões. Entretanto, se o particionamento não for bem feito, 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.

Particionamento Implícito Nos algoritmos de avanço de fronteira por particionamento implícito, uma sub-região é definida como uma parte do contorno, juntamente com uma parte da região interna a ele. Então, malhas são geradas, uma para cada sub-região, concorrentemente. Posteriormente, os processadores sincronizam e geram malhas entre duas sub-regiões. Esse procedimento continua até que não haja mais região para ser preenchida (Figura 2.22).

Esse tipo de algoritmo tem a vantagem de não precisar particionar uma malha, um problema NP-completo, além de não precisar definir contornos internos, que podem interferir na qualidade da malha final. Outra vantagem desse algoritmo é a de preservar o contorno dado como entrada, sem modificá-lo através de refinamentos. Entretanto, o algoritmo sequencial deve ser



(a) Contorno inicial. (b) Particionamento do contorno. (c) Malhas geradas em paralelo. (d) Malha final após sincronização.

Figura 2.22: Geração em paralelo por particionamento implícito.

levemente modificado para decidir se a criação de um ponto interno respeita os limites de uma sub-região. Além disso, é necessário haver sincronização para a geração completa da malha.

As técnicas de particionamento implícito envolvem subdivisões regulares do domínio, subdivisões recursivas (*quadtree* ou *octree*) [Löhner 2001], criação de linhas de contorno [Globisch 1995] ou planos separadores apropriados para o particionamento [deCougny e Shephard 1999, Zagaris et al. 2009] (definindo contornos internos, o que pode levar aos mesmos problemas dos algoritmos de particionamento explícito). O particionamento implícito deve ser facilmente encontrado, para não degradar a performance do algoritmo como um todo.

A Tabela 2.1 indica as principais vantagens e desvantagens dos métodos de avanço de fronteira por particionamento implícito e explícito.

Tabela 2.1: Avanço de fronteira: particionamento explícito x particionamento implícito.

Característica	Explícito	Implícito
Algoritmos conhecidos	Sem modificações ☺	Com modificações
Transição entre elementos	Pode ser ruim	Boa ☺
Contornos internos	Necessários	Não necessários ☺
Refinamento da borda	Necessário	Não necessário ☺
Forma de particionamento	Malha grosseira	Qualquer ☺
Sincronização	Inexistente/pouca ☺	Razoável/muita

2.3.2 Delaunay

Delaunay Sequencial

Na técnica de Delaunay básica (nome dado graças ao matemático russo Boris Delaunay), a região a ser preenchida é o fecho convexo do conjunto de pontos dados como entrada, sem utilizar pontos de Steiner. Essa técnica é bem estudada e tem uma teoria bem formalizada [Hjelle e Dæhlen 2006]. A triangulação gerada por Delaunay é um grafo dual ao diagrama de

Voronoi (nome dado por conta do também matemático russo Georgy Voronoi) (Figura 2.23). Dessa forma, um ponto qualquer é sempre adjacente ao ponto mais próximo dele.

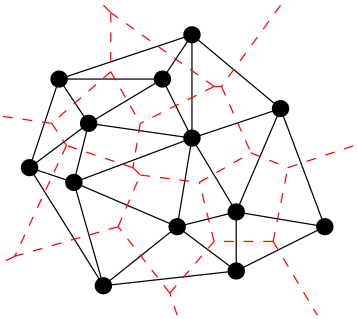
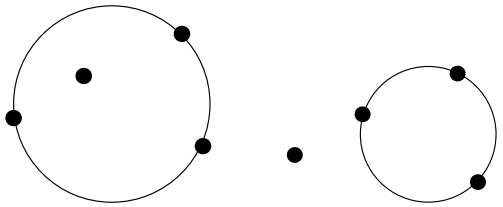


Figura 2.23: Diagrama de Voronoi (em vermelho).



(a) Critério não satisfeito.

(b) Critério satisfeito.

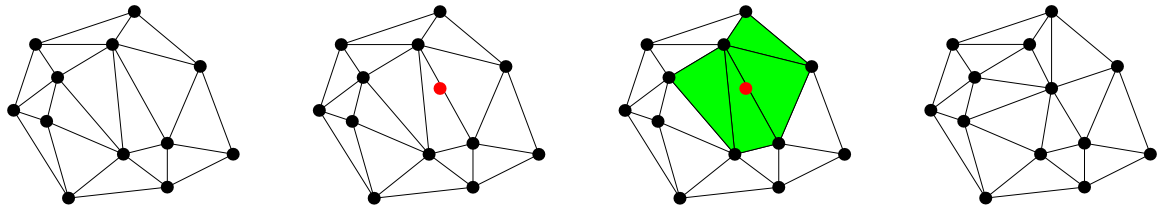
Figura 2.24: Critério de Delaunay.

Para um triângulo ser chamado de triângulo de Delaunay, e, portanto, pertencer à malha final, é necessário que não exista nenhum outro ponto dentro do círculo que passa pelos três pontos desse triângulo (o circuncírculo desse triângulo, Figura 2.24), chamado de critério do círculo ou critério de 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. Mais especificamente [Hjelle e Dæhlen 2006], para cada triangulação possível T de um conjunto de pontos P, associe um vetor indicador $I(T) =< \alpha_1, \alpha_2, \ldots, \alpha_{|T|} >$, onde α_i representa o menor ângulo interno de cada triângulo em T e as entradas do vetor $I(T)$ estão ordenadas de forma não-decrescente (ou seja, $\alpha_i \leq \alpha_j$, para $i < j$). É possível, então, definir uma ordenação lexicográfica entre os vetores $I(T)$ para o mesmo conjunto P, onde $I(T) < I(T')$ se, para algum inteiro m, $\alpha_m < \alpha'_m$ e $\alpha_i = \alpha'_i$, para $i = 1, 2, \ldots, m-1$. Dessa forma, a triangulação de Delaunay T^D é aquela cujo $I(T^D)$ é máximo, na ordem lexicográfica.

Existem dois tipos de algoritmos de Delaunay. No primeiro tipo, encontra-se uma aresta que faz parte da triangulação e, a partir dela, acha-se um triângulo de Delaunay. Assim, com as novas arestas, encontra-se novos triângulos, em um algoritmo parecido com o de avanço de fronteira. No segundo tipo, insere-se um ponto por vez, criando triângulo inicial ou modificando-se uma triangulação de Delaunay (de apenas um subconjunto de pontos da entrada) pré-existente (Figura 2.25).

A complexidade do problema de gerar uma triangulação de Delaunay é $\Theta(n \log n)$. Existem variações da técnica de Delaunay com restrições, que podem ser tanto internas quanto de fronteira (reveja a Figura 2.4). Nesses casos, a malha gerada é a mais próxima de uma Delaunay possível, respeitando as restrições [Hjelle e Dæhlen 2006].



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

Figura 2.25: Triangulação por inserção de pontos.

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, gerando instabilidade em métodos numéricos. Para contornar essa situação, refinamentos e otimizações, que fazem uso de pontos de Steiner, são utilizados para se melhorar a qualidade da malha [Ruppert 1999].

Delaunay Paralelo

Os algoritmos paralelos de geração de malhas de Delaunay geralmente tem início a partir de uma malha de Delaunay gerada sequencialmente. Essa malha pode ser um triângulo englobando todos os pontos dados como entrada (gerado em uma etapa de pré-processamento), uma malha para alguns dos pontos dados, ou uma malha de todos os pontos. A partir dessa malha, pontos são inseridos concorrentemente. Esses pontos podem ser da entrada (quando a malha for um triângulo ou somente para alguns dos pontos da entrada) ou criados por algoritmos de refinamento de malhas de Delaunay [Okusanya e Peraire 1996, Chernikov e Chrisochoides 2006].

Sempre que um novo ponto é inserido, os triângulos afetados, ou seja, aqueles cujos circuncírculos contêm o novo ponto, devem ser removidos e novos triângulos que satisfaçam o critério de Delaunay devem ser gerados. Se a malha estiver particionada entre os processadores, essa operação pode envolver a migração da parte afetada para outro processador. Após a malha ser gerada, os novos triângulos podem permanecer no processador ou podem ser divididos entre os processadores envolvidos [Chrisochoides e Nave 2000]. A determinação da parte afetada pode necessitar do uso de travas (*locks*) [Kohout et al. 2005], além de sincronização.

Assim como os algoritmos de avanço de fronteira, os algoritmos de Delaunay também podem particionar a entrada explicita ou implicitamente. No caso do particionamento explícito, usa-se um particionador de grafos [Chrisochoides e Nave 2000]. No particionamento implícito, usa-se subdivisões regulares do domínio [Chernikov e Chrisochoides 2006], planos separadores apropriados para o particionamento [Okusanya e Peraire 1996] ou particionamento das estrutu-

ras de dados [Tang et al. 1993, Kohout et al. 2005].

2.3.3 Arbitrária

Arbitrária Sequencial

Neste trabalho, são chamadas malhas arbitrárias aquelas que não se enquadrarem nem como avanço de fronteira nem como Delaunay. Algumas das malhas aqui agrupadas são as malhas de *quadtree*/*octree* e as geradas por algoritmos de varredura [Carvalho e Figueiredo 1991], entre outras.

Por não terem tantos requisitos, as malhas arbitrárias podem ser utilizadas para demonstrações de teoremas sobre malhas. Um teorema importante, por exemplo, afirma que o problema de ordenação de pontos pode ser reduzido ao problema de geração de malhas bidimensionais [Carvalho e Figueiredo 1991]. Assim sendo, para se achar uma malha triangular do fecho convexo de um conjunto de pontos em duas dimensões, necessita-se de um tempo de $\Omega(n \log n)$, com n sendo a quantidade de pontos da entrada.

Uma demonstração para esse teorema pode ser feita da seguinte maneira. Considere uma sequência de valores, $< x_1, x_2, \ldots, x_n >$, a ser ordenada. Construa, então, os pontos $< (0,1), (x_1,0), (x_2,0), \ldots, (x_n,0) >$ e obtenha uma triangulação qualquer desses pontos. Como o ponto $(0,1)$ é o único não colinear, todos os outros pontos são adjacentes a ele. Dessa forma, basta achar o ponto de menor abscissa (pode ser feito em tempo $O(n)$), e percorrer a malha por triângulos adjacentes por arestas (Figura 2.26). Assim, a ordem percorrida é uma ordenação da sequência dada.

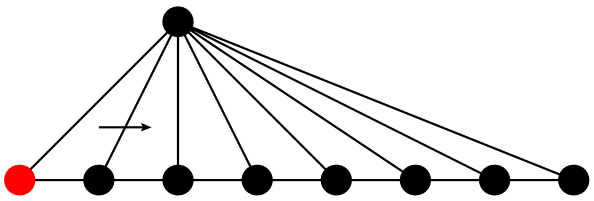


Figura 2.26: Redução da ordenação para a triangulação.

Arbitrária Paralela

Assim como as abordagens de avanço de fronteira e Delaunay, as malhas arbitrárias geradas em paralelo podem ser feitas tanto por particionamento explícito quanto implícito. Depois de serem determinados os particionamentos, cada sub-região é preenchida com elementos, por exemplo,

que satisfazem o critério de Delaunay. Entretanto, nesses casos, a malha como um todo não é de Delaunay, pois os triângulos nas interfaces de cada sub-região podem não satisfazer ao critério [Hodgson e Jimack 1996, Lämmer e Burghardt 2000].

Além disso, as malhas arbitrárias também podem ser utilizadas para a determinação de cotas inferiores para o problema. Foi demonstrado [Merks 1986] que, em duas dimensões, é possível gerar-se uma triangulação arbitrária do fecho convexo em um tempo de $O(\log n)$, utilizando-se $O(n)$ processadores. Para mais de duas dimensões [ElGindy 1986], é possível gerar um complexo simplicial em um tempo de $O(\log^2 n)$, utilizando-se $O(n/\log n)$ processadores, desde que o fecho convexo do conjunto de pontos seja um *simplex*. Nesses dois casos, o *speed-up* obtido multiplicando-se o tempo pelo número de processadores é $O(n \log n)$, que é ótimo para o problema de triangulação.

2.4 Considerações Finais

Este capítulo apresentou alguns conceitos envolvidos no problema de geração em paralelo de malhas, desde as suas bases de geometria computacional, com as definições de fecho convexo, complexo simplicial, triangulação, tetraedralização e malhas, e suas bases de paralelismo, com a apresentação ou definição da infra-estrutura existente, da taxonomia de Flynn, de particionamento, agrupamento, balanceamento de carga, de escalabilidade e *speed-up*.

Além disso, foram apresentados os principais problemas envolvendo malhas e paralelismo, que são o particionamento de malhas e a geração de malhas. No particionamento de malhas, utiliza-se geralmente particionadores de grafos. Na geração, o particionamento do domínio pode ser feito explicitamente, reduzindo o problema de geração ao problema de particionamento de grafos, ou implicitamente, utilizando qualquer outro método.

As técnicas estudadas, de avanço de fronteira, Delaunay e arbitrárias, foram apresentadas. Na técnica de avanço de fronteira, gera-se um elemento por vez a partir da borda do objeto. Na técnica de Delaunay, adiciona-se um ponto por vez à triangulação existente. As técnicas arbitrárias são as outras técnicas, e são muitas vezes utilizadas para provar teoremas sobre malhas.

Algumas formas de paralelizar essas técnicas foram apresentadas. Nos algoritmos de avanço de fronteira, a fronteira deve ser particionada de forma a diversos elementos serem gerados concorrentemente. Já nos algoritmos de Delaunay, diversos pontos devem ser adicionados ao mesmo tempo. As técnicas arbitrárias também são utilizadas para provar teoremas.

Esses conceitos são importantes para que o leitor possa se situar e ter uma visão geral

das pesquisas correntes nesse tema multidisciplinar. O próximo capítulo abordará os trabalhos relacionados com o presente trabalho, na área de geração de malhas em paralelo.

3 *Trabalhos Relacionados*

3.1 Introdução

Este capítulo apresenta, em ordem cronológica, alguns trabalhos que de alguma forma estão relacionados com esta pesquisa. Posteriormente, as técnicas aqui descritas serão classificadas de acordo com os conceitos dados no Capítulo 2:

Tabela 3.1: Classificações.

Relativo a	Classificação		
Técnica	Arbitrário	Delaunay	Avanço de fronteira
Dimensão	Duas	Três	Qualquer
Particionamento	Implícito	Explícito	
Balanceamento de carga	Estático	Dinâmico	
Memória	Compartilhada	Distribuída	
Grau de acoplamento	Acoplado	Parcialmente	Desacoplado
Escalabilidade	Escalável	Parcialmente	Não escalável

3.2 Geração em Paralelo de Malhas

[Merks 1986] e [ElGindy 1986] apresentam 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. [ElGindy 1986] apresenta 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 de [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, divide essa lista ordenada em $\sqrt{n}$ sub-listas, cada lista com $\sqrt{n}$ pontos mais o ponto de menor ordenada, e aloca $O(\sqrt{n})$ processadores 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. Esse método é arbitrário, bidimensional, de particionamento explícito, balanceamento estático, memória compartilhada, desacoplado e escalável.

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. Esse método é arbitrário, n-dimensional, de particionamento explícito, balanceamento estático, memória compartilhada, desacoplado e escalável.

[Tang et al. 1993] apresentam 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. Esse método é de Delaunay, tridimensional, de particionamento implícito, balanceamento dinâmico, memória compartilhada, desacoplado e escalável.

Em [Globisch 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 divida 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, simples-

mente assume que a ordem dada já está balanceada. Esse método é de avanço de fronteira, bidimensional, de particionamento implícito, balanceamento estático, memória distribuída, desacoplado e escalável.

[Okusanya e Peraire 1996] mostram 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 3.1). 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. Esse método é de Delaunay, bidimensional, de particionamento implícito, balanceamento dinâmico, memória compartilhada, muito acoplado e pouco escalável.

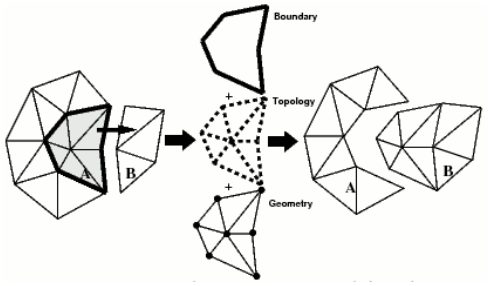


Figura 3.1: Migração de elementos do processador A para o processador B. Fonte: [Okusanya e Peraire 1996].

[Hodgson e Jimack 1996] apresentam 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 de Steiner) 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 de Steiner. 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 3.2). Por utilizar particionamento explícito, essa técnica

pode levar à inserção de artefatos no interior da malha, o que pode influenciar a qualidade dos elementos gerados. Esse método é arbitrário, bidimensional, de particionamento explícito, balanceamento estático, memória distribuída, desacoplado e pouco escalável.

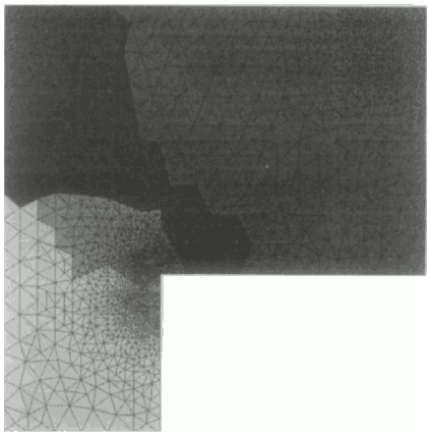
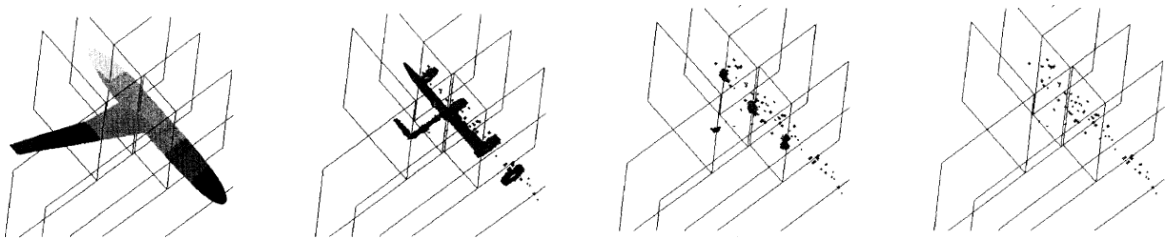


Figura 3.2: Malha gerada pela técnica de [Hodgson e Jimack 1996], já particionada. Fonte: [Hodgson e Jimack 1996].

[Wilson e Topping 1998] apresentam 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 no particionamento explícito, além de não tentar distribuir melhor o trabalho total entre os processadores. Esse método é de avanço de fronteira, tridimensional, de particionamento explícito, memória distribuída, balanceamento estático, desacoplado e escalável.

Em [deCougny 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 3.3a), 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 3.3b). 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 3.3c). Posteriormente, uma técnica de consenso parecida é empregada para as retas definidas pelas interseções de dois planos (Figura 3.3d). 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 elemen-

tos ruins nas proximidades do contorno. Esse método é de avanço de fronteira com *templates*, tridimensional, de particionamento implícito, memória distribuída, balanceamento dinâmico, parcialmente acoplado e não escalável.



(a) Borda com planos de separação.

(b) Fronteira após geração de elementos internos.

(c) Fronteira após geração da malha conectando dois lados do plano.

(d) Fronteira após geração da malha entre interseções de planos.

Figura 3.3: Técnica de [deCougny e Shephard 1999]. Fonte: [deCougny e Shephard 1999].

[Lämmer e Burghardt 2000] apresentam 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. Os resultados apresentados não apresentam uma boa escalabilidade da técnica. Esse método é arbitrário, bidimensional, de particionamento implícito, balanceamento estático, memória distribuída, parcialmente acoplado e não escalável.

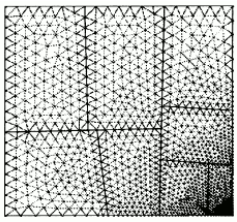
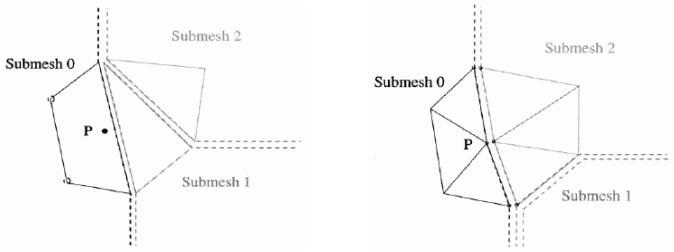


Figura 3.4: Malha gerada pela técnica de [Lämmer e Burghardt 2000]. Fonte: [Lämmer e Burghardt 2000].

[Chrisochoides e Nave 2000] apresentam uma técnica de refinamento de malhas de Delaunay que, ao final, já estão particionadas para a aplicação de métodos de elementos finitos. Uma vez explicitamente particionada a malha entre os processadores, pontos são inseridos simultaneamente, e uma busca em profundidade é feita para se determinar os elementos existentes

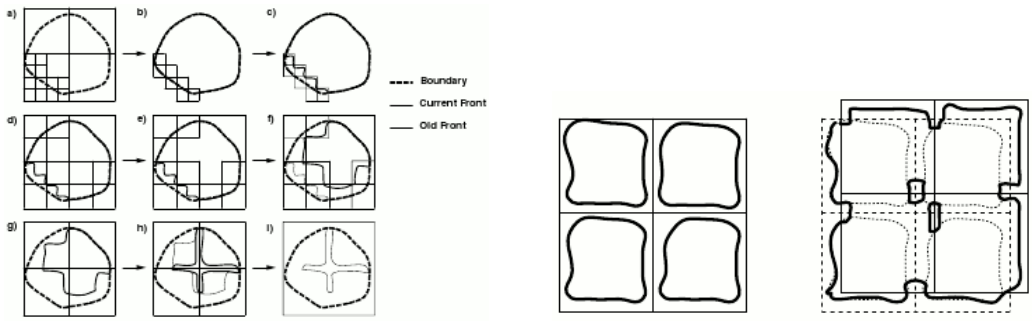
influenciados pelo novo ponto. Se alguns desses elementos encontram-se em outros processadores, eles trocam mensagens e a parte da malha influenciada é transferida para o processador inicial, que insere o novo ponto (Figura 3.5a). A nova malha é, então, particionada entre os processadores envolvidos (Figura 3.5b), 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. Esse método é de Delaunay, tridimensional, de particionamento explícito, balanceamento dinâmico, memória distribuída, parcialmente acoplado e parcialmente escalável.



(a) Malha particionada antes da inserção.

(b) Malha particionada depois da inserção.

Figura 3.5: Inserção paralela de um novo ponto. Fonte: [Chrisochoides e Nave 2000].



(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 3.6: Técnica de [Löhner 2001]. Fonte: [Löhner 2001].

Na técnica de [Löhner 2001], uma *octree* grosseira, com relação ao contorno dado como entrada, é gerada (Figura 3.6a). 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 3.6b). 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. Esse método é de avanço de fronteira, tridimensional, de particionamento implícito, balanceamento estático, memória compartilhada, muito acoplado e não escalável.

[Moretti 2001] descreve uma técnica de geração em paralelo de malhas tridimensionais com fraturas, 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 particionamento explícito, 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. Esse método é de avanço de fronteira, tridimensional, de particionamento explícito, balanceamento estático, memória distribuída, desacoplado e parcialmente escalável.

[Chrisochoides 2005] apresenta um levantamento de algumas técnicas de geração em paralelo de malha. Os algoritmos analisados são de avanço de fronteira, Delaunay e subdivisão de arestas. Além disso, o artigo discute as abordagens de decomposição de domínio (implícita/explícita ou, como no artigo, contínua/discreta), o grau de acoplamento, a estabilidade da técnica (a qualidade dos elementos), o reuso do código, a escalabilidade e forma de geração de malha nas interfaces entre dois subdomínios das técnicas descritas.

[Kohout et al. 2005] apresentam técnicas de Delaunay de implementação prática para computadores de memória compartilhada. Os algoritmos utilizam um grafo direcionado acíclico (DAG) para guardar a malha e auxiliar na busca por triângulos influenciados pela inserção de um novo ponto. Os algoritmos 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. No terceiro 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 3.7). 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. Os métodos são de Delaunay, bidimensionais ou tridimensionais, de particionamento implícito, balanceamento estático, memória compartilhada, parcialmente acoplados e escaláveis.

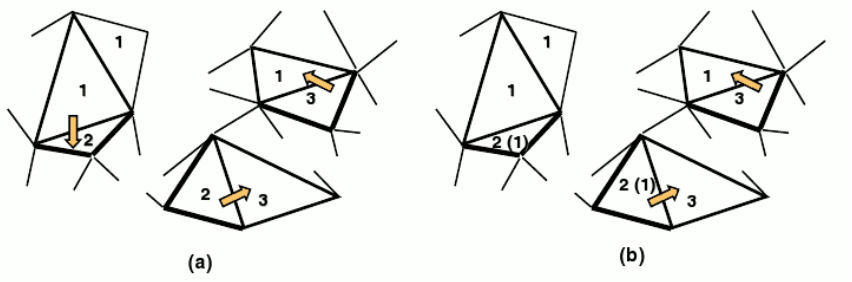


Figura 3.7: Processador 1 "rouba" um triângulo do processador 2, quando as dependências, indicadas pelas setas, causam um *deadlock*. Fonte: [Kohout et al. 2005].

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 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. Esse método é de Delaunay, bidimensional, de particionamento implícito, balanceamento dinâmico, memória compartilhada ou distribuída, parcialmente acoplado e escalável.

[Ito et al. 2007] apresentam uma técnica que gera em paralelo uma malha tridimensional utilizando particionamento explícito. 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 3.8), 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 particionamento explícito, essa técnica também pode apresentar artefatos no interior da malha. Esse método é de avanço de fronteira, tridimensional, de particionamento explícito, balanceamento estático, memória distribuída, desacoplado e escalável.

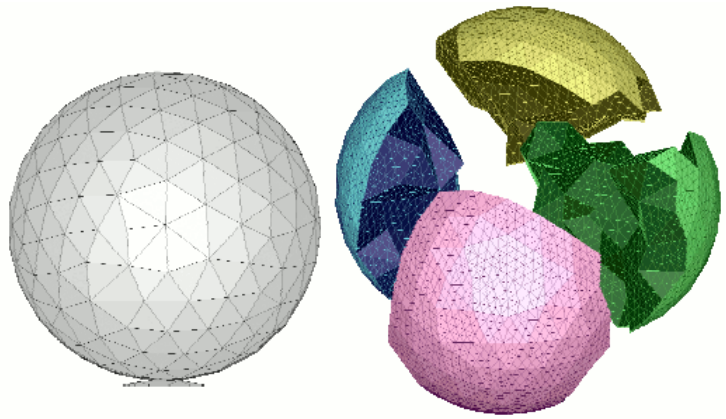
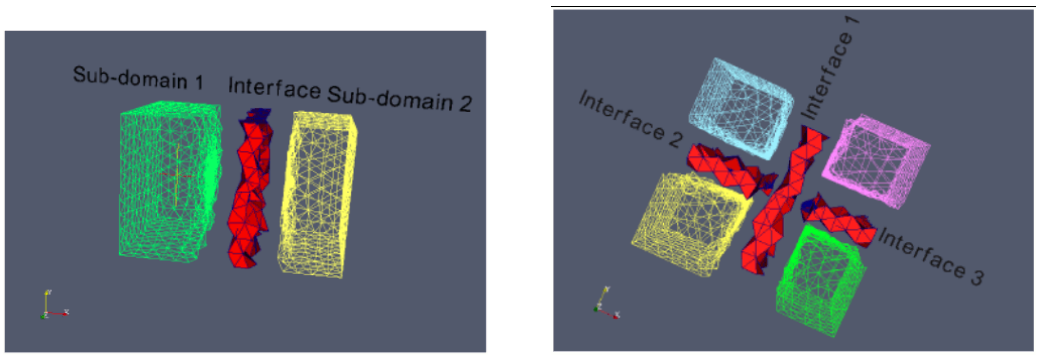


Figura 3.8: Contorno e particionamento de sua malha grosseira refinada. Fonte: [Ito et al. 2007].



(a) Primeiro passo. (b) Passo recursivo.

Figura 3.9: Malhas geradas particionando o domínio. Fonte: [Zagaris et al. 2009].

Na técnica apresentada por [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 3.9a). Assim, duas sub-regiões desconexas são criadas, onde a mesma técnica de particionamento é aplicada novamente (Figura 3.9b), 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. Esse algoritmo é de avanço de fronteira,

tridimensional, de particionamento implícito, balanceamento dinâmico, memória distribuída, desacoplado e parcialmente escalável.

3.3 Considerações Finais

Este capítulo apresentou algumas técnicas paralelas de geração de malhas. Os métodos analisados foram de avanço de fronteira, Delaunay e arbitrária, em duas, três ou qualquer dimensões. Os tipos de particionamento possíveis foram classificados em explícito, quando uma malha é particionada, ou implícito, quando o critério de particionamento leva em conta outros fatores além dos triângulos.

Os tipos de balanceamento analisados foram estático, quando o conjunto de subdomínios de um processador não muda, ou dinâmico, caso contrário. A arquitetura ideal para um determinado algoritmo pode ser de memória compartilhada ou distribuída. Os algoritmos foram classificados em muito acoplados, parcialmente acoplados e desacoplados, de acordo com o grau de comunicação envolvida, e podem ser escaláveis, parcialmente escaláveis, ou não escaláveis.

Apesar dos bons resultados teóricos apresentados pelas malhas de Delaunay, este tipo de algoritmo nem sempre gera malhas de boa qualidade perto do contorno do objeto. Nesse quesito, os algoritmos de avanço de fronteira se sobressaem, uma vez que estes geram bons elementos no contorno, deixando os elementos ruins para o interior do objeto. Esta característica é desejável em MEFs, pois os resultados mais relevantes, em geral, estão no contorno, e não no interior.

Os algoritmos de particionamento implícito têm diversas vantagens sobre os de particionamento explícito (Tabela 2.1) como, por exemplo, um controle melhor sobre a transição entre elementos de tamanhos diferentes. Além disso, o particionamento explícito geralmente requer que o contorno seja refinado, modificando assim o contorno original, o que pode quebrar os requisitos de um método de elementos finitos.

Como em qualquer aplicação de computação distribuída, o problema de geração em paralelo necessita de um bom balanceamento de carga. Dessa forma, uma técnica com balanceamento de carga dinâmico é desejável, uma vez que ela pode, em tempo de execução, determinar qual processador tem menos trabalho e/ou qual está sobrecarregado, e redistribuir as tarefas de forma a garantir que todos os processadores terminem aproximadamente ao mesmo tempo.

Com o aumento da quantidade de núcleos por processador, algoritmos de memória compartilhada têm ficado cada vez mais frequentes e acessíveis, sendo possível a sua execução em

computadores de mesa (*desktops*) atuais. O barateamento das arquiteturas de memória distribuída também tem auxiliado na difusão deste tipo de arquitetura, que tem-se tornado comum em grandes centros computacionais. Por essa razão, algoritmos escaláveis combinando os dois tipos de arquitetura devem ter foco crescente nos próximos anos.

As técnicas estudadas estão resumidamente classificadas na Tabela 3.2. O trabalho apresentado em [Kohout et al. 2005] foi implementado tanto para duas quanto para três dimensões e a técnica de [Chernikov e Chrisochoides 2006] tem versões para memória compartilhada e para memória distribuída. Portanto, esses métodos estão contados duas vezes nas Tabelas 3.2b e 3.2e, respectivamente.

Tabela 3.2: Classificação das técnicas estudadas.

(a) Tipo de Método.

Método	Métodos
Avanço de fronteira	7
Delaunay	5
Arbitrário	4
Total	16

(b) Dimensão.

Dimensão	Métodos
Duas	7
Três	9
Qualquer	1
Total	17

(c) Tipo de particionamento.

Particionamento	Métodos
Implícito	9
Explícito	7
Total	16

(d) Tipo de balanceamento de carga.

Balanceamento	Métodos
Estático	10
Dinâmico	6
Total	16

(e) Tipo de memória utilizada.

Memória	Métodos
Compartilhada	7
Distribuída	10
Total	17

(f) Grau de acoplamento.

Acoplamento	Métodos
Muito acoplado	2
Parcialmente acoplado	5
Desacoplado	9
Total	16

(g) Grau de escalabilidade.

Escalabilidade	Métodos
Escalável	8
Parcialmente escalável	7
Não escalável	1
Total	16

4 *Técnica Proposta*

4.1 Introdução

O objetivo principal deste trabalho é desenvolver uma técnica de geração de malha em paralelo. Ela 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]. Esta técnica sequencial, que será brevemente explicada na Seção 4.2, 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 triângulos seja explicitamente utilizada.

3. A malha deve fornecer uma boa transição entre regiões muito refinadas, com muitos triângulos pequenos, e regiões pouco refinadas, com poucos triângulos grandes.

4. A técnica deve tratar fronteiras com trincas, ou fraturas, e gerar malhas respeitando-as. As fraturas são descritas como regiões de área nula, ou seja, existem arestas/pontos diferentes, porém geometricamente coincidentes (Figura 4.1).

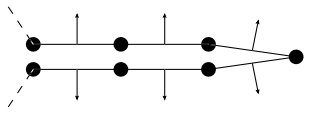


Figura 4.1: Exemplo de fratura. A distância entre vértices e arestas mostrada é apenas esquemática e não existe na realidade.

A técnica paralela foi projetada utilizando-se particionamento implícito, 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 particionamento explícito não satisfazem. Os

outros requisitos são atendidos pela própria técnica sequencial, que é empregada pela técnica paralela. A paralelização da técnica sequencial será descrita na Seção 4.3.

4.2 Técnica Sequencial

4.2.1 Introdução

A entrada para a técnica é uma discretização da fronteira de um objeto. Tal discretização é composta de arestas, na versão bidimensional da técnica, ou por triângulos, na versão tridimensional. A versão bidimensional da técnica será mais detalhada neste trabalho, mas sua correspondente em 3D é direta.

O primeiro passo da técnica sequencial (Figura 4.2) é gerar uma decomposição recursiva do domínio, uma *quadtree*. Esta servirá como guia, ou função densidade, para a inserção dos novos pontos. Posteriormente, baseado no tamanho das células da *quadtree*, os elementos triangulares são gerados, por avanço de fronteira. Ao final, são aplicadas estratégias de otimização na malha gerada, para melhorar a sua qualidade.

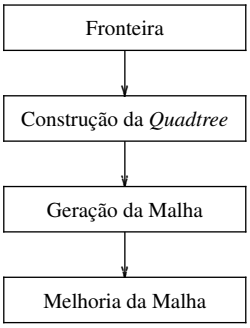
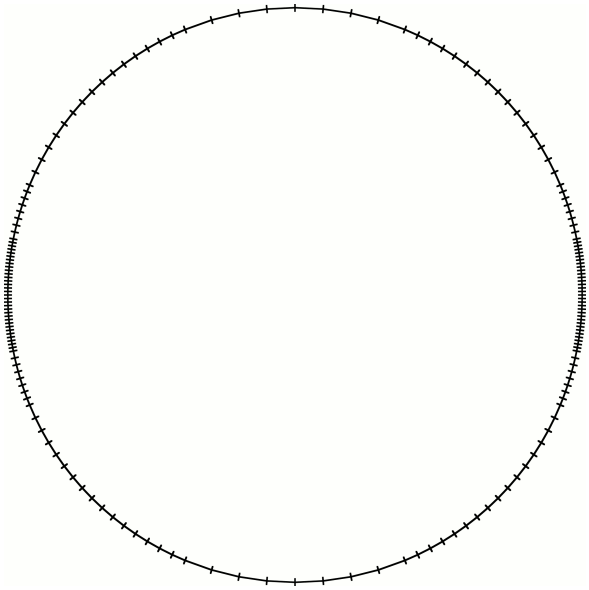


Figura 4.2: Funcionamento da técnica sequencial.

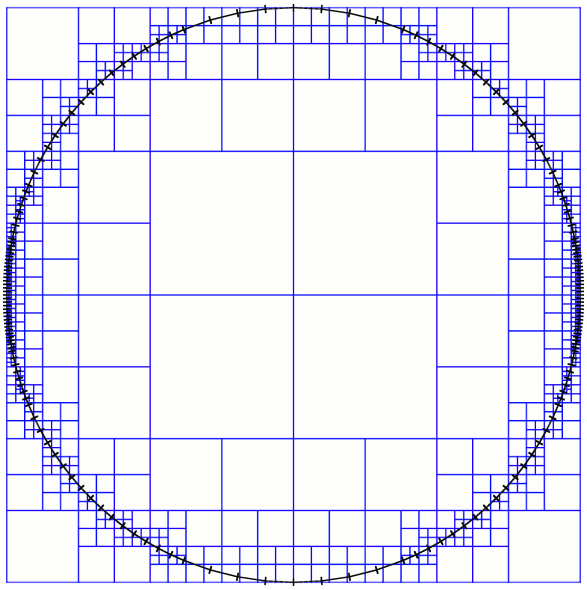
4.2.2 Construção da *Quadtree*

Inicialmente, um quadrado limitante é gerado, enquadrando toda a região do domínio. Esse quadrado é a célula-raiz da *quadtree*. Seleciona-se, então, uma aresta da fronteira. Após isso, a célula da *quadtree* que contém o ponto médio da aresta, inicialmente, a célula-raiz da *quadtree*, é subdividida, recursivamente, até que o seu tamanho (o tamanho de seu lado) seja menor que o comprimento da aresta, multiplicada de um fator. Este fator controla o refinamento da *quadtree* e, portanto, o tamanho dos triângulos gerados. Neste trabalho, o fator utilizado é de 0,85, que é uma aproximação para $\frac{\sqrt{3}}{2}$, a altura de um triângulo equilátero. Isso é feito para todas as

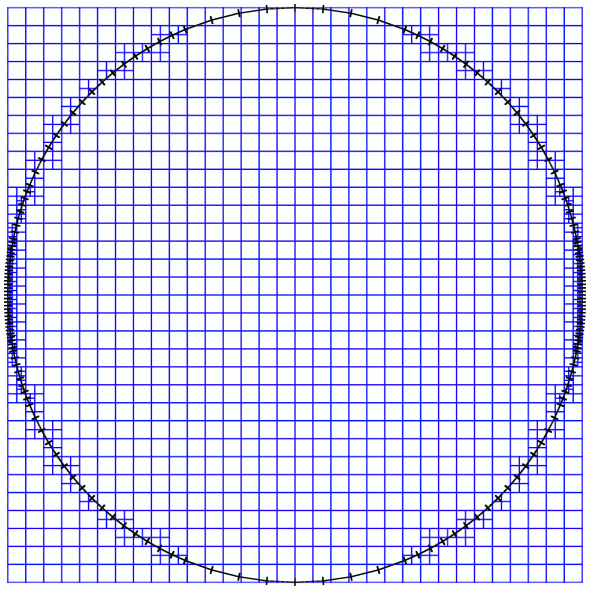
arestas da fronteira. As Figuras 4.3a e 4.3b mostram um exemplo de fronteira e sua *quadtree* gerada.



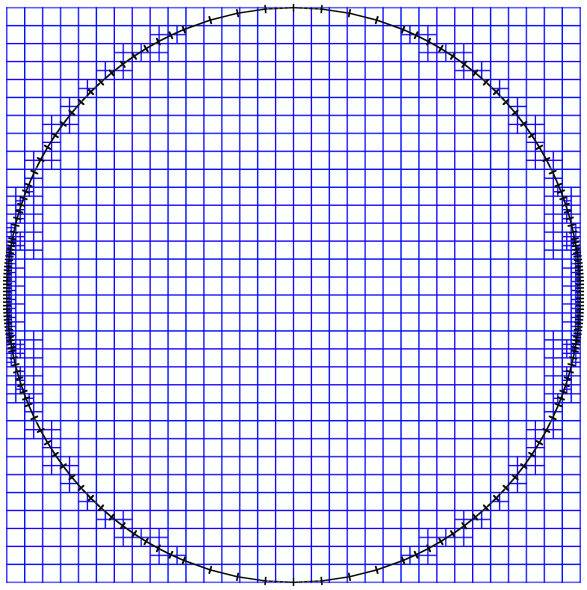
(a) Fronteira.



(b) *Quadtree* gerada.



(c) *Quadtree* após o primeiro refinamento.



(d) *Quadtree* após o segundo refinamento.

Figura 4.3: Fronteira e *quadtree* gerada.

Após gerar a *quadtree*, esta é refinada de duas formas diferentes. Primeiramente, a maior célula interna ao domínio não deve ser maior que a maior célula da fronteira. Isso é feito para que a malha não tenha elementos excessivamente grandes no interior (Figura 4.3c). 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 4.3d). Este refina-

mento, conhecido na literatura como refinamento 2:1, garante que a transição entre elementos pequenos e grandes seja suave.

4.2.3 Geração da Malha

O próximo passo da técnica é o de geração da malha. Este passo consiste de duas etapas, uma que gera elementos de boa qualidade, chamada de fase baseada em geometria, e outra que apenas gera elementos válidos, tentando gerar elementos da melhor qualidade possível, chamada de fase baseada em topologia. A fase baseada em topologia garante que uma malha seja totalmente gerada (em duas dimensões), mesmo que a malha não seja totalmente de boa qualidade. Isto não é um problema sério na maioria das aplicações, especialmente considerando-se que possíveis elementos com qualidade não tão boa estarão longe da fronteira original, que é o que se deseja normalmente.

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

A fase baseada em geometria inicia-se com a lista de arestas ativas contendo as arestas da fronteira, ordenadas crescentemente de acordo com seu comprimento, e com a lista de arestas rejeitadas vazia. Então, a aresta de menor comprimento é retirada da lista de arestas ativas (chamada de aresta base), e um ponto deve ser encontrado para gerar um novo triângulo. Caso tal ponto não seja encontrado, essa aresta é movida para a lista de arestas rejeitadas. Caso um ponto seja encontrado, um novo triângulo é formado, e a fronteira é atualizada, inserindo a(s) nova(s) aresta(s) na lista de arestas ativas ou removendo a(s) que já existia(m) previamente.

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

Para encontrar um ponto para uma aresta, e assim formar um novo triângulo, a fase baseada em geometria segue os seguintes passos:

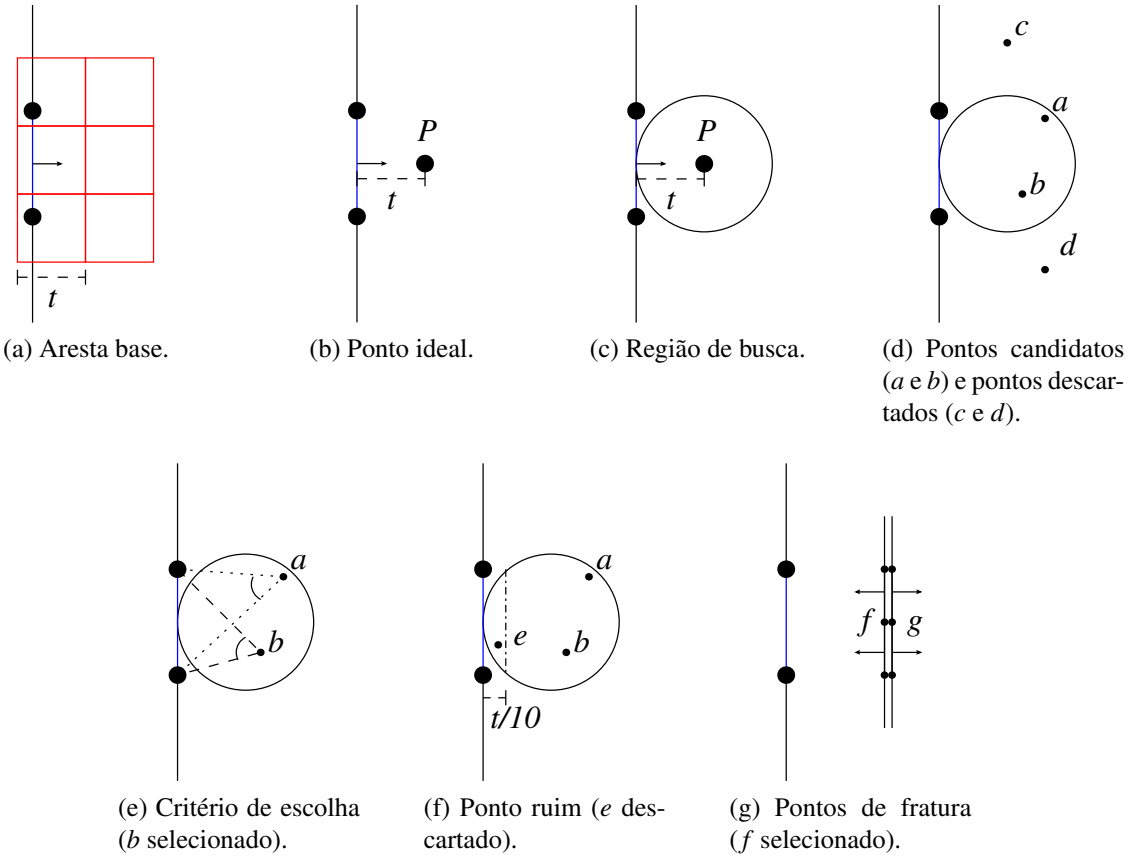
1. Encontre a célula da *quadtree* onde está localizado o ponto médio da aresta base, e pegue o seu tamanho t (Figura 4.4a).

2. Gere um ponto ideal P, passando pela mediatriz da aresta, cuja distância à aresta base seja t, e esteja no semi-plano definido pela normal da aresta (Figura 4.4b).

3. Crie uma região de busca circular centrado em P e de raio t (Figura 4.4c).

4. Ache todos os pontos da fronteira corrente que estejam dentro da região de busca (Figura 4.4d) e selecione o melhor ponto (ou seja, o ponto que tenha maior ângulo com os pontos da aresta base, Figura 4.4e) que forme um triângulo válido (isto é, que não cruze nenhuma aresta da fronteira corrente).

5. Caso nenhum ponto seja encontrado, teste se o triângulo formado pela aresta base e por P é válido. Se for válido, forme um novo elemento com esse ponto e com essa aresta base e atualize a fronteira. Se não for, rejeite esta aresta, selecione uma nova aresta base, e vote ao passo inicial.



(a) Aresta base. (b) Ponto ideal. (c) Região de busca. (d) Pontos candidatos (a e b) e pontos descartados (c e d).

(e) Critério de escolha (b selecionado). (f) Ponto ruim (e descartado). (g) Pontos de fratura (f selecionado).

Figura 4.4: Avanço de uma aresta da fronteira.

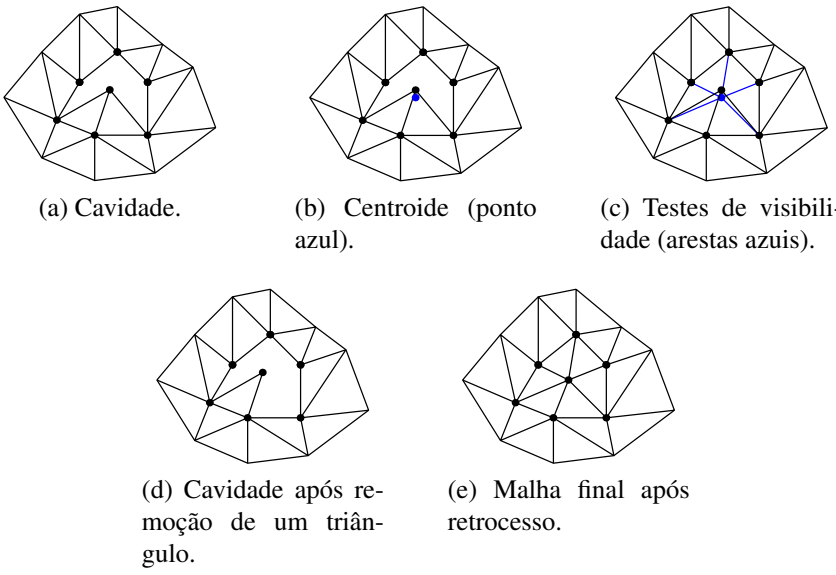
Durante a busca por pontos existentes, duas observações devem ser feitas:

- Os pontos que estiverem à distância de até $\frac{t}{10}$ da aresta base são descartados, pois estes formariam triângulos de má qualidade (Figura 4.4f).

- Quando dois pontos fizerem parte de uma fratura, caso em que terão mesmas coordenadas, o ponto dentre eles a ser selecionado deve ser aquele cujas arestas adjacentes, pertencentes à fronteira original, tenham normais apontando para a aresta base (Figura 4.4g).

A fase baseada em topologia ocorre somente se uma aresta for rejeitada duas vezes, a primeira quando estava na lista de arestas ativas e a segunda quando estava na lista de arestas rejeitadas, de forma que uma malha ainda não foi totalmente gerada para aquela fronteira. Assim como na fase baseada em geometria, esta fase também utiliza duas listas, a lista de arestas ativas, inicializada como a lista de arestas rejeitadas da fase passada, e a lista de arestas rejeitadas, inicialmente vazia. A diferença entre as duas fases é que esta fase não mais se preocupa com a forma dos triângulos gerados. Assim, todos os pontos que estiverem no semi-plano definido pela normal da aresta base são candidatos a formar um novo triângulo, e o ponto que formar o maior ângulo com a aresta base é selecionado, desde que o novo triângulo seja válido. Quando essa fase estiver terminada, uma malha foi gerada para a fronteira dada.

Na versão tridimensional da técnica, entretanto, é possível que uma malha não tenha sido gerada na fase anterior. A fronteira corrente consiste de 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 fase chamada de geração de malhas por retrocesso (*back-tracking*). Uma esquematização do procedimento, para o caso bidimensional, pode ser observada na Figura 4.5.



(a) Cavidade. (b) Centroide (ponto azul). (c) Testes de visibilidade (arestas azuis).

(d) Cavidade após remoção de um triângulo. (e) Malha final após retrocesso.

Figura 4.5: Geração de malhas por retrocesso.

Para uma determinada cavidade (Figura 4.5a), um ponto é tentativamente inserido no seu centroide (Figura 4.5b), e testes de visibilidade são feitos gerando-se arestas entre o ponto criado

e os pontos da fronteira da cavidade (Figura 4.5c). Caso um triângulo intercepte alguma(s) dessas arestas, este triângulo é removido, aumentando-se a cavidade (Figura 4.5d), e tentando-se criar um novo ponto no centroide, voltando ao início.

O triângulo removido é aquele que interceptar o maior número de arestas, sendo assim aquele que mais atrapalha a visibilidade. Além disso, triângulos cujas três arestas pertençam à fronteira da cavidade também são removidos. Se um triângulo for adjacente à fronteira original, então não é removido pois, como foi criado primeiramente, é o de melhor qualidade para a aresta da fronteira inicial. Mesmo com esta etapa, entretanto, é possível 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. Entretanto, apesar de poder acontecer, isso não foi observado em nenhum exemplo testado, pois foi sempre possível gerar uma malha válida, mesmo para geometrias bem complexas.

4.2.4 Melhoria da Malha

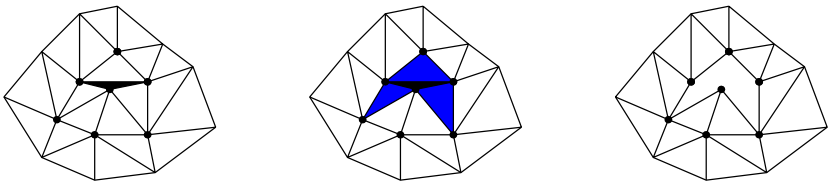
A malha gerada passa ainda por duas etapas de melhoramento, uma de suavização de vértices, e outra de otimização de triângulos, aplicadas alternadamente, uma determinada quantidade de vezes. No caso desta implementação, foram aplicadas 5 vezes, conforme é sugerido na literatura.

A suavização de vértices, conhecida como suavização de Laplace (referente a Pierre-Simon Laplace, matemático francês), 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 4.1.

$$X_O^{n+1} = X_O^n + \phi \frac{\sum_{i=1}^{m} (X_i^n - X_O^n)}{m} \tag{4.1}$$

Nessa equação, X_O^n é a posição do vértice O na iteração de suavização n, m é o número de vértices adjacentes a O por alguma aresta e ϕ é o parâmetro de relaxamento, normalmente ajustado para um valor entre 0 e 1 (neste trabalho, o valor de $0,5$ foi utilizado para ϕ). Tal suavização é aplicada iterativamente em todos os vértices internos da malha.

A otimização consiste em remover triângulos classificados como ruins (Figura 4.6a), juntamente com os triângulos adjacentes (Figura 4.6b), criando uma cavidade (Figura 4.6c). Assim, uma estratégia parecida com a geração por retrocesso pode ser empregada para gerar uma malha interna (Figura 4.5). A otimização pode ser aplicada tanto em duas quanto em três dimensões.



(a) Triângulo ruim (preto). (b) Triângulos adjacentes (azuis). (c) Cavidade.

Figura 4.6: Remoção de elementos e geração de cavidade.

Na otimização, após determinar-se que um ponto pode ser inserido no centroide da cavidade e, portanto, novos triângulos podem ser criados, há um teste de qualidade para determinar se todos os novos elementos são melhores que o pior elemento removido. Caso sejam, os triângulos antigos são substituídos, havendo de fato uma melhora na qualidade da malha. Caso não sejam, os triângulos antigos são restaurados, impedindo que uma malha de pior qualidade substitua uma de melhor qualidade já existente.

Observa-se ainda que, se um número alto de elementos tiver que ser removido, então a otimização não é realizada e a malha antiga é restaurada. Isso é feito para que não aconteça de muitos triângulos serem criados compartilhando um mesmo vértice, o que pode ser ruim para a malha como um todo, e para que este procedimento seja local e não demande muito tempo de execução. O limite para esse número máximo de elementos que podem ser removidos é definido empiricamente, mas segue algumas restrições de largura de banda em aplicações de elementos finitos e de cálculo de normais em computação gráfica. Nesse trabalho, foi adotado o número de 10 elementos.

4.2.5 Resultados

Esta técnica gera malhas de boa qualidade, para fronteiras contínuas ou descontínuas, com ou sem fraturas. 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 2.3.1. Vários exemplos podem ser encontrados nas referências anteriormente mencionadas.

4.3 Técnica Paralela

4.3.1 Introdução

A técnica proposta neste trabalho pode ser adequada a diversas arquiteturas paralelas, de memória compartilhada, distribuída ou híbrida, possuindo processadores de velocidades iguais ou diferentes. Dessa forma, esta técnica pode ser utilizada em um único computador de dois núcleos, em diferentes computadores conectados por uma rede local, em um *cluster* com rede de alta velocidade, ou qualquer outro ambiente desejado pelo usuário.

Esta técnica paralela segue o modelo de paralelização mestre/escravo, que se adequa facilmente a diferentes arquiteturas paralelas. Neste modelo, um processador especial, o processador mestre, é responsável pela leitura das entradas (a fronteira e os parâmetros de geração), pela distribuição do trabalho entre os diversos processadores, e pela escrita da saída (a malha). Já os processadores restantes, chamados de escravos, ficam responsáveis exclusivamente pelo trabalho pesado, ou seja, a geração da malha.

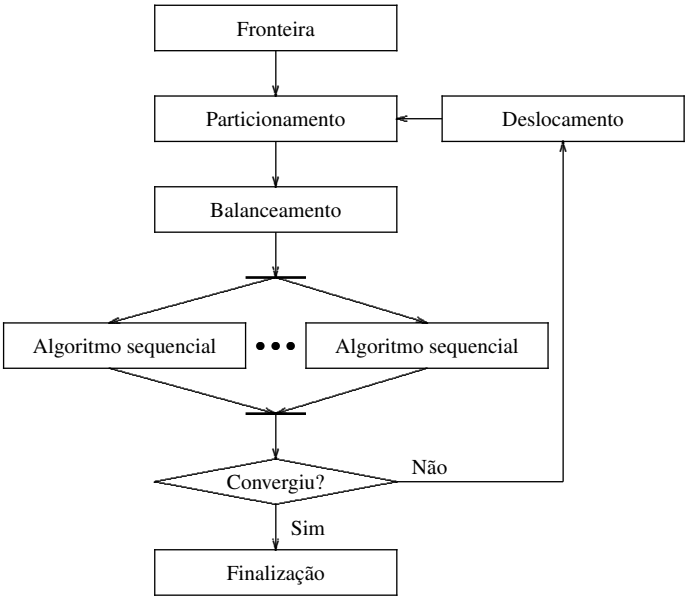
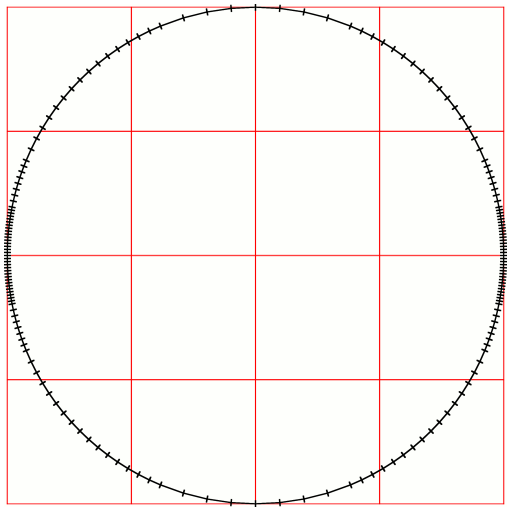
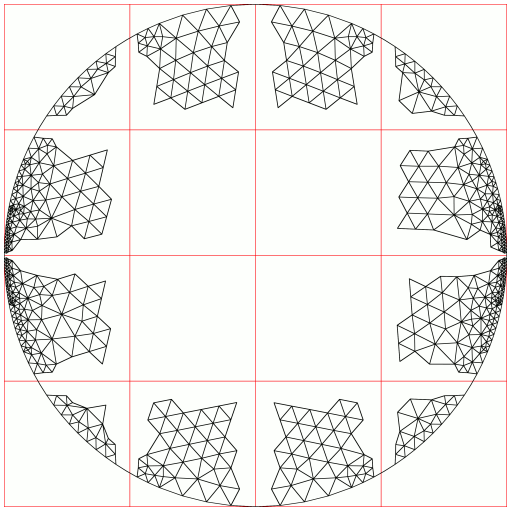


Figura 4.7: Funcionamento da técnica paralela.

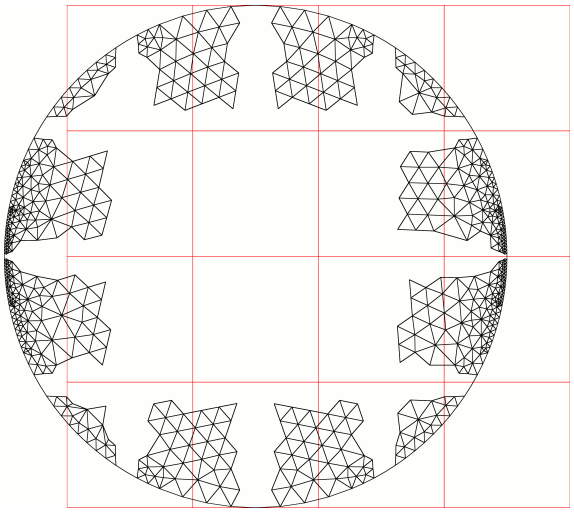
A Figura 4.7 mostra como a técnica paralela implícita procede. Dada uma fronteira, a técnica particiona o domínio na forma de uma *quadtree*, chamada de *quadtree* de particionamento neste trabalho (Figura 4.8a), não tão refinada quanto a *quadtree* utilizada pela técnica sequencial, que, deste ponto em diante, será chamada de *quadtree* de densidade. Uma vez criada essa *quadtree* de particionamento, suas células-folha serão as partições, e as que estiverem na fronteira serão distribuídas entre os processadores envolvidos. Assim, malhas serão geradas de forma a não cruzarem as bordas dessas células (Figura 4.8b).



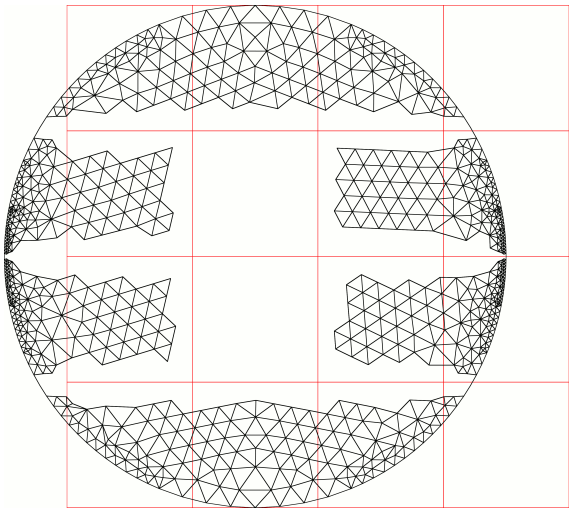
(a) *Quadtree* de particionamento do domínio.



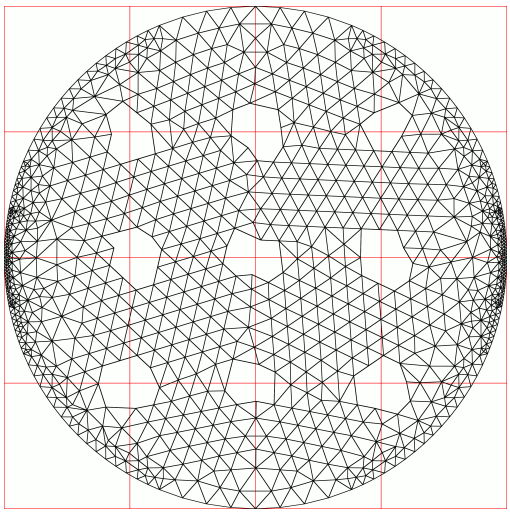
(b) Malhas geradas em cada partição.



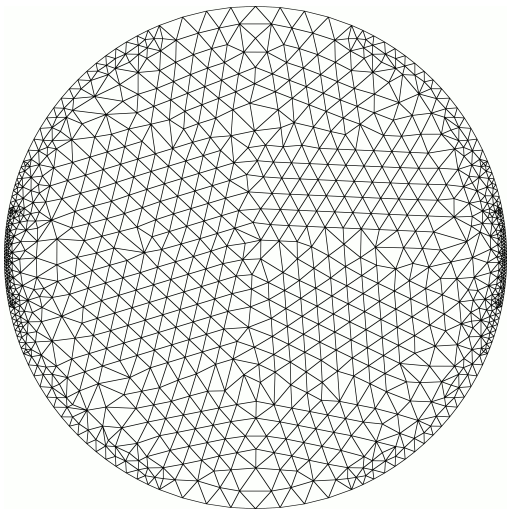
(c) Deslocamento para a direita.



(d) Geração de mais malhas.



(e) Cavidades restantes.



(f) Malha final (sem melhoria).

Figura 4.8: *Overview* do procedimento paralelo.

Uma vez geradas as malhas, a nova fronteira será encontrada e as células da *quadtree* de particionamento sofrerão um deslocamento em alguma direção cartesiana, horizontal ou vertical (Figura 4.8c). Então, a fronteira é particionada entre as células deslocadas, que serão novamente distribuídas entre os processadores para gerarem mais malhas (Figura 4.8d). Esse procedimento de deslocamento segue-se em direções e sentidos alternados, até que mais nenhum triângulo possa ser gerado. Posteriormente, as cavidades restantes, se existirem (Figura 4.8e), serão preenchidas sequencialmente, e a malha estará completamente gerada (Figura 4.8f).

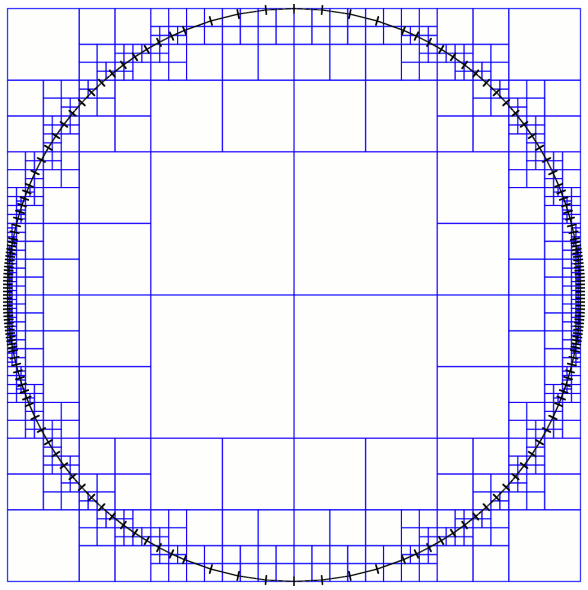
4.3.2 Construção da *Quadtree* de Densidade

Na técnica paralela, a construção da *quadtree* de densidade é feita de forma ligeiramente diferente da da técnica sequencial (Seção 4.2.2, Figuras 4.3b a 4.3d). Após gerar a *quadtree* de densidade inicial (Figura 4.9a), as suas células são classificadas como internas, na borda, ou externas à fronteira original (Figura 4.9b). Assim, os dois refinamentos posteriores não são aplicados às células de fora do objeto (Figuras 4.9c e 4.9d). A razão disso é a economia de memória, uma vez que a fronteira, geralmente, é bastante discretizada, gerando uma *quadtree* de densidade também bastante discretizada, com uma quantidade muito grande de células fora do objeto, desperdiçando memória.

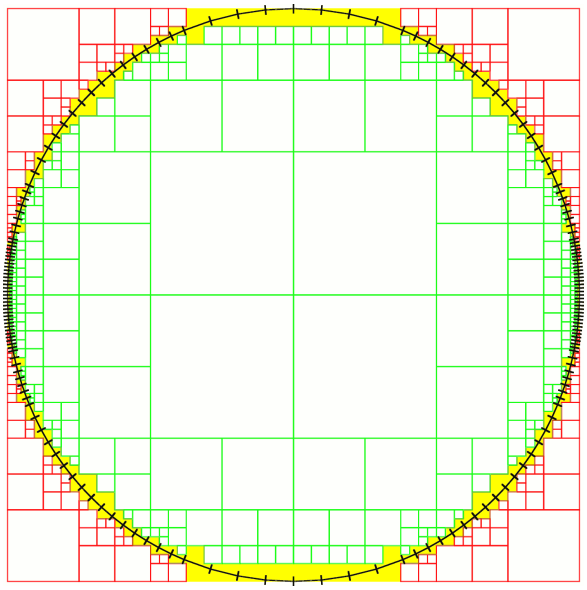
A classificação é feita da seguinte maneira:

- Identifique as células da fronteira, ou seja, aquelas que cruzam pelo menos uma das arestas da fronteira.
- Encontre uma célula que não tenha sido classificada.
- Encontre o ponto médio dessa célula e, pelo algoritmo do ângulo [Carvalho e Figueiredo 1991], determine se ele está dentro ou fora da fronteira.
- Aplique, nessa célula, um algoritmo parecido com o preenchimento de regiões por *flood-fill* [Hearn e Baker 1996, Angel 2008] para classificar as outras células:
  - Encontre uma célula vizinha não classificada.
  - Classifique-a com a mesma classificação da original.
  - Aplique este algoritmo recursivamente a essa célula.

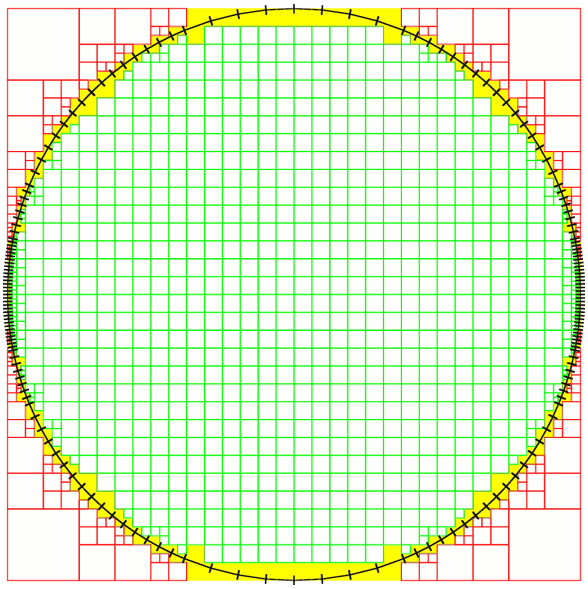
O algoritmo de *flood-fill* termina, pois as células da fronteira foram todas previamente classificadas.



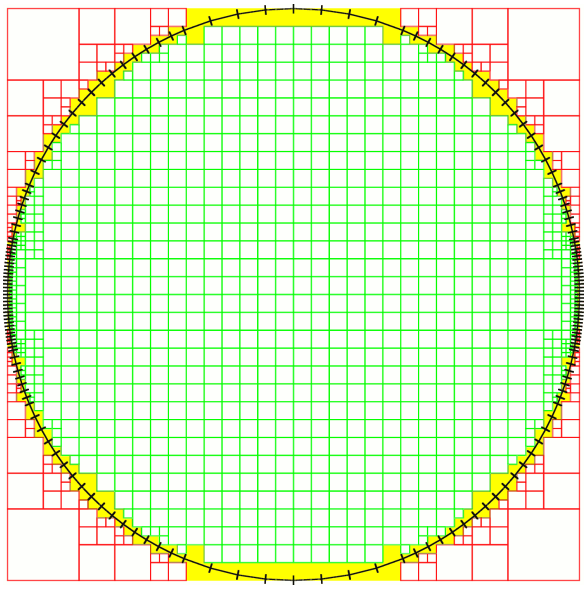
(a) *Quadtree* de densidade inicial.



(b) *Quadtree* de densidade inicial classificada.



(c) *Quadtree* de densidade após o primeiro refinamento.



(d) *Quadtree* de densidade após o segundo refinamento.

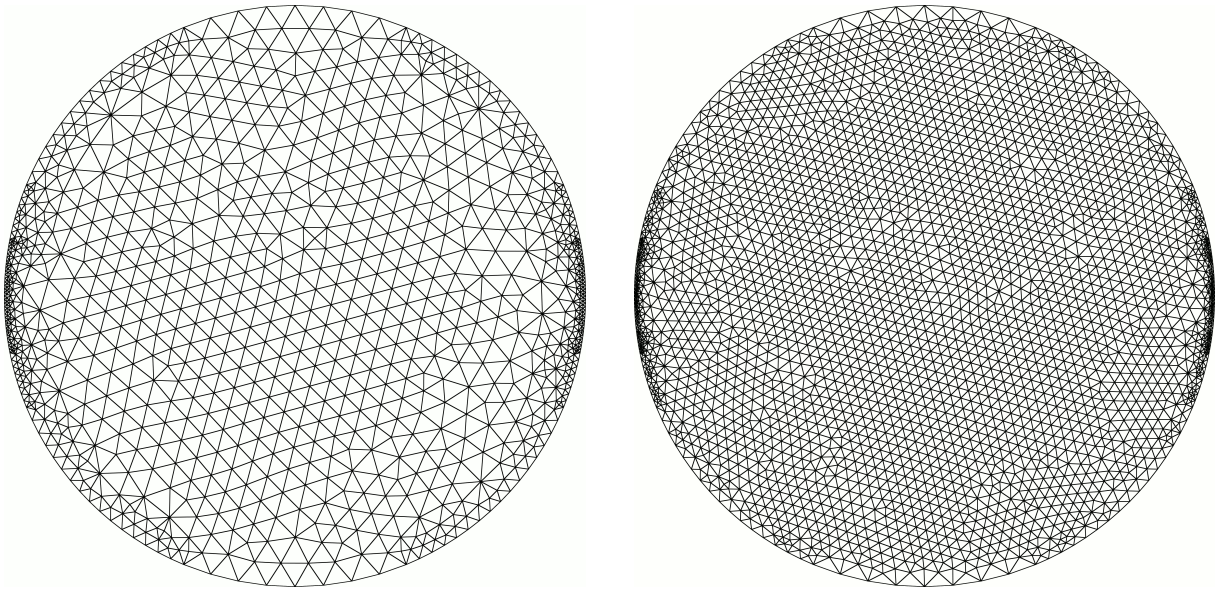
Figura 4.9: *Quadtree* de densidade (células verdes - dentro do domínio, células vermelhas - fora do domínio, células amarelas - sobre a fronteira).

Além disso, os dois refinamentos da *quadtree* de densidade (Figuras 4.9c e 4.9d) são feitos utilizando paralelismo de memória compartilhada, quando disponível pela arquitetura, de forma que cada célula filha da raiz fica em um processador diferente. Apesar de poder ser feita facilmente, essa divisão de trabalho pode levar a desbalanceamentos durante a execução dos refinamentos, o que será tratado adequadamente na sequência do algoritmo.

4.3.3 Cálculo da Carga

Como dito na Seção 2.2.2, a carga, em algoritmos de computação de alto desempenho, é um valor que tenta predizer a quantidade de computação a ser processada por uma determinada partição. Em técnicas de geração de malhas, portanto, a carga deve refletir a quantidade de elementos gerados. Dessa forma, ela deve levar em conta:

1. 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 4.10).



(a) Malha menos refinada. (b) Malha mais refinada.

Figura 4.10: Comparação entre malhas refinada e não refinada. Perceba que a mesma fronteira foi utilizada para as duas malhas.

2. Em regiões com discretização uniforme, uma sub-região de maior área deve gerar mais elementos que uma de menor área (Figura 4.11).

3. 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 4.12a). 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 4.12b).

A *quadtree* de densidade (Figura 4.3d) reúne todas essas características, pois é ela que define o tamanho dos elementos gerados, tanto em regiões próximas à fronteira quanto em

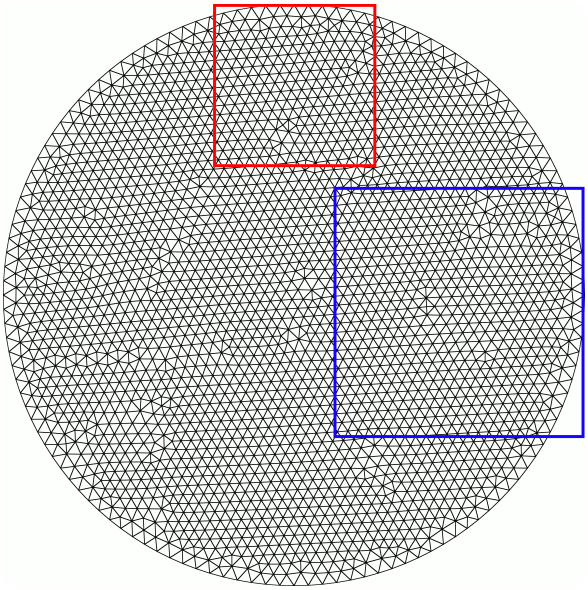
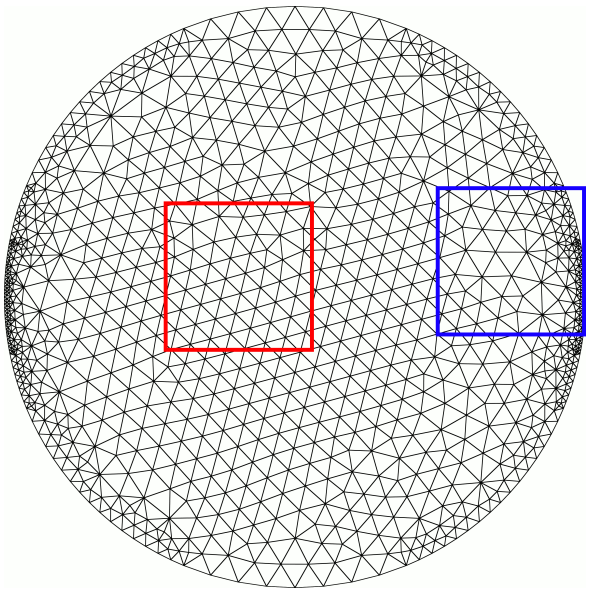
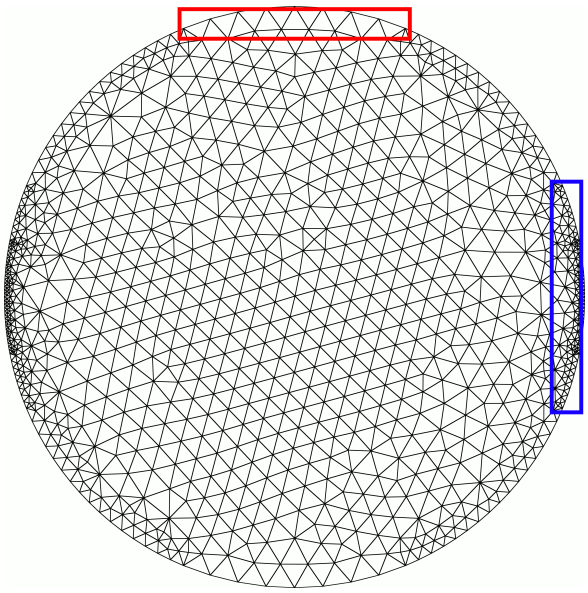


Figura 4.11: Fronteira com discretização uniforme (azul - área maior, vermelha - área menor).



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



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

Figura 4.12: Regiões de transição.

regiões internas, considerando ainda variações suaves de tamanhos. Portanto, indiretamente, essas células predizem a quantidade de elementos que serão gerados. Dessa forma, a carga total calculada neste trabalho é determinada como a quantidade de células da *quadtree* de densidade que não foram classificadas como externas à fronteira (Figura 4.13).

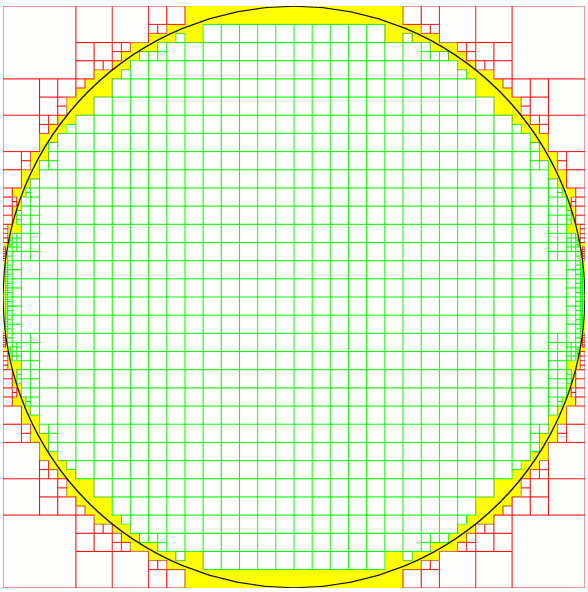


Figura 4.13: Classificação das células da *quadtree* de densidade (vermelha - externa, verde - interna, amarela - na borda).

4.3.4 Decomposição do Domínio

O domínio, ou seja, a fronteira e o seu interior, serão decompostos de acordo com uma nova *quadtree*, a *quadtree* de particionamento (reveja a Figura 4.8a), cuja geração baseia-se na carga. Inicialmente, um quadrado é construído circunscrevendo o domínio (a célula raiz da *quadtree* de particionamento), e sua carga é atribuída como a carga total calculada. Se a carga desta célula for maior que uma determinada carga máxima, dada pelo usuário ou calculada como a carga total dividida pela quantidade de escravos, esta célula é subdividida. Esta verificação é feita recursivamente para as células-folha da *quadtree* de particionamento, subdividindo uma célula até que sua carga seja menor que a carga máxima.

A carga de cada célula da *quadtree* de particionamento é a quantidade de células da *quadtree* de densidade internas a ela (Figura 4.14) e não externas ao domínio. Nesta *quadtree* de particionamento, também é aplicado o refinamento 2:1 (Seção 4.2.2), a fim de facilitar o procedimento de deslocamento das células, que será descrito na Seção 4.3.7. Cada célula-folha da *quadtree* de particionamento que cruzar ou que estiver no interior do objeto será, portanto, uma partição do domínio.

As arestas da fronteira são divididas entre as partições existentes. Para cada aresta, verifica-se se ela está totalmente interna a uma partição, ou seja, se seus dois pontos extremos estão internos ao quadrado da partição (Figura 4.15a). Caso isso aconteça, essa aresta pertence exclusivamente a essa partição. Uma aresta pode ainda pertencer a várias partições, se ela estiver cruzando (ou apenas tocando) as bordas dos quadrados (Figura 4.15b).

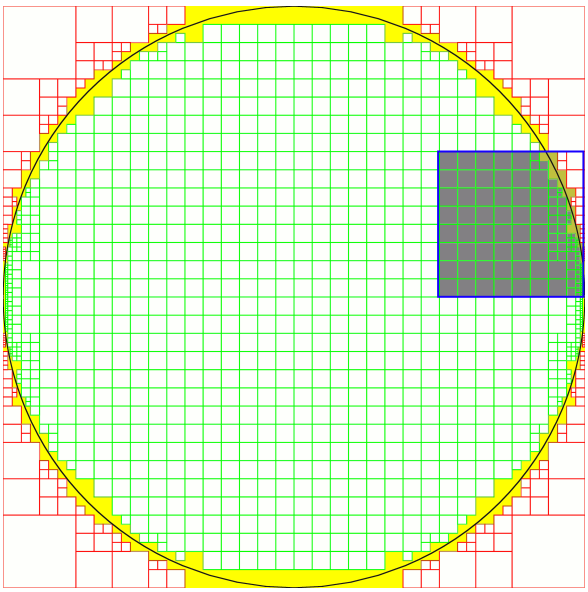
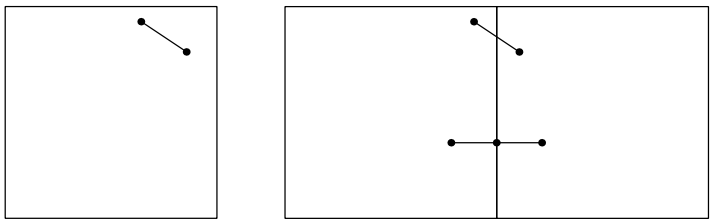


Figura 4.14: Carga de uma partição - quantidade de células internas ou na borda (partição em azul, células consideradas para a carga em destaque).



(a) Aresta exclusiva. (b) Arestas compartilhadas.

Figura 4.15: Classificação das arestas.

Pode ainda acontecer de nenhuma partição ser encontrada para uma determinada aresta, no caso em que a *quadtree* de particionamento está deslocada (Seção 4.3.7). É possível, ainda, que algumas células da *quadtree* de particionamento deslocada se tornem retangulares, deixando de ser quadradas. Portanto, o teste de pertinência de arestas é feito independentemente do formato da partição, seja ele um retângulo ou um quadrado.

Para determinar se uma aresta cruza um retângulo, utiliza-se uma técnica de *clipping* [Hearn e Baker 1996, Angel 2008], que é um problema bem conhecido e estudado em computação gráfica.

4.3.5 Balanceamento de Carga

Uma vez determinadas as partições do domínio, estas devem ser distribuídas entre os diversos processadores para que as sub-malhas sejam geradas. Essa distribuição deve ser de tal forma que

cada processador gaste aproximadamente o mesmo tempo, para que não haja nem processadores ociosos nem processadores sobrecarregados.

Neste trabalho, as partições são enviadas aos processos sob demanda. Inicialmente, o processo mestre retém todas as partições. Um processo escravo requisita uma partição ao mestre e, assim que receber a partição, executa o seu trabalho de geração de malha (Seção 4.3.6), enquanto o mestre fica esperando a requisição de outro (que pode ser o mesmo) escravo. Assim que um escravo terminar sua execução, este requisita uma nova partição, e assim por diante.

No caso de um processo escravo ter diversas *threads* rodando concorrentemente, a comunicação entre uma delas e o processador mestre é feita de maneira exclusiva, ou seja, não é possível que duas *threads* se comuniquem ao mesmo tempo com o processador mestre. Isso é feito por que a maioria das implementações da biblioteca utilizada para passagem de mensagens não é segura com relação a *threads*, ou seja, existe um mesmo *buffer* de envio/recebimento de mensagens, que não deve ser modificado simultaneamente por duas ou mais *threads*, para não haver perda de dados.

É possível ainda acontecer de muitos processos escravos quererem se comunicar com o processo mestre ao mesmo tempo, sendo necessária a formação de uma fila de espera (fila implícita, tratada pela própria biblioteca de comunicação inter-processos). Este gargalo pode levar a degradações de desempenho para um grande número de processos.

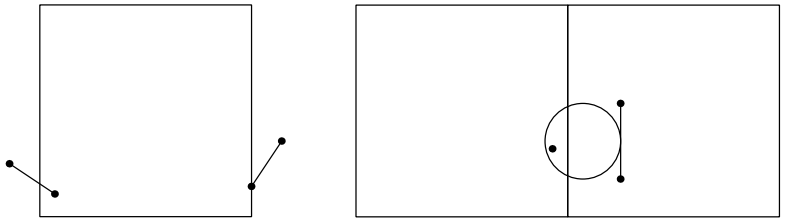
Uma vez que todas as partições do mestre tiverem sido enviadas aos escravos, o mestre entra em espera pelos resultados.

4.3.6 Geração da Malha

Antes de gerar malha nas partições atribuídas, cada processador escravo constrói a *quadtree* de densidade para todo o domínio (Figura 4.9d, Seção 4.3.2). Comparada com a malha gerada, a *quadtree* de densidade ocupa pouco espaço na memória, além de possibilitar o recebimento de qualquer partição sem a necessidade de receber a parte da árvore dentro dela, diminuindo o volume de comunicação com o processador mestre. Outra possibilidade seria passar para cada processador somente a parte da árvore de interesse, mas isso, além de complexo, poderia aumentar consideravelmente o volume de comunicação com o mestre, degradando o sistema.

Essa árvore é construída uma única vez em cada processador e é utilizada por todas as suas *threads*. Assim, se alguém for utilizar, por exemplo, um computador com 4 núcleos, é melhor executar um processo e ativar todas as 4 *threads* do que executar 4 processos, cada um com uma única *thread*.

Determinadas as partições de um processo, este deve gerar uma sub-malha para cada partição, aplicando a técnica sequencial brevemente descrita na Seção 4.2, excetuando-se a construção da *quadtree* de densidade, feita anteriormente. Algumas modificações foram feitas nessa técnica para certificar que os triângulos gerados estejam completamente contidos na região retangular definida pela partição.



(a) Arestas cruzando a partição (não avançam).

(b) Aresta com região de busca cruzando a partição (não avançam).

Figura 4.16: Restrições no avanço de fronteira.

1. Tanto na fase baseada em geometria quanto a fase baseada em topologia, as arestas que cruzam o retângulo definido por uma partição não podem participar o avanço de fronteira (Figura 4.16a).

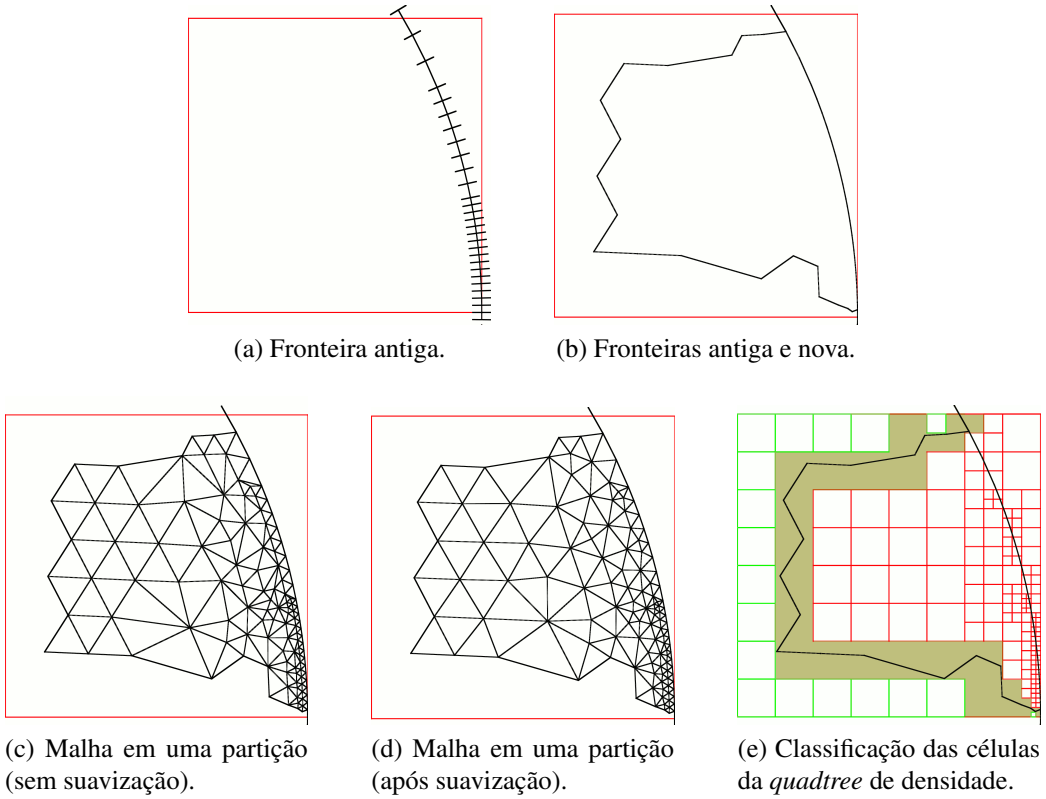
2. Ainda durante o avanço da fronteira, uma aresta cuja região de busca circular (Figura 4.4c) cruzar o retângulo definido pela partição também não pode avançar (Figura 4.16b). Quando isto acontece, um vértice que formaria um triângulo bom poderia estar em uma partição vizinha, que não seria encontrado localmente no processador. Algumas observações devem ser feitas:

   - Como a restrição é que nenhum triângulo formado cruze o retângulo, poderia-se mudar o teste para que somente o ponto ideal de uma aresta não esteja fora do retângulo, em vez de toda a região de busca. No entanto, isso poderia levar à geração de elementos muito ruins entre duas partições em passos posteriores, uma vez que o espaço livre para a criação dos triângulos seria consideravelmente reduzido.
   - Exemplos práticos mostraram que a fase baseada em topologia também deve obedecer a essa restrição. Portanto, essa fase deve saber como construir a região de busca da fase geométrica para decidir se ela cruza ou não a borda da partição, mesmo que não a utilize para a geração dos elementos.

3. Após avançar o máximo que puder, as arestas remanescentes formam a nova fronteira (Figura 4.17b), que será futuramente enviada para o processador mestre. Esta nova fronteira

não pode ser modificada pela etapa de suavização/otimização, bem como a fronteira antiga, recebida do processador mestre (Figuras 4.17c e 4.17d). Isto ocorre porque algumas informações de vizinhança de vértices ou triângulos, necessárias para a melhoria, estão ausentes ou ainda não foram determinadas nesse processo escravo. Portanto:

- A suavização não é aplicada nos vértices dessas fronteiras.
- A otimização é rejeitada caso essas fronteiras sejam alcançadas, como se suas arestas pertencessem à fronteira original.



(a) Fronteira antiga.

(b) Fronteiras antiga e nova.

(c) Malha em uma partição (sem suavização).

(d) Malha em uma partição (após suavização).

(e) Classificação das células da *quadtree* de densidade.

Figura 4.17: Fronteira, malha, e classificação das células da *quadtree* de densidade de uma partição.

Além da geração da malha, o processo escravo tem, ainda, informações suficientes para determinar a posição das células da *quadtree* de densidade que estiverem dentro da partição, se estão dentro, fora ou sobre o novo domínio. Isso é feito da seguinte maneira:

1. A fronteira antiga (que foi recebida do mestre) e a nova fronteira (que será enviada ao mestre) são identificadas e armazenadas em duas listas distintas (Figura 4.18).
2. As arestas que estiverem presentes nas duas listas ao mesmo tempo são as que cruzam o retângulo, e são movidas para uma terceira lista.

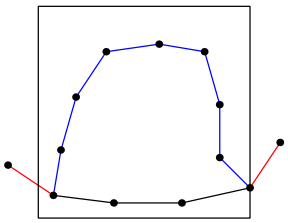


Figura 4.18: Fronteiras de uma partição (preta - fronteira recebida, azul - fronteira após a geração da malha, vermelhas - arestas que cruzam a partição).

3. As arestas remanescentes nas duas listas formam um ou mais polígonos (Figura 4.17b), contra o(s) qual(is) serão testadas as células da *quadtree* de densidade que estiverem dentro do retângulo:

   - Se a célula da *quadtree* de densidade cortar alguma aresta que cruza o retângulo (arestas vermelhas na Figura 4.18) ou alguma aresta da nova fronteira (arestas azuis na Figura 4.18), então essa célula é classificada como 'sobre' a fronteira, caso ela seja folha da *quadtree* de densidade. Se não for folha, os filhos são classificados.
   - Se a célula cortar a fronteira antiga, então ela é classificada como 'fora' do domínio, caso seja folha da *quadtree* de densidade. Se não for folha, os filhos são classificados.
   - Se a célula estiver dentro do polígono, então ela é classificada como 'fora' do domínio, pois o polígono define a região onde a malha foi gerada.
   - Se a célula estiver fora do polígono então ela é classificada como 'dentro' do domínio.

Nesse processo, algumas células fora do domínio são "erroneamente" classificadas como 'dentro', algo que será corrigido no processador mestre, resultando na classificação da Figura 4.17e.

Ao final, um processador escravo envia ao mestre os resultados de cada partição, ou seja, a nova fronteira (Figura 4.17b) e a nova classificação das células da *quadtree* de densidade (Figura 4.17e). A malha gerada (Figura 4.17d) é mantida no próprio processador escravo.

4.3.7 Deslocamento

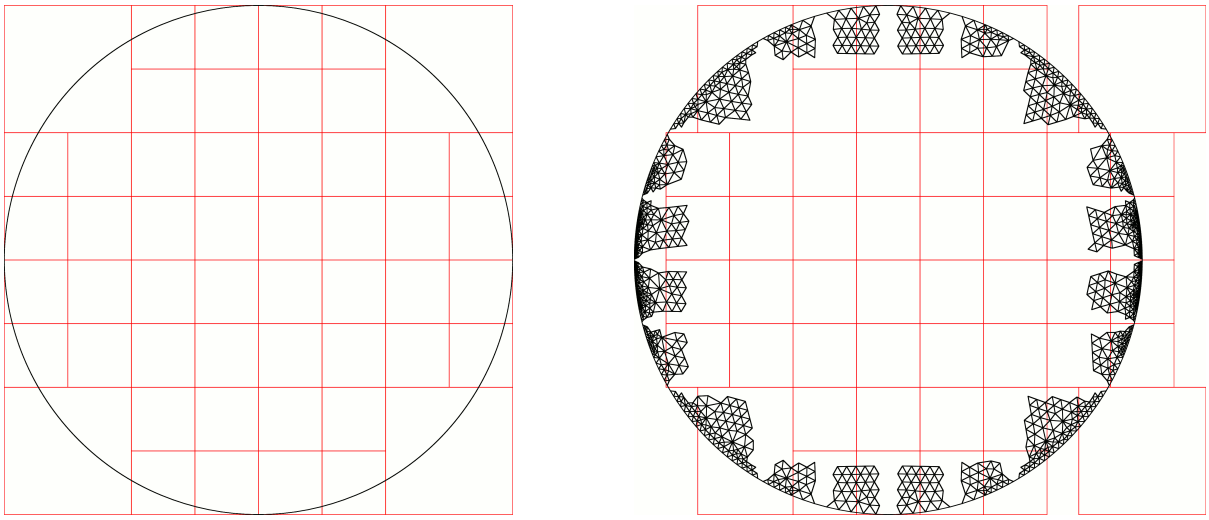
Após enviar as partições para os processadores escravos, o processador mestre entra em espera pelos resultados. Uma vez que esses tenham sido recebidos, para cada partição, o processador mestre faz o seguinte procedimento:

1. Atualiza a classificação das células da *quadtree* de densidade, ignorando as células previamente classificadas como ‘fora’ do domínio.

   - Esse teste simples corrige a classificação “errônea” feita pelos escravos.
   - No caso de fraturas ocorrerem na borda do retângulo da partição, algumas células inicialmente classificadas como ‘sobre’ podem passar a ser classificadas como ‘dentro’. Como isso é impossível de acontecer na realidade (algo que era ‘sobre’ virar ‘dentro’), esses casos são classificados, no mestre, como ‘fora’.

2. Atualiza a fronteira, inserindo vértices e arestas da nova fronteira, removendo vértices e arestas da fronteira antiga e deixando vértices e arestas que não participaram de nenhum avanço.

3. Desloca a *quadtree* de particionamento, ou seja, cada célula-folha da *quadtree* de particionamento sofre um deslocamento de metade de seu tamanho em uma direção cartesiana (Figura 4.19b).

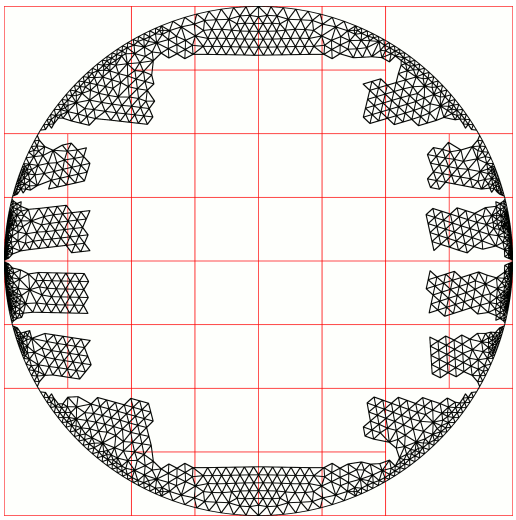


(a) *Quadtree* de particionamento.

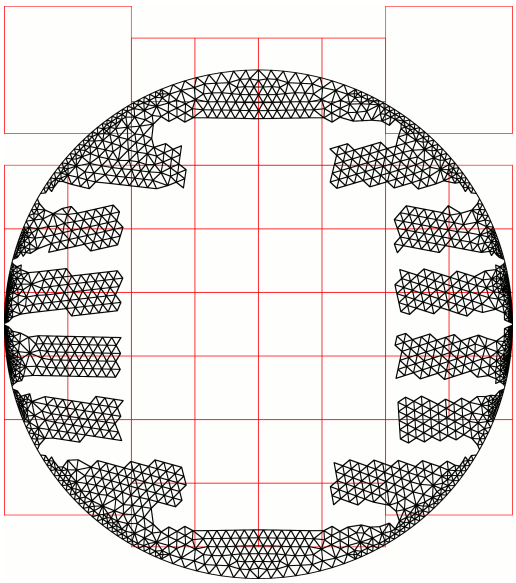
(b) *Quadtree* de particionamento deslocada para a direita.

Figura 4.19: *Quadtree* de particionamento e seus deslocamentos.

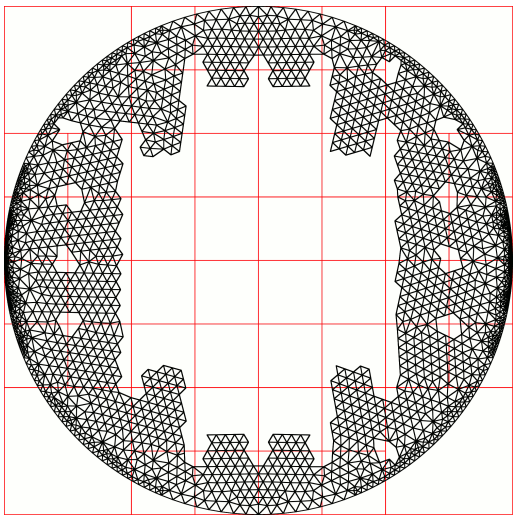
   - Como consequência desse deslocamento, é possível gerar malha entre duas partições da *quadtree* de particionamento na posição original, avançando a fronteira para uma partição vizinha (Figura 4.19c).
   - Caso a *quadtree* de particionamento já esteja deslocada, ela é movida de volta para a posição original (Figura 4.19c).



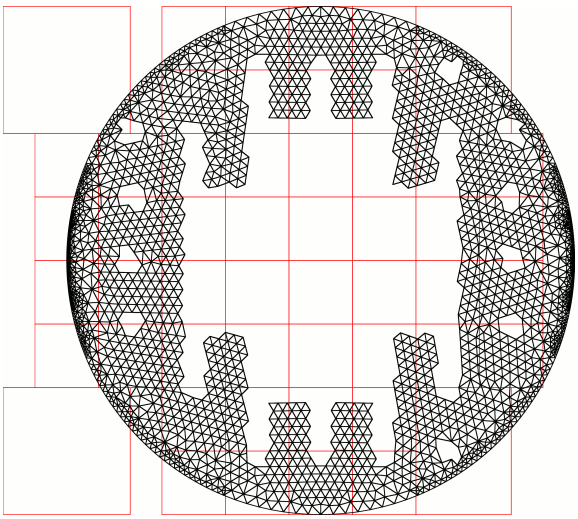
(c) *Quadtree* de particionamento na posição original.



(d) *Quadtree* de particionamento deslocada para cima.



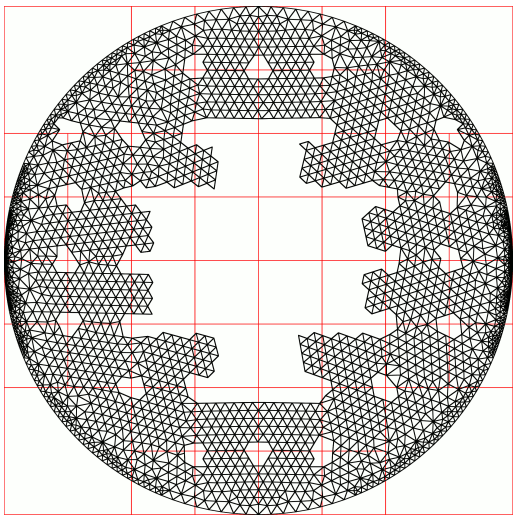
(e) *Quadtree* de particionamento na posição original.



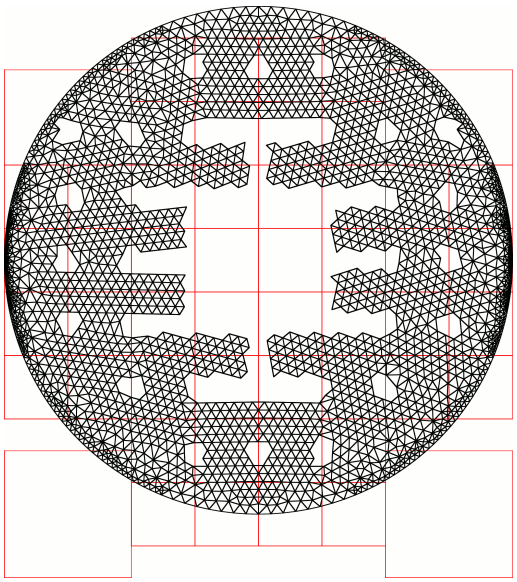
(f) *Quadtree* de particionamento deslocada para a esquerda.

Figura 4.19: *Quadtree* de particionamento e seus deslocamentos (continuação).

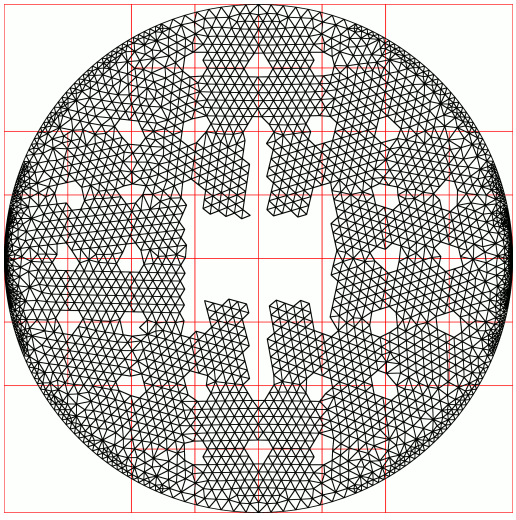
- Caso uma célula-folha tenha uma célula vizinha na direção do deslocamento de tamanho menor, ou seja, estiver um nível abaixo na árvore (o refinamento 2:1 garante que estará, no máximo, a um nível de diferença), então o deslocamento de metade do tamanho da célula maior a deixa sobre a outra. Como isso não é desejado, pois triângulos seriam gerados sobre outros, o lado da célula maior que a liga à outra célula sofre um deslocamento de somente $\frac{1}{4}$ do tamanho da célula, que é o deslocamento que a célula menor sofrerá, garantindo que não haja sobreposição (células superior esquerda e inferior esquerda na Figura 4.19b).



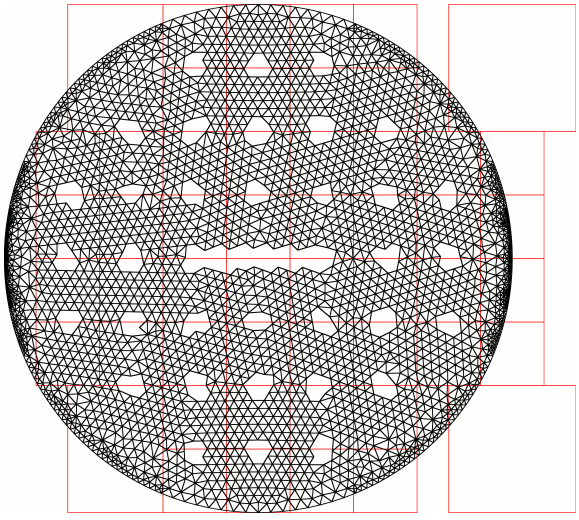
(g) *Quadtree* de particionamento na posição original.



(h) *Quadtree* de particionamento deslocada para baixo.



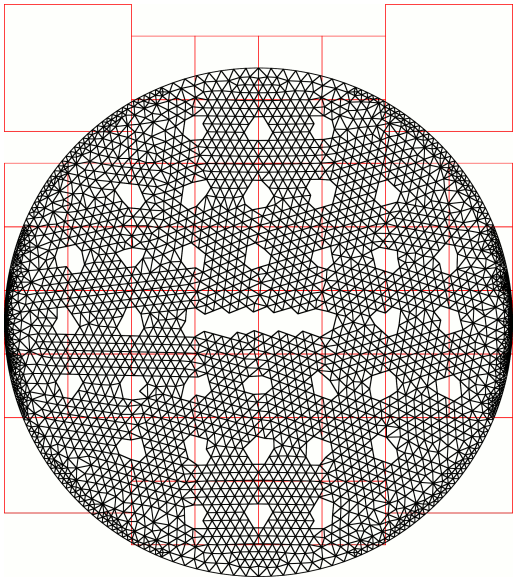
(i) *Quadtree* de particionamento na posição original.



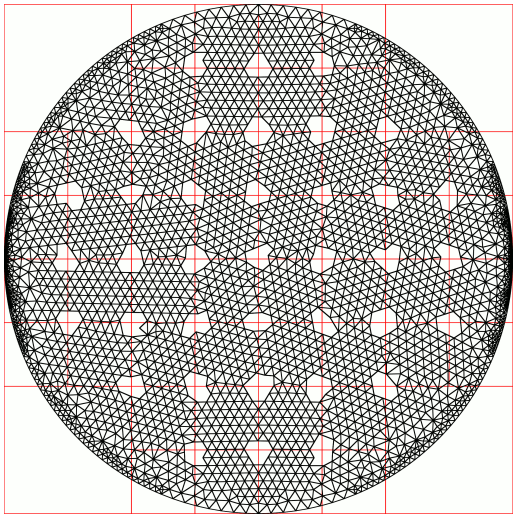
(j) *Quadtree* de particionamento deslocada para a direita novamente.

Figura 4.19: *Quadtree* de particionamento e seus deslocamentos (continuação).

- O deslocamento é feito seguindo a ordem das coordenadas cartesianas. Primeiramente, para a direita (+X) (Figura 4.19b), depois para cima (+Y) (Figura 4.19d), para a esquerda (-X) (Figura 4.19f) e, por fim, para baixo (-Y) (Figura 4.19h), reiniciando o ciclo posteriormente (Figura 4.19j).
- Sempre que uma célula deslocada gerar malha para um lado (+X, por exemplo), não é mais necessário que as células vizinhas por aquele lado se desloquem para o lado oposto (-X, no mesmo exemplo) em um passo futuro, pois a malha entre as



(k) *Quadtree* de particionamento deslocada para cima novamente.



(l) *Quadtree* de particionamento na posição original com cavidades remanescentes.

Figura 4.19: *Quadtree* de particionamento e seus deslocamentos (continuação).

células já foi gerada. Isso economiza tempo e comunicação entre os processos. Essa observação não vale quando uma célula menor se desloca para o lado de uma célula maior, pois o deslocamento contrário da célula maior pode ainda gerar malha.

4. Classifica a nova fronteira de acordo com a nova conformação da *quadtree* de particionamento, observando que algumas arestas podem não pertencer a partição alguma.

5. Calcula a nova carga das partições, baseada na nova classificação da *quadtree* de densidade.

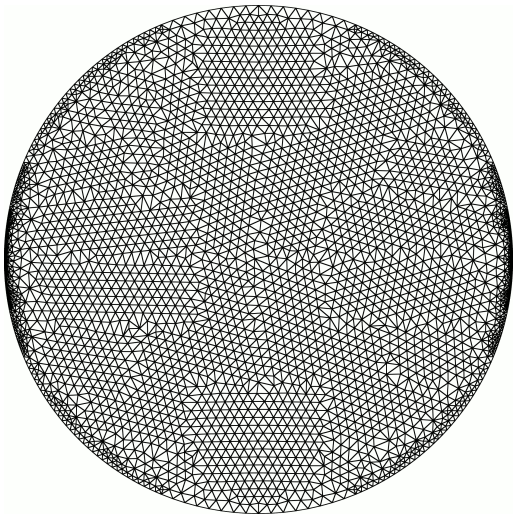
6. Balanceia a carga e envia as partições deslocadas para os processadores escravos (Seção 4.3.5), novamente entrando em espera pelos resultados.

O deslocamento de cada célula da *quadtree* de particionamento é somente de metade do seu tamanho, pois isso mantém a localidade da célula. Se o deslocamento fosse fixo, isto é, se a *quadtree* de particionamento se deslocasse como um todo, uma célula poderia ser movida para uma região muito distante, sem nenhuma relação com a célula.

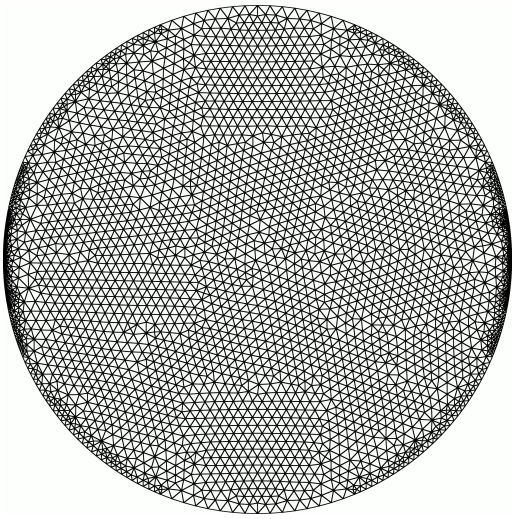
O deslocamento da *quadtree* de particionamento e a geração de malha pelos processos escravos são feitas alternadamente, até que todas as células tenham deslocados para todos os lados e gerado malha (Figura 4.19l).

4.3.8 Finalização da Malha

Quando não for mais possível gerar malha pelo deslocamento, o processador mestre finaliza a malha (Figura 4.19m), gerando os triângulos nas cavidades remanescentes (Figura 4.19l), utilizando a técnica sequencial descrita na Seção 4.2, não se fazendo mais necessárias as restrições observadas na Seção 4.3.6. Tanto a fase baseada em geometria quando a fase baseada em topologia são aplicadas, de forma que a malha seja completamente gerada.



(m) Malha final sem melhoria.



(n) Malha final após suavização e otimização.

Figura 4.19: *Quadtree* de particionamento e seus deslocamentos (continuação).

4.3.9 Melhoria da Malha

O processador mestre segue, então, para a etapa de melhoria da malha. Como esta etapa já foi aplicada nas sub-malhas espalhadas pelos processos, é suficiente suavizar e otimizar a parte da malha que ainda não foi melhorada, ou seja, as diversas fronteiras recebidas pelo mestre e a malha gerada pelo mestre, juntamente com algumas camadas de triângulos adjacentes.

Essas camadas são definidas por elementos adjacentes à fronteira. Em outras palavras, a camada 0 é a própria fronteira e a camada N é formada por vértices, arestas e triângulos adjacentes aos vértices das camadas 0 a $N-1$. Neste trabalho, o número de camadas pode ser modificado, mas foi verificado experimentalmente [Ito et al. 2007] que duas camadas de elementos são suficientes para se ter uma boa malha.

Uma vez que essas camadas são melhoradas no processador mestre, não é necessário que eles sejam previamente melhoradas no processador escravo. Portanto, tal conceito de camadas

também foi utilizado nessa melhoria (Seção 4.3.6). Entretanto, essas camadas não melhoradas correspondem a apenas uma pequena porcentagem da sub-malha, de forma que é mais rápido suavizar e otimizar toda a sub-malha do que encontrar essas camadas. Assim, a quantidade de camadas é ajustada para 0, de forma que somente as fronteiras, nova e antiga (Figura 4.17b), não serão suavizadas nem otimizadas (Figuras 4.17c e 4.17d). A malha final gerada pode ser vista na Figura 4.19n.

4.4 Considerações Finais

Este capítulo apresentou a técnica proposta neste trabalho. Inicialmente, uma *quadtree* de particionamento é gerada, e suas células são as diversas partições do domínio. Esta *quadtree* de particionamento é criada de forma que suas células tenham uma carga inferior a um limite, carga esta que estima a quantidade de elementos gerados em cada partição. Assim, essas partições distribuídas entre os processadores escravos.

Os processadores escravos, então, geram malha nas suas partições, de forma que os triângulos não saiam dos limites da partição. As arestas que não avançarem, portanto, fazem parte da nova fronteira, que é enviada de volta ao processo mestre. O processo mestre, depois de achar a nova fronteira, enviada pelos processos escravos, desloca a *quadtree* de particionamento em uma direção cartesiana. Esse deslocamento, de metade do tamanho de cada célula, é responsável por levar a fronteira de uma partição à sua vizinha naquela direção de deslocamento. Depois de gerar mais malha, as células voltam à posição original. Quando não é mais possível gerar triângulos pelo deslocamento, o processo mestre finaliza a malha e faz a melhoria dos elementos vizinhos às diversas arestas das fronteiras que passaram por ele.

O método proposto neste trabalho é de avanço de fronteira, bidimensional, de particionamento explícito, balanceamento estático, memória compartilhada, distribuída e híbrida, parcialmente acoplado e parcialmente escalável.

4.4.1 Estrutura de Busca Geométrica

Foi implementada também uma estrutura de busca geométrica, para acelerar a procura por vértices na região circular. Essa estrutura é composta de uma árvore, que imita a *quadtree* de densidade, ou seja, cada célula da *quadtree* de densidade tem uma célula correspondente na estrutura de busca. Em cada célula da estrutura de busca, são guardados os vértices que estiverem dentro ou na borda da célula da *quadtree* de densidade. Porém, os testes geométricos são feitos

com a célula da própria *quadtree* de densidade.

Assim, as buscas exaustivas são substituídas por buscas na estrutura. Sempre que os vértices em uma dada região circular (Seção 4.2.3, Figura 4.4c) são requeridos, o quadrado circunscrito à região é encontrado, e as células da estrutura de busca que cruzarem esse quadrado são determinadas. Então, os vértices dessas células que não estiverem no quadrado são descartados. Dessa forma, reduz-se a quantidade de testes geométricos feitos pois, em vez de testar-se com todos os vértices da fronteira, testa-se apenas com os vértices nesse quadrado.

Uma estrutura de busca análoga também foi criada para a busca por arestas que possam interceptar um elemento a ser criado. Assim, diminui-se consideravelmente os testes de interseção. As arestas guardadas em uma célula da estrutura de busca são aquelas cujas caixas limitantes (*bounding boxes*) interceptam a célula da *quadtree* de densidade correspondente.

Essas estruturas também são utilizadas na técnica paralela. Quando um processador tem mais de uma *thread* executando, duas estruturas são utilizadas por cada *thread* (uma para vértices e a outra para arestas), para não haver a possibilidade de uma *thread* fazer os testes com os vértices ou arestas de outra partição. Assim, a localidade de dados é respeitada, não há interferência entre as diferentes partições de um mesmo processador e faz-se desnecessária a utilização de seções críticas [Ben-Ari 2006] para a modificação das estruturas de busca. Observe que, mesmo que cada *thread* tenha suas próprias estruturas de busca, a mesma *quadtree* de densidade é utilizada, de forma compartilhada.

Esta técnica é mais rápida que a técnica sequencial, e gera uma malha tão boa quanto. Estes resultados foram obtidos a partir de exemplos, que serão descritos no próximo capítulo.

Algumas estratégias de balanceamento de carga foram implementadas, de forma a tentar distribuir as partições igualmente entre os processadores. Estas estratégias foram escolhidas por serem simples e de rápida implementação. A quantidade de processos considerada para o balanceamento da carga é a quantidade máxima de *threads* do ambiente.

4.4.2 Outras Estratégias de Balanceamento de Carga

A estratégia de balanceamento de carga descrita na Seção 4.3.5 foi a que obteve melhores resultados práticos. Além dela, outras três estratégias também foram implementadas, e serão aqui descritas.

Estratégia 1

A primeira estratégia, observada em [Ito et al. 2007], consiste em ordenar as partições de forma crescente de cargas e atribuir ao processo $p \in \{1,\ldots,N\}$ as partições p, $2N-p+1$, $2N+p$, $4N-p+1$, e assim por diante (Figura 4.20). Claramente, essa estratégia não leva em conta a carga acumulada em um processo nem a velocidade do processador, e não é adequada a arquiteturas heterogêneas, onde os processadores tenham diferentes capacidades computacionais.

Figura 4.20: Distribuição de 25 partições ordenadas para 5 processos, cada qual representado por uma cor.

Estratégia 2

A segunda estratégia utiliza o paradigma de algoritmos gulosos e é feita iterativamente. Assim, a partição de maior carga é atribuída ao processo mais leve, ou seja, de menor carga acumulada. A carga desse processo é atualizada, somando-se a carga dessa partição à carga acumulada do processo. Então, a próxima partição é selecionada e atribuída a outro (possivelmente o mesmo) processo, e assim por diante, até que todas as partições tenham sido alocadas. Essa estratégia é melhor que a anterior, pois considera a carga acumulada em um processo. Entretanto, como não considera a velocidade, esta estratégia também não é apropriada para arquiteturas heterogêneas.

Estratégia 3

A terceira estratégia é baseada na segunda, mas considera pesos que armazenam informações relativas a velocidade dos processadores. Primeiramente, é preciso estimar o tempo de execução de cada processador para uma determinada tarefa em comum, por exemplo, o tempo de geração e refinamento da *quadtree* utilizada pela técnica sequencial (Seção 4.3.6, Figura 4.3d). Esses tempos são normalizados, dividindo-os pelo menor tempo, de forma que esses valores indicam uma proporção de quanto um certo processador é mais lento que o processador mais rápido. Esses valores serão os pesos de cada processador.

Uma vez tendo esses pesos, quando a carga de um processo for atualizada, a carga da partição em questão é multiplicada pelo peso daquele processador. Esse procedimento tem como efeito estimar que, em um processador mais lento, uma determinada partição terá uma carga maior, isto é, levará mais tempo para ser terminada, do que se fosse executada em um

processador mais rápido.

Apesar de teoricamente melhor que as duas estratégias anteriores, esta estratégia também apresenta falhas. Uma delas é que essa estratégia calcula a velocidade do processador baseando-se no tempo de execução de uma tarefa em comum, feita uma única vez. Isto pode não ser muito preciso, por exemplo, se o processador estiver executando outros processos pesados e/ou tiver que fazer muitos acessos à memória virtual durante a estimativa do tempo de execução. Dessa forma, dois processadores de velocidades iguais podem acabar tendo pesos diferentes.

Considerações sobre o Balanceamento de Carga

Nessas três estratégias, as partições de um processador são determinadas antes da geração da malha, de forma que é possível utilizar-se de comunicação coletiva entre os processadores. É sabido [Grama et al. 2003] que é possível ter algumas otimizações nos algoritmos de comunicação coletiva, e essas estratégias podem fazer uso delas. Portanto, a degradação a que se refere a Seção 4.3.5, quando se aumenta a quantidade de processadores, é menos sentida nessas estratégias. Para o envio dos resultados, foi utilizada comunicação coletiva em todas as estratégias.

5 *Exemplos e Análise de Resultados*

5.1 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 bom *speed-up* atingido pelo algoritmo.

O programa foi desenvolvido em C++, utilizando a biblioteca de paralelismo em memória compartilhada OpenMP (*Open Multi-Processing*) e a biblioteca de passagem de mensagens MPI (*Message Passing Interface*) para paralelismo de memória distribuída. Na versão com interface, foram ainda utilizadas a biblioteca de renderização OpenGL (com duas de suas bibliotecas, glu e glew), e a biblioteca de interfaces wxWidgets.

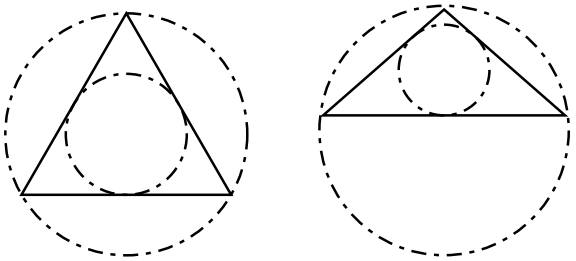
Para testar a implementação, foram utilizados computadores de memória compartilhada, distribuída e híbrida, rodando unicamente a implementação, além do sistema operacional:

I7 Computador com processador Intel® Core i7™ (com 4 núcleos HT) de 2,67 GHz 64 bits, com 8 GB de memória RAM DDR3 de 1067 MHz, rodando a distribuição Ubuntu 9.10 do sistema operacional GNU/Linux. O compilador utilizado foi o GNU/gcc 4.4.1. Este computador de última geração serviu para testes de memória compartilhada somente.

Cluster Computador *cluster* Itautec® com 12 nós, 10 deles com 2 processadores Intel® Xeon™ (com 2 núcleos) de 1,8 GHz 32 bits, e os outros 2 nós com 1 processador do mesmo tipo, totalizando 44 núcleos no *cluster*. Cada nó tem 1 GB de memória RAM DDR, compartilhada entre os seus processadores. Foi utilizada a distribuição Rocks Cluster do sistema operacional GNU/Linux, o compilador GNU/gcc 4.3.3 e a versão do MPI foi o OpenMPI 1.2.3. Este *cluster* foi utilizado para os testes de memória distribuída e híbrida, com todos os processadores tendo a mesma capacidade computacional. Assim, os dois nós com apenas um processador não foram utilizados, sendo utilizados somente os 10 nós com 2 processadores, sendo um mestre e 9 escravos.

Os exemplos considerados neste capítulo foram todos retirados da literatura existente em geração de malha e métodos de elementos finitos, e foram escolhidos por possuírem formas diversificadas. As geometrias estão definidas no Apêndice A. Em todos eles, a carga foi calculada como a carga total dividida pelo número de processadores trabalhadores, que pode ser o número de núcleos, para o caso de memória compartilhada; o número de nós, no caso de memória distribuída; ou o somatório do número de núcleos dos nós, para o caso de memória híbrida.

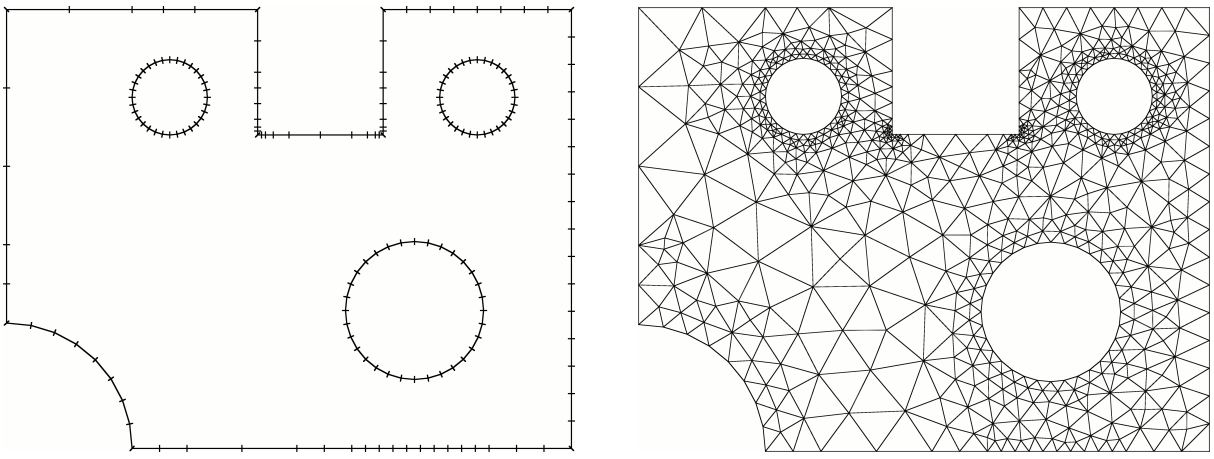
A métrica utilizada para o cálculo da qualidade dos triângulos é definida como $\alpha = \frac{2R_i}{R_c}$, onde R_i e R_c são os raios dos círculos inscrito e circunscrito, respectivamente. Esta métrica α tem valor $1,0$ para um triângulo equilátero (Figura 5.1a). Quanto pior a qualidade do elemento, mais próximo de $0,0$ é o valor de α. 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 os de boa qualidade (Figura 5.1b).



(a) Triângulo equilátero. (b) Triângulo isósceles.

Figura 5.1: Triângulos e seus círculos inscrito e circunscrito.

5.2 Placa 1



(a) Placa 1 [Cavalcante-Neto 1994]. (b) Malha de 1305 elementos.

Figura 5.2: Exemplo da Placa 1.

A Figura 5.2 mostra o exemplo de uma placa, retirada de [Cavalcante-Neto 1994]. Do lado esquerdo, é mostrada a fronteira (Figura 5.2a) e do lado direito, a malha (Figura 5.2b). A fronteira desse exemplo foi refinada 4 vezes, ou seja, cada aresta mostrada na Figura 5.2a foi dividida ao meio 4 vezes, recursivamente, gerando 16 arestas. Esse refinamento foi aplicado para que se obtivesse uma malha relativamente grande, com cerca de 80 mil elementos, a fim de ser vantajoso o emprego do paralelismo na sua geração. Os resultados e análises que se seguem dizem respeito à essa malha maior.

O gráfico da Figura 5.3 mostra o *speed-up* atingido pela implementação da técnica. O *speed-up* da implementação em memória compartilhada foi melhor do que o obtido para os casos de memória distribuída e híbrida, sendo quase linear entre 4 e 7 *threads*, o que é um resultado muito bom.

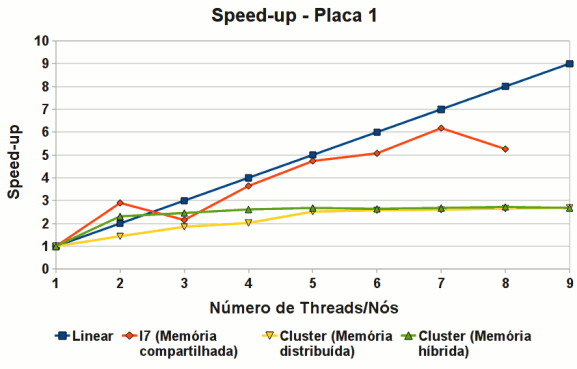
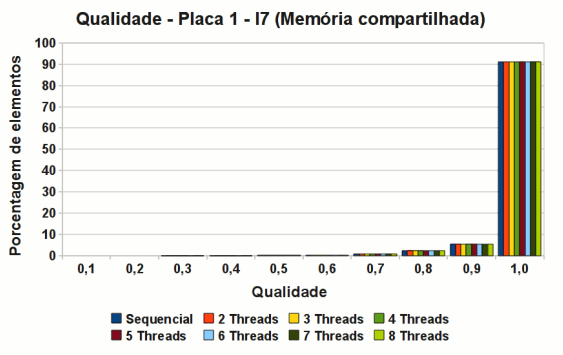


Figura 5.3: *Speed-up* para o exemplo da Placa 1.

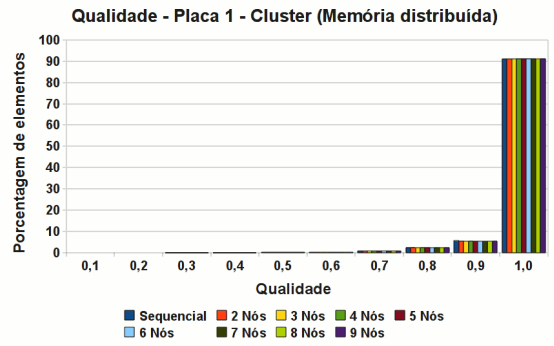
Os gráficos da Figura 5.4 mostram a qualidade das malhas geradas pelas diversas execuções do programa. Por conta do não-determinismo inerente aos algoritmos paralelos, diferentes execuções do mesmo programa podem gerar malhas diferentes, ainda que a entrada, os parâmetros e os processadores utilizados sejam os mesmos. Portanto, faz-se necessária a análise da qualidade das malhas geradas em cada execução do programa. Observa-se que, neste exemplo, a grande maioria dos triângulos têm boa qualidade.

A Figura 5.5 mostra uma malha gerada pela técnica paralela com 4 *threads*, cada uma indicada por uma cor diferente. A parte da malha em preto foi gerada pelo mestre, na etapa de finalização (Seção 4.3.8) ou de melhoria da malha (Seção 4.3.9).

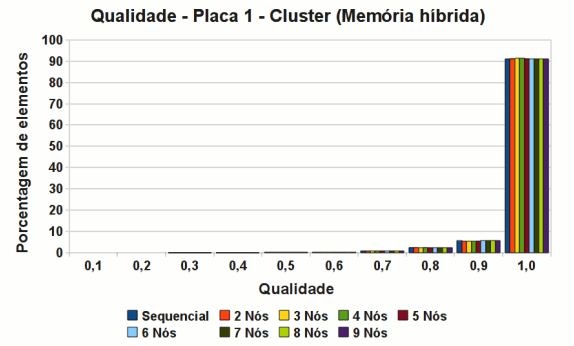
As regiões finas coloridas presentes na Figura 5.5 que ocorrem entre duas partições são provenientes do deslocamento. Como as duas partições vizinhas já geraram elementos, sobra uma região fina entre elas, cujos elementos gerados são decorrentes do deslocamento de uma das células no sentido da outra. Essa partição deslocada pode ser atribuída para o mesmo processador que gerou a malha de uma das duas partições, gerando uma malha da mesma cor



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.4: Qualidade das malhas do exemplo da Placa 1.

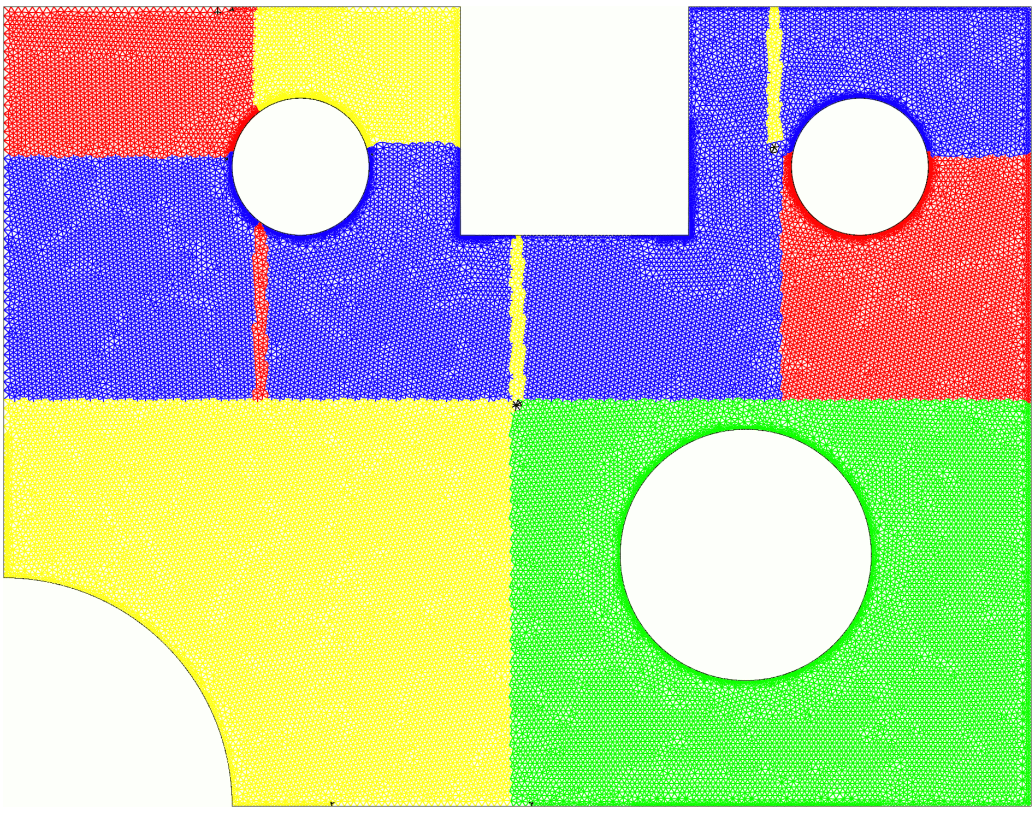
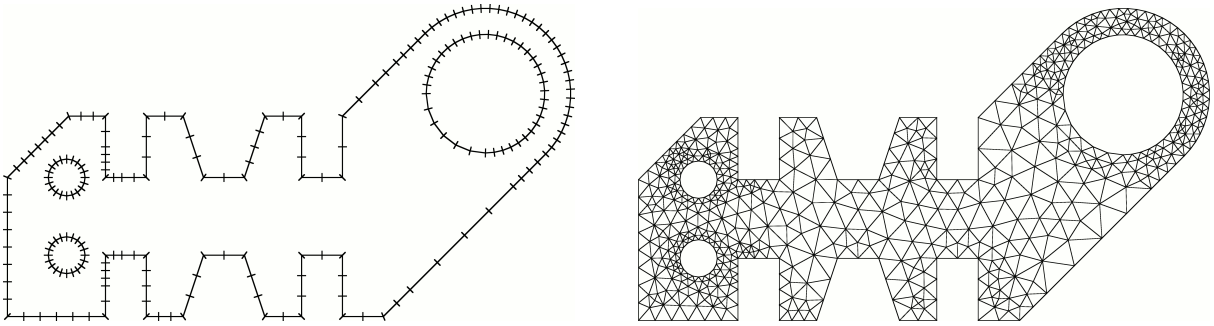


Figura 5.5: Malha do exemplo da Placa 1 gerada em paralelo com 4 *threads*.

(por exemplo, a região imperceptível entre a malha amarela e a malha verde, que foi atribuída ao processador verde), ou para um outro processador, gerando uma malha de outra cor (por exemplo, a região fina vermelha entre duas malhas azuis). No caso de memória compartilhada, a atribuição da partição para um mesmo processador pode ser benéfica para o *speed-up*, pois, se alguma parte da fronteira já estiver presente na memória *cache*, reduz-se a quantidade de acessos à memória RAM, que é mais lenta.

5.3 Placa 2

A Figura 5.6 mostra outra placa, retirada de [Persson 2004]. Do lado esquerdo, é mostrada a fronteira (Figura 5.6a) e do lado direito, a malha (Figura 5.6b). A fronteira desse exemplo foi refinada 4 vezes gerando uma malha de 121 mil elementos.



(a) Placa 2 [Persson 2004]. (b) Malha de 1121 elementos.

Figura 5.6: Exemplo da Placa 2.

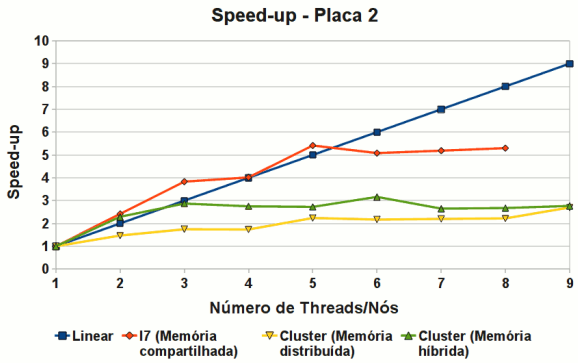
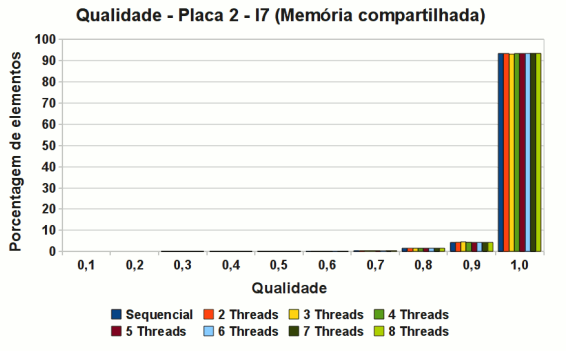


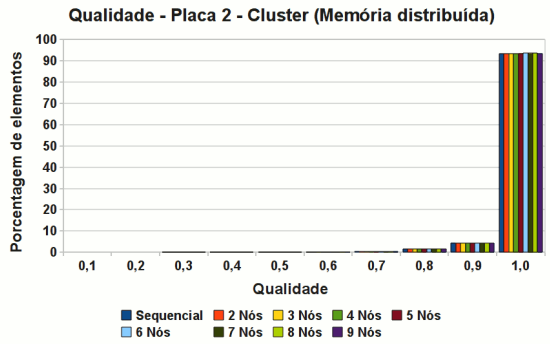
Figura 5.7: *Speed-up* para o exemplo da Placa 2.

O gráfico da Figura 5.7 mostra o *speed-up* da implementação, e os gráficos da Figura 5.8 mostram a qualidade das diversas malhas obtidas. Assim como a Placa 1, o *speed-up* apresentou-se melhor em memória compartilhada do que em memória distribuída/híbrida, por

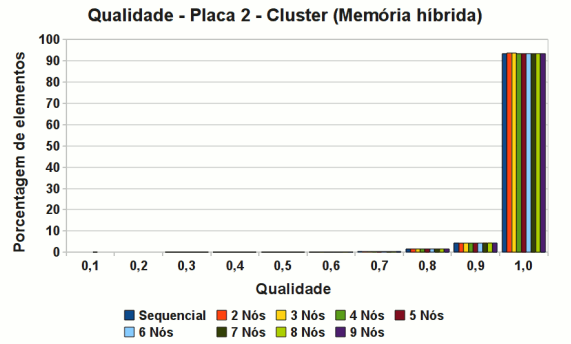
conta da ausência de comunicação entre processos. Pode-se ver, entretanto, que houve um ganho bom em utilizar-se memória híbrida, quando comparado com memória distribuída.



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.8: Qualidade das malhas do exemplo da Placa 2.

A Figura 5.9 mostra uma malha gerada para o exemplo utilizando-se 4 *threads*.

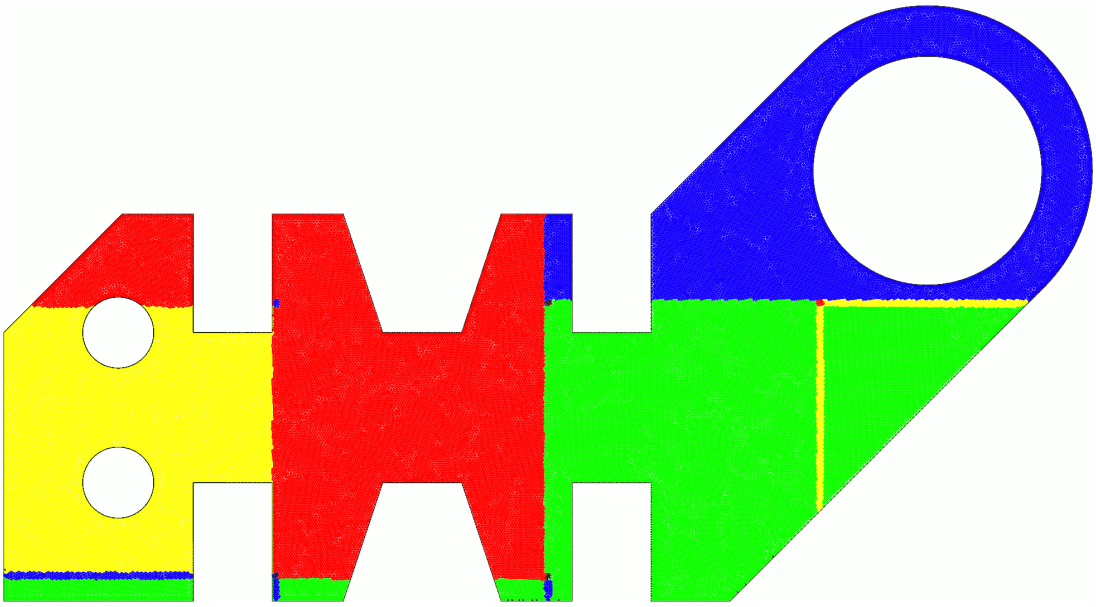
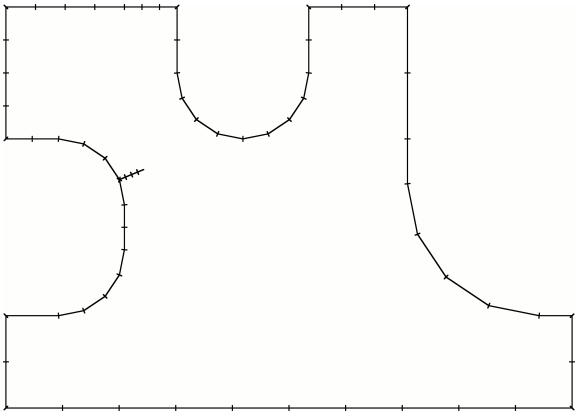


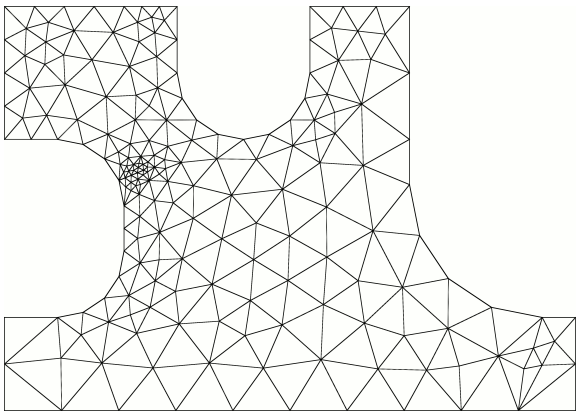
Figura 5.9: Malha do exemplo da Placa 2 gerada em paralelo com 4 *threads*.

5.4 Fecho de Porta de Avião

A Figura 5.10 mostra uma versão bidimensional de um fecho de porta de avião com fratura, retirado de [Cavalcante-Neto 1998]. Do lado esquerdo, é mostrada a fronteira (Figura 5.10a) e do lado direito, a malha (Figura 5.10b). A fronteira desse exemplo foi refinada 5 vezes gerando uma malha de 202 mil elementos.



(a) Versão bidimensional de um fecho de porta de avião com fratura [Cavalcante-Neto 1998].



(b) Malha de 333 elementos.

Figura 5.10: Exemplo do Fecho de Porta de Avião.

O gráfico da Figura 5.11 mostra o *speed-up* atingido neste exemplo, e os gráficos da Figura 5.12 mostram a qualidade das malhas geradas nas diversas execuções do programa. Nesse exemplo, as três arquiteturas testadas apresentaram gráficos de *speed-up* parecidos, com a memória híbrida se destacando entre elas, como esperado.

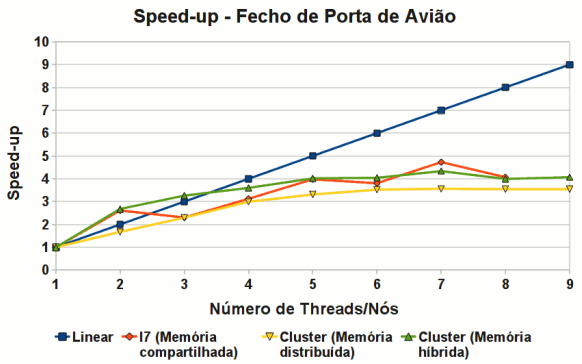
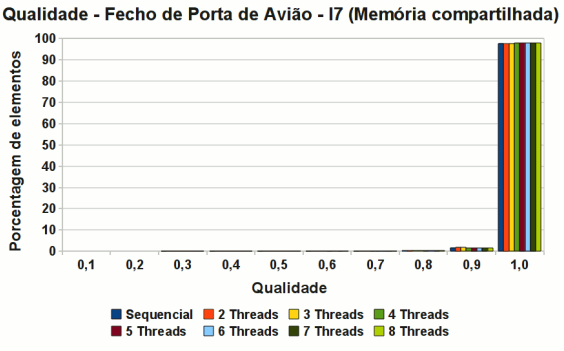
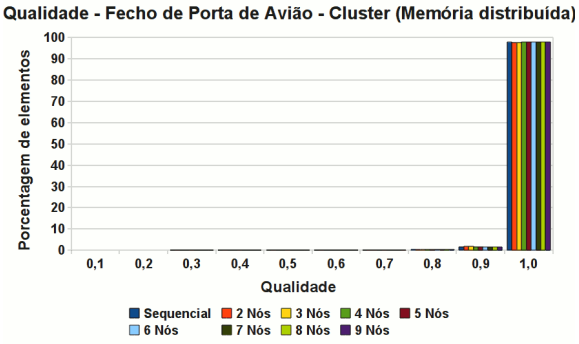


Figura 5.11: *Speed-up* para o exemplo do Fecho de Porta de Avião.

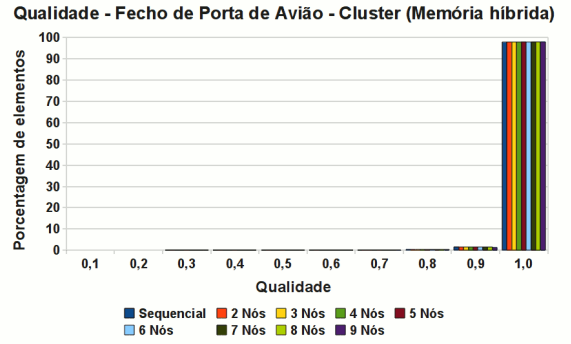
A Figura 5.13 mostra uma malha gerada para o exemplo do fecho de porta de avião utilizando-se 4 *threads*.



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.12: Qualidade das malhas do exemplo do Fecho de Porta de Avião.

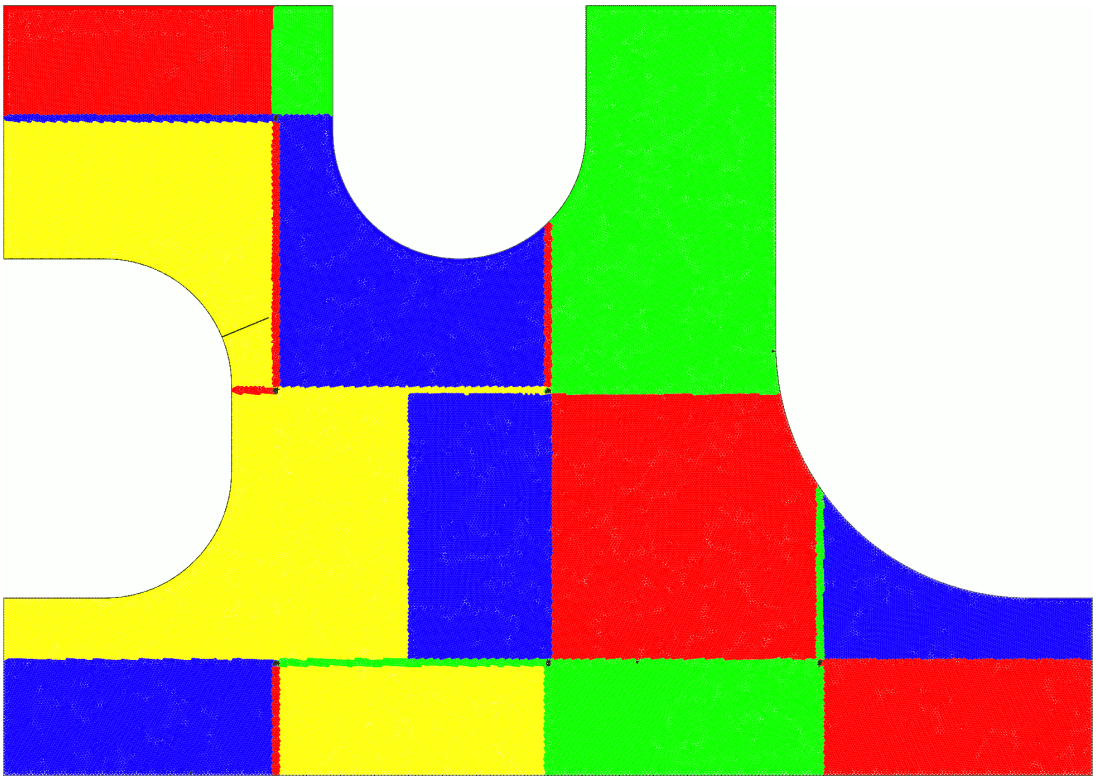
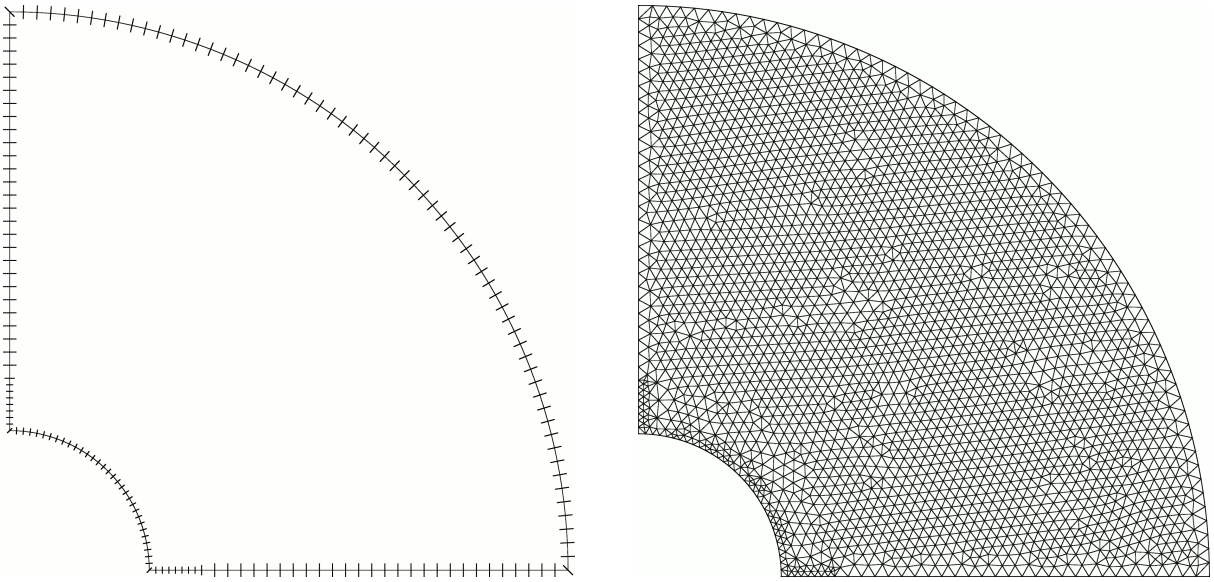


Figura 5.13: Malha do exemplo do Fecho de Porta de Avião gerada em paralelo com 4 *threads*.

5.5 Cilindro

A Figura 5.14 mostra o exemplo de um cilindro, retirado de [Cavalcante-Neto 1994]. Do lado esquerdo, é mostrada a fronteira (Figura 5.14a) e do lado direito, a malha (Figura 5.14b). A fronteira desse exemplo foi refinada 3 vezes, gerando uma malha de cerca de 322 mil elementos.



(a) Cilindro [Cavalcante-Neto 1994]. (b) Malha de 4976 elementos.

Figura 5.14: Exemplo do Cilindro.

O gráfico da Figura 5.15 mostra o *speed-up* atingido neste exemplo, e os gráficos da Figura 5.16 mostram a qualidade das malhas geradas nas diversas execuções do programa. Pode-se ver que a arquitetura de memória híbrida novamente se sobressaiu, apresentando um *speed-up* acima do linear em até 6 processadores. Isso ocorre por causa da redução do tamanho da fronteira e, consequentemente, da quantidade de testes de interseção.

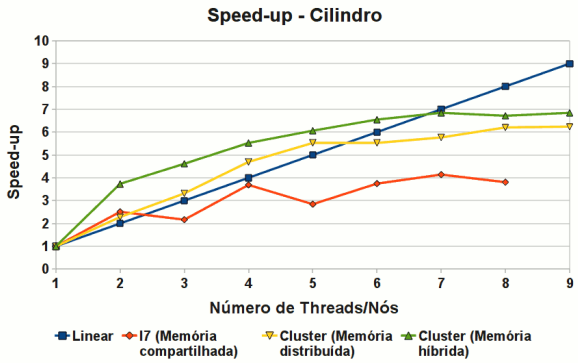
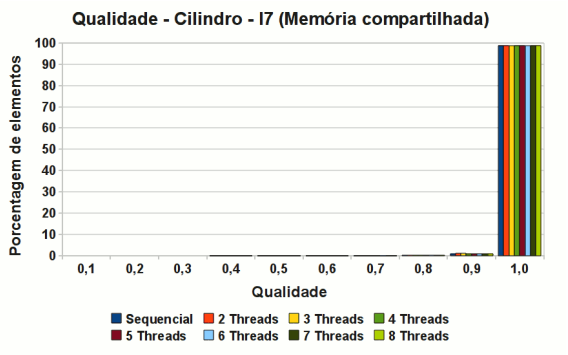
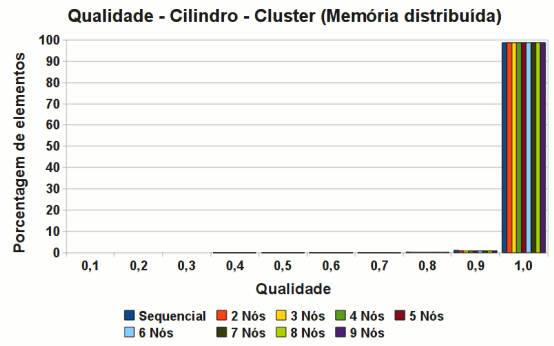


Figura 5.15: *Speed-up* para o exemplo do Cilindro.

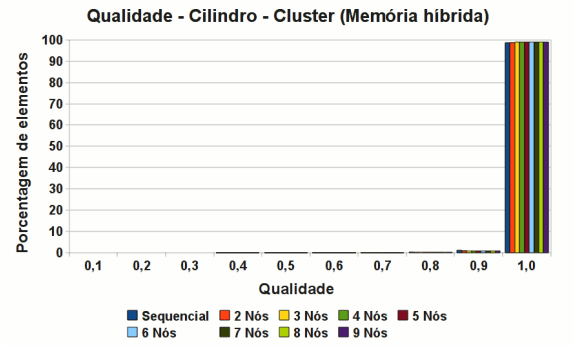
Uma malha gerada com 4 *threads* para o exemplo do cilindro pode ser vista na Figura 5.17.



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.16: Qualidade das malhas do exemplo do Cilindro.

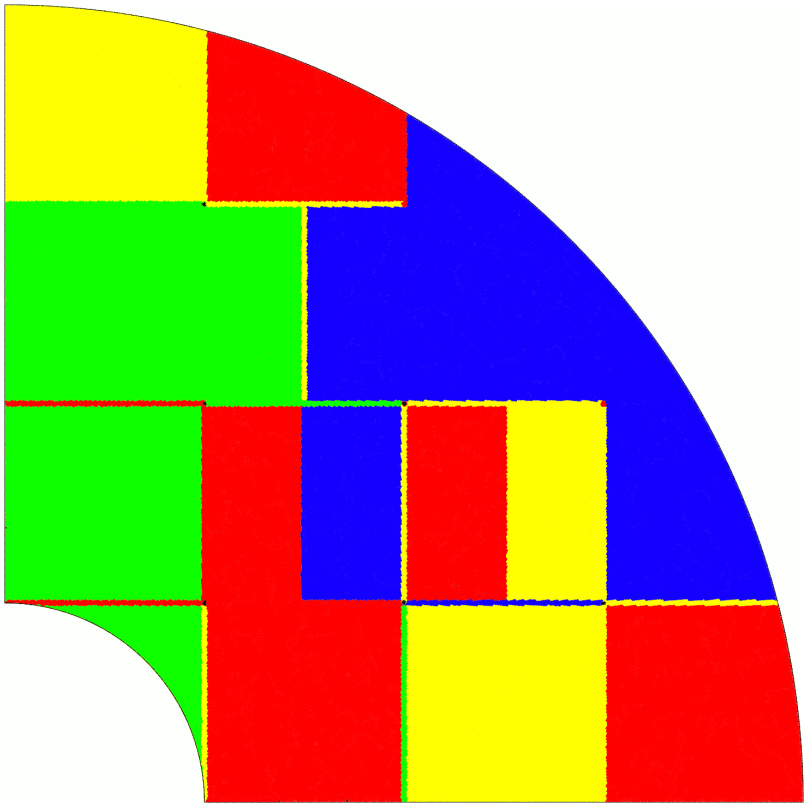
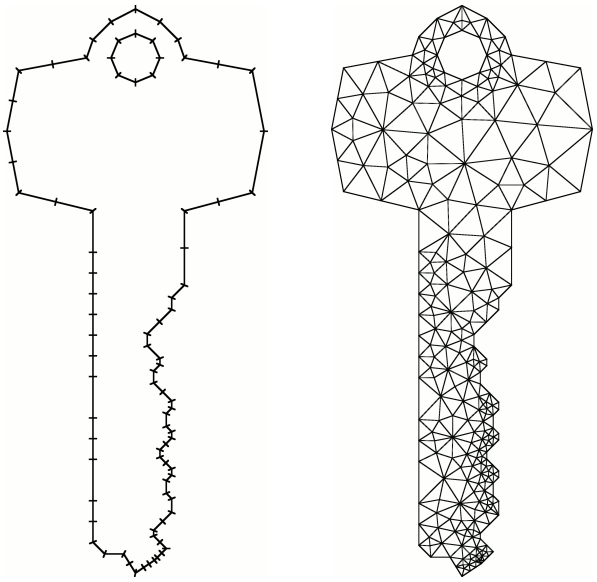


Figura 5.17: Malha do exemplo do Cilindro gerada em paralelo com 4 *threads*.

5.6 Chave

A Figura 5.18 mostra uma chave retirada de [Shewchuk 1997]. Do lado esquerdo, é mostrada a fronteira (Figura 5.18a) e do lado direito, a malha (Figura 5.18b). A fronteira desse exemplo foi refinada 6 vezes gerando uma malha de cerca de 367 mil elementos, a maior gerada dentre esses exemplos.



(a) Chave [Shewchuk 1997].

(b) Malha de 420 elementos.

Figura 5.18: Exemplo da Chave.

O gráfico da Figura 5.19 mostra o *speed-up* da implementação, e os gráficos da Figura 5.20 mostram a qualidade das diversas malhas obtidas. Por ser uma malha maior que as outras e por ter mais partições, no caso de memória híbrida, a técnica tem um melhor desempenho quando comparado à memória distribuída.

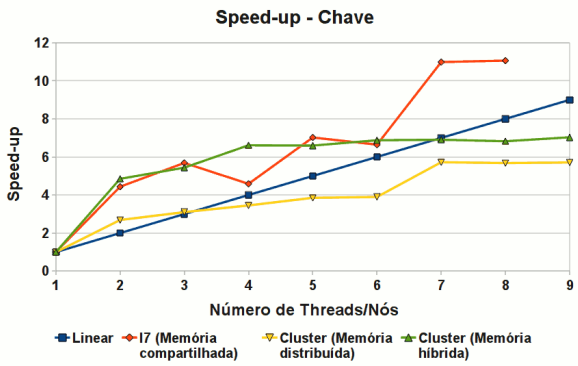
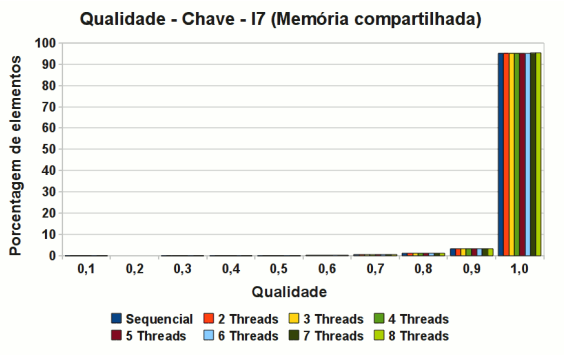
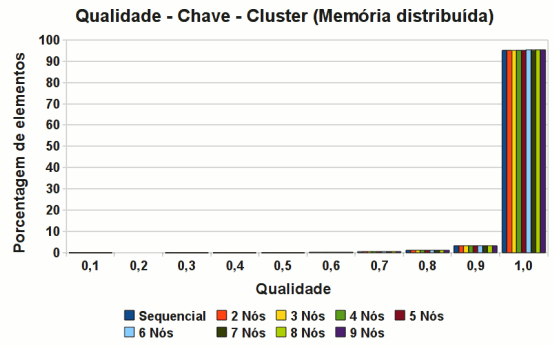


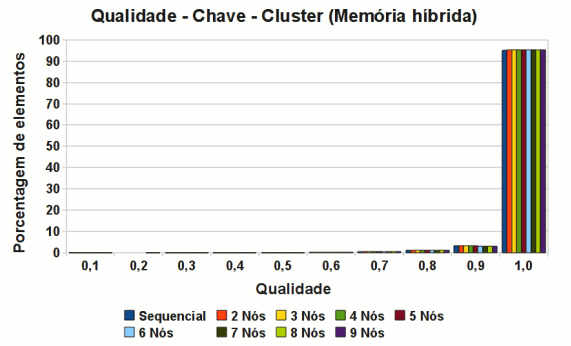
Figura 5.19: *Speed-up* para o exemplo da Chave.



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.20: Qualidade das malhas do exemplo da Chave.

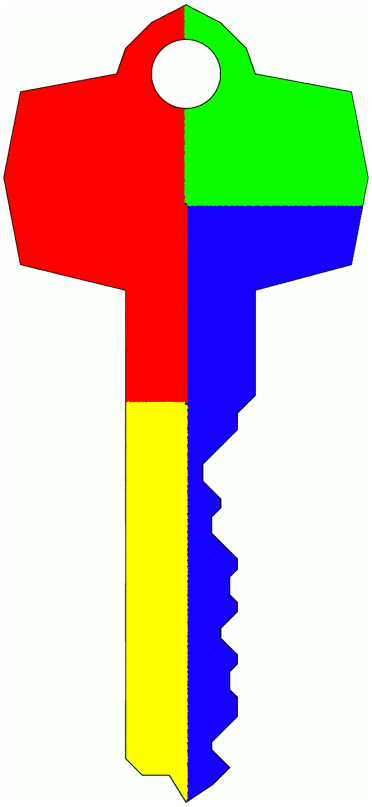


Figura 5.21: Malha do exemplo da Chave gerada em paralelo com 4 *threads*.

A Figura 5.21 mostra uma malha gerada pela técnica paralela com 4 *threads*. Como pode ser visto, a técnica apresentou um comportamento uniforme. Entretanto, como as partições são atribuídas aos processadores de maneira não-determinística, essa uniformidade aconteceu ao acaso, e uma execução subsequente do programa pode apresentar uma atribuição diferente.

5.7 Considerações Finais

Os tamanhos das malhas geradas para os exemplos estão resumidas no gráfico da Figura 5.22 e os gráficos de *speed-up* para cada arquitetura estão mostradas na Figura 5.23. Apesar de grandes, as malhas couberam na memória RAM dos computadores testados, sendo possível gerá-las sequencialmente e em memória compartilhada. Nos casos de memória distribuída e híbrida, seria possível gerar malhas ainda maiores, uma vez que seus elementos estariam divididos entre os diversos processadores envolvidos.

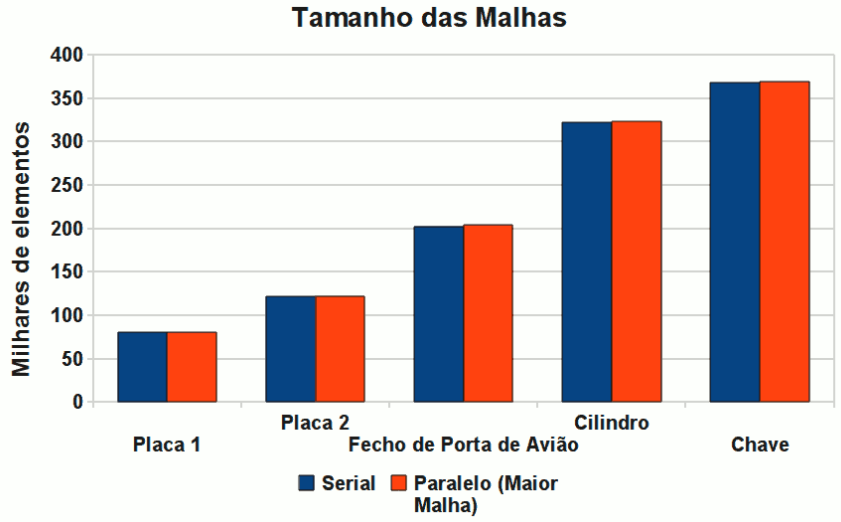
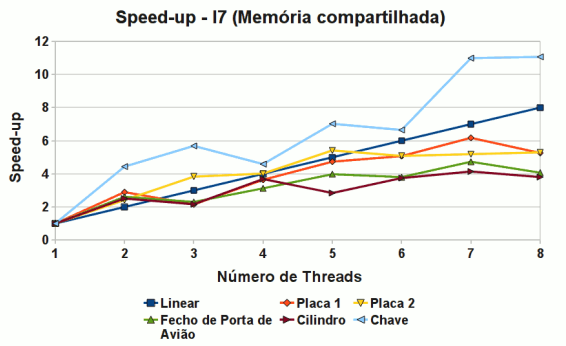
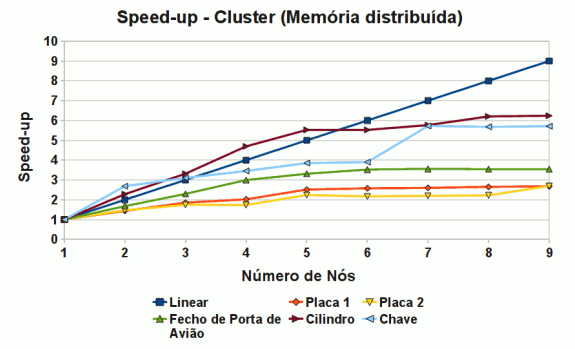


Figura 5.22: Tamanhos das malhas geradas sequencialmente e em paralelo.

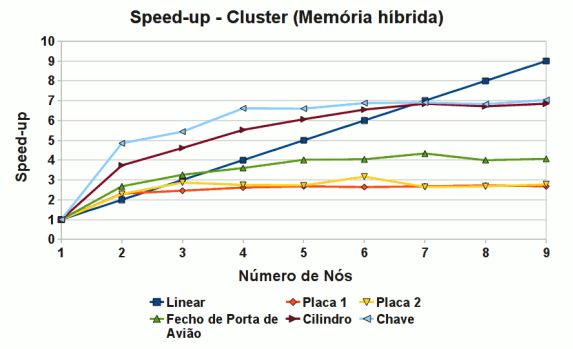
Como pôde-se observar neste capítulo, a técnica proposta neste trabalho sempre gera malhas de boa qualidade. Além disso, a técnica paralela é capaz de atingir um bom *speed-up* tanto em memória compartilhada, quanto distribuída ou híbrida. A estratégia apresentada satisfaz os quatro requisitos da técnica sequencial apresentados na Seção 4.1, por conta tanto do particionamento implícito quanto da própria técnica sequencial, levemente modificada, aplicada nas partições. Por fim, pode-se dizer que a perda de qualidade introduzida pela paralelização da técnica sequencial é insignificante quando comparado ao ganho de velocidade obtido.



(a) Memória compartilhada.



(b) Memória distribuída.



(c) Memória híbrida.

Figura 5.23: *Speed-up* de cada exemplo em cada arquitetura.

6 *Conclusão e Trabalhos Futuros*

6.1 Principais Contribuições

Este trabalho apresentou uma técnica paralela de geração de malhas triangulares para aplicações de métodos de elementos finitos e computação gráfica, utilizando o modelo de paralelismo mestre/escravos. Nesse modelo, um processador, o mestre, fica responsável pelo gerenciamento do algoritmo e os outros processadores, os escravos, são responsáveis pelo trabalho pesado, no caso, a geração de malha.

Essa técnica paralela pode ser utilizada em arquiteturas de memória tanto compartilhada quanto distribuída ou híbrida. Por esse motivo, a quantidade total de processadores escravos é calculada como a quantidade de *threads* disponíveis no sistema como um todo, ou seja, a soma da quantidade de *threads* de cada processador físico.

A técnica procede da maneira descrita a seguir. Inicialmente, uma *quadtree* representativa da fronteira dada é construída e refinada de duas formas diferentes, uma para que as células do interior não sejam muito grandes, quando comparadas com as células da fronteira, e outra para que células adjacentes não tenham tamanhos muito discrepantes. Essa *quadtree* serve como função densidade para o tamanho dos triângulos gerados e, portanto, é uma boa estimativa para a quantidade de elementos da malha final. Assim, a carga total é contabilizada como a quantidade de células-folha que não estiverem fora daquela fronteira.

Uma nova *quadtree* é construída, dessa vez para particionar o domínio, formado pela fronteira e pelo seu interior. As células dessa *quadtree* de particionamento são subdivididas até que a carga de cada uma seja menor que uma carga limite, calculada como a razão entre carga total e a quantidade de escravos disponíveis no sistema paralelo. Nesta *quadtree* de particionamento, também é aplicado o refinamento para que duas células adjacentes não tenham tamanhos muito discrepantes.

Depois disso, essas partições retangulares são distribuídas entre os processadores escravos, para que estes apliquem uma técnica sequencial de geração de malha por avanço de fronteira,

técnica esta que foi levemente modificada para que não sejam gerados triângulos fora da partição atribuída. Os processadores escravos ainda aplicam uma melhoria nas malhas geradas em cada partição, além de classificarem novamente as células da *quadtree* de densidade de acordo com a conformação da nova fronteira.

Esta nova fronteira é enviada ao processador mestre, juntamente com a nova classificação da *quadtree* de densidade. As células da *quadtree* de particionamento são, então, deslocadas de metade do seu tamanho em uma certa direção cartesiana, tomando cuidado para que não haja sobreposição de células. Posteriormente, a carga de cada célula é, então, contabilizada de acordo com a nova classificação, recebida dos processadores escravos. Estas células são novamente enviadas para os processadores escravos gerarem mais malha.

Este procedimento de deslocamento/geração é repetido em direções e sentidos alternados, até que não seja mais possível gerar nenhum triângulo. O processador mestre, então, termina de gerar a malha nas cavidades remanescentes e aplica a melhoria na parte da malha próxima às diversas fronteiras que passaram por ele ao longo do procedimento.

Ao final, a malha gerada atende aos requisitos que motivaram o desenvolvimento da técnica sequencial original: respeitar o contorno, gerar uma malha de boa qualidade e com uma boa transição entre regiões refinadas e regiões desrefinadas, e tratar contornos com fraturas. Além disso, a técnica paralela apresentou um bom *speed-up*, às vezes super-linear, principalmente para malhas com mais de 200 mil elementos.

Esta super-linearidade pode ser explicada pelo fato de os tamanhos da fronteira e da malha diminuírem consideravelmente, diminuindo também a quantidade de testes de interseção. A versão de memória compartilhada apresentou-se melhor que a de memória distribuída, principalmente por conta da não existência de comunicação inter-processos. A versão de memória híbrida por vezes se apresentou a melhor de todas, mostrando que, de fato, uma maior quantidade de partições pode ser melhor distribuída entre uma quantidade maior de processos escravos.

6.2 Trabalhos Futuros

O bom desempenho desta técnica depende da carga máxima, uma vez que ela determina a quantidade de partições: quanto mais partições, mais paralelizável é esta técnica. Em compensação, a quantidade de malha a ser melhorada no final é maior, em uma etapa sequencial. Por outro lado, quanto menos partições, menos processos são utilizados. Portanto, o número de partições deve ser bem ajustado para uma certa quantidade de processos escravos ser bem utilizada e para minimizar o tempo sequencial final. Assim, deve-se investigar uma forma mais inteligente de

calcular a carga máxima de cada partição do que apenas dividir a carga total pela quantidade de processos escravos.

O deslocamento da *quadtree* de particionamento nas direções cartesianas acaba deixando cavidades em áreas próximas do encontro de quatro células. Para resolver este problema, pode-se aplicar o deslocamento das células na diagonal. Além disso, o deslocamento da *quadtree* de particionamento é fixo, para a direita, para cima, para a esquerda e para baixo. Isso pode não ser muito vantajoso para determinados contornos, onde um deslocamento inicial para cima, por exemplo, poderia levar a um melhor *speed-up*. Portanto, deve-se investigar uma maneira de adaptar a ordem do deslocamento de acordo com a entrada dada.

Como o deslocamento de uma célula é de metade do seu tamanho, o trabalho dessa partição para o próximo passo de geração também cai pela metade. Assim, é indicado estudar outros valores de deslocamento e o seu impacto no *speed-up* final da técnica. Além disso, conforme os deslocamentos acontecem e a fronteira avança, diminui o espaço interno, diminuindo também a quantidade de partições ativas, que pode chegar a ser menor que a quantidade de processadores escravos. Subdividir as células internas resolve este problema, aumentando dinamicamente a quantidade de partições.

Nos casos de memórias distribuída e híbrida, o envio de partições com pouca carga, ou seja, que terão pouco trabalho a ser executado pode ser desvantajoso. Assim, deve-se investigar uma maneira de deixar as partições com pouca carga para o próprio processador mestre gerar malha.

Outro ponto que merece observação é o da melhoria da malha aplicada sequencialmente ao final. Se, durante a etapa paralela, os processadores escravos enviassem, junto com a fronteira, a parte da malha que vai ser melhorada sequencialmente, o processador mestre poderia passar essa parte da malha para outro processador escravo, depois do deslocamento. Assim, este processador escravo já faria a melhoria dessa parte da malha, diminuindo o tempo sequencial final da técnica.

Foi verificado um certo desbalanceamento de carga nesta técnica. Esse desbalanceamento acontece na etapa de melhoria da malha, quando esta é muito grande, e ocorre porque a busca e a remoção de elementos são feitas em tempo linear no tamanho da malha. Este problema pode ser resolvido modificando a estrutura de dados da malha, para que a busca e a remoção sejam feitas em tempo, no máximo, logarítmico no tamanho da malha.

É possível, ainda, remover a sincronização global presente no deslocamento da *quadtree* de particionamento. Para deslocá-la em uma direção, é necessário somente que uma partição conheça a(s) nova(s) fronteira(s) da(s) partição(ões) vizinha(s) naquela direção. Portanto, uma

sincronização local, somente entre as partições envolvidas, é necessária para o deslocamento, abrindo mão da sincronização global.

Por fim, uma extensão natural é implementar uma versão tridimensional da técnica. Esta versão não deve ser muito complexa de se implementar, uma vez que a ideia aplica-se tanto para duas quanto para três dimensões.

APÊNDICE A – Geometrias dos Exemplos

A.1 Introdução

Este apêndice mostra a geometria dos exemplos utilizados no presente trabalho.

A.2 Placa 1

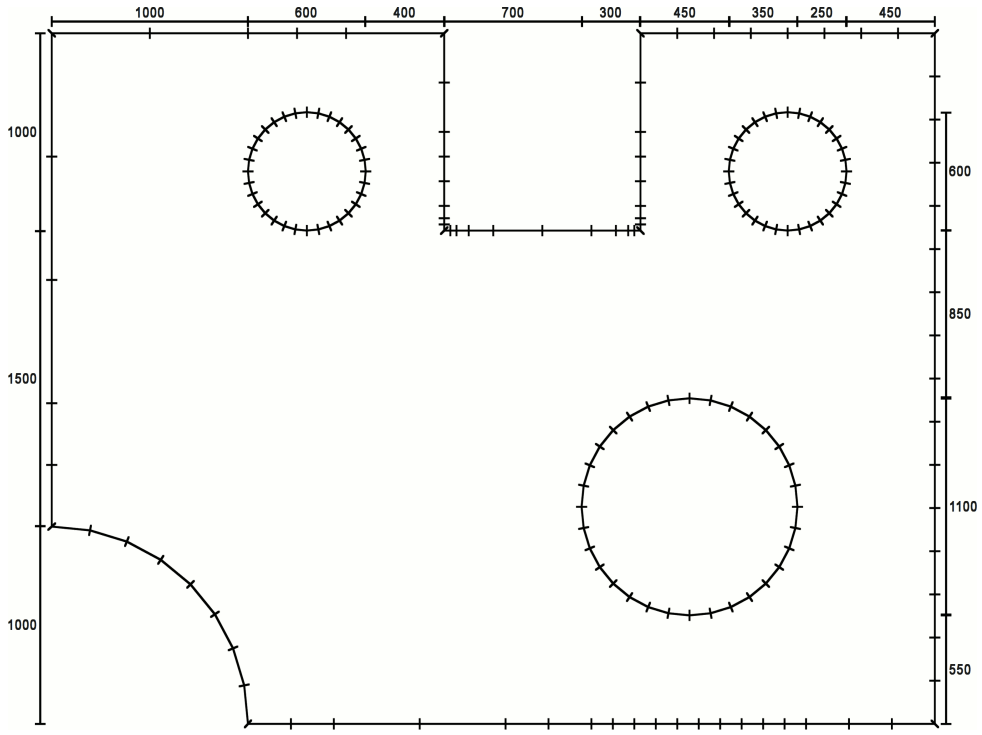


Figura A.1: Geometria do exemplo da Placa 1.

A.3 Placa 2

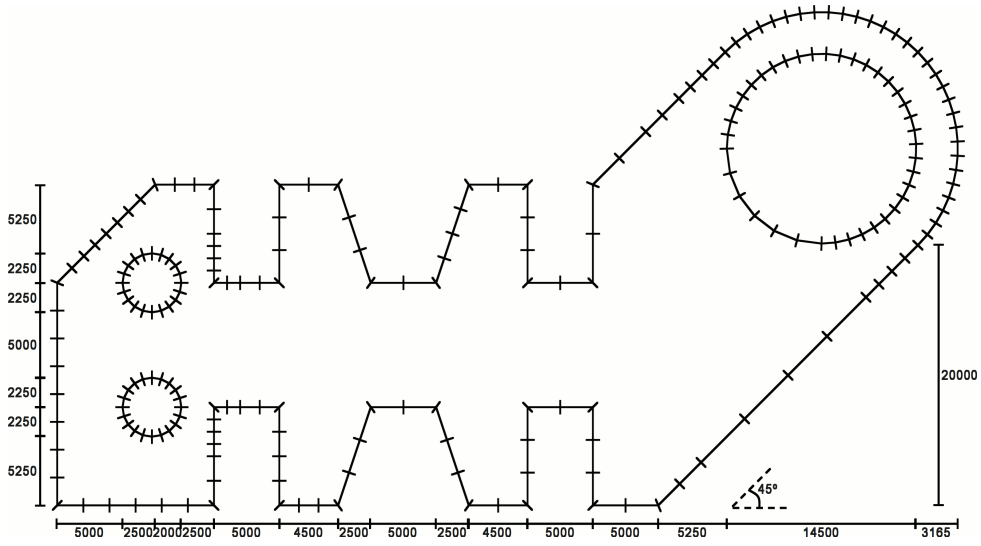


Figura A.2: Geometria do exemplo da Placa 2.

A.4 Fecho de Porta de Avião

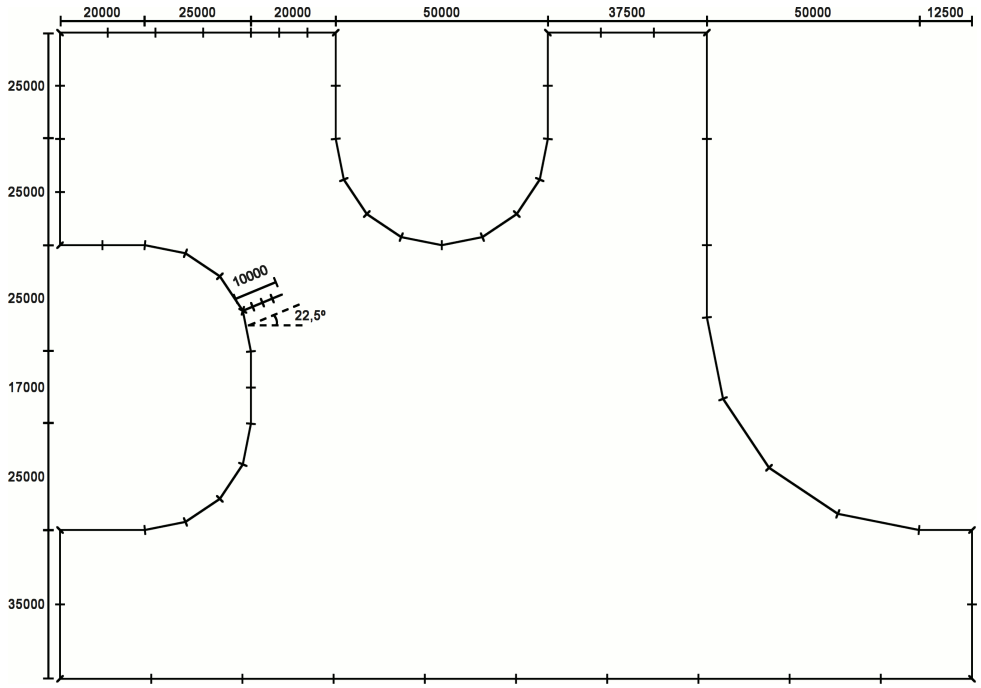


Figura A.3: Geometria do exemplo do Fecho de Porta de Avião.

A.5 Cilindro

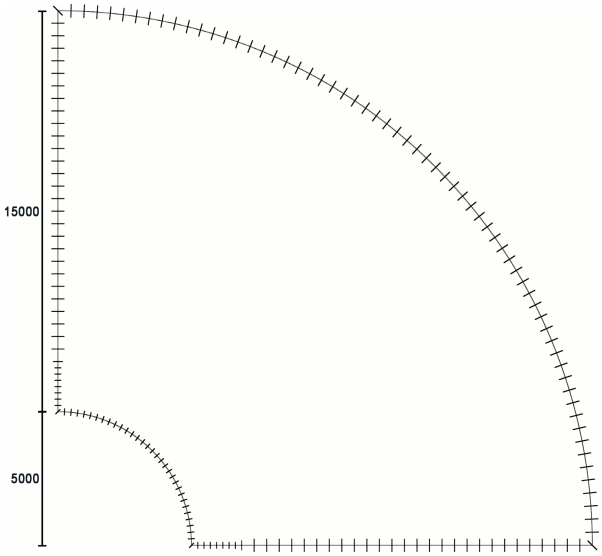


Figura A.4: Geometria do exemplo do Cilindro.

A.6 Chave

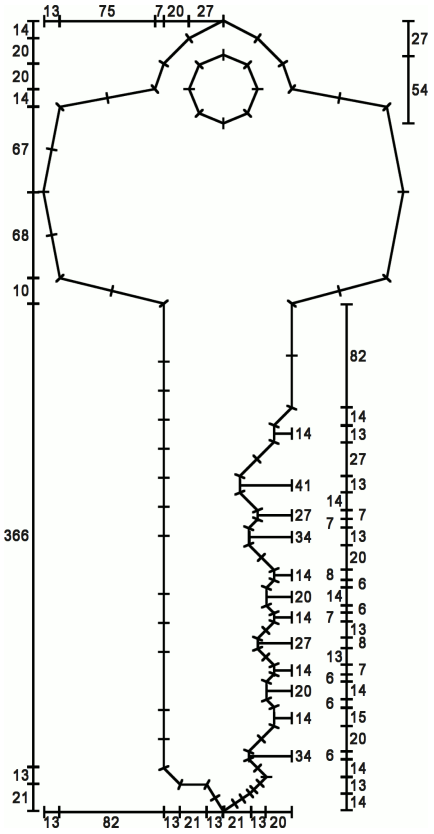


Figura A.5: Geometria do exemplo da Chave.

Referências Bibliográficas

[Andrews 1991] ANDREWS, G. R. *Concurrent Programming - Principles and Practice*. 1st. ed. [S.l.]: Addison-Wesley, 1991.

[Angel 2008] ANGEL, E. *Interactive Computer Graphics - A Top-Down Approach Using OpenGL*. 5th. ed. [S.l.]: Addison Wesley, 2008.

[Barker et al. 2004] BARKER, K.; CHERNIKOV, A.; CHRISOCHOIDES, N.; PINGALI, K. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems*, v. 15, n. 2, p. 183–192, 2004.

[Ben-Ari 2006] BEN-ARI, M. *Principles of Concurrent and Distributed Programming*. 2nd. ed. [S.l.]: Addison-Wesley, 2006.

[Berg et al. 2000] BERG, M. de; KREVELD, M. van; OVERMARS, M.; SCHWARZKPOF, O. *Computational Geometry: Algorithms and Applications*. 2nd. ed. [S.l.]: Springer-Verlag, 2000.

[Biswas et al. 2000] 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.

[Carvalho e Figueiredo 1991] 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.

[Cavalcante-Neto 1994] 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.

[Cavalcante-Neto 1998] 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.

[Cavalcante-Neto et al. 2001] CAVALCANTE-NETO, J. B.; WAWRZYNEK, P. A.; CARVALHO, M. T. M.; 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.

[Chernikov e Chrisochoides 2006] 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.

[Chrisochoides 2005] CHRISOCHOIDES, N. Parallel mesh generation. *Numerical Solution of Partial Differential Equations on Parallel Computers*, Springer-Verlag, v. 51, p. 237–259, 2005.

[Chrisochoides e Nave 2000] CHRISOCHOIDES, N.; NAVE, D. Simultaneous mesh generation and partitioning for Delaunay meshes. *Mathematics and Computers in Simulation*, v. 54, n. 4-5, p. 321–339, 2000.

[Das et al. 2001] DAS, S. K.; HARVEY, D. J.; BISWAS, R. Parallel processing of adaptive meshes with load balancing. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n. 12, p. 1269–1280, 2001.

[deCougny e Shephard 1999] DECOUGNY, 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.

[Edelsbrunner 2006] EDELSBRUNNER, H. *Geometry and Topology for Mesh Generation*. 1st. ed. [S.l.]: Cambridge University Press, 2006.

[ElGindy 1986] 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.

[Farrashkhalvat e Miles 2003] FARRASHKHALVAT, M.; MILES, J. P. *Basic Structured Grid Generation - With an Introduction to Unstructured Grid Generation*. 1st. ed. [S.l.]: Butterworth-Heinemann, 2003.

[Flynn 1972] FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, v. 21, n. 9, p. 948–960, 1972.

[Frey e George 2000] FREY, P. J.; GEORGE, P. L. *Mesh Generation - Application to Finite Elements*. 1st. ed. [S.l.]: ISTE Publishing Company, 2000.

[George e Borouchaki 1998] GEORGE, P.-L.; BOROUCHAKI, H. *Delaunay Triangulation and Meshing - Application to Finite Elements*. [S.l.]: Hermes, 1998.

[Globisch 1995] GLOBISCH, G. PARMESH – A parallel mesh generator. *Parallel Computing*, v. 25, p. 509–524, 1995.

[Golub e Loan 1996] GOLUB, G. H.; LOAN, C. F. V. *Matrix Computations*. 3rd. ed. [S.l.]: The Johns Hopkins University Press, 1996.

[Grama et al. 2003] GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V. *Introduction to Parallel Computing*. 2nd. ed. [S.l.]: Addison Wesley, 2003.

[Hearn e Baker 1996] HEARN, D.; BAKER, M. P. *Computer Graphics - C Version*. 2nd. ed. [S.l.]: Prentice Hall, 1996.

[Hjelle e Dæhlen 2006] HJELLE, Ø.; DÆHLEN, M. *Triangulation and Applications - Mathematics and Visualizations*. 1st. ed. [S.l.]: Springer, 2006.

[Hodgson e Jimack 1996] 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.

[Ito et al. 2007] ITO, Y.; SHIH, A. M.; ERUKALA, A. K.; SONI, B. K.; CHERNIKOV, A.; 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.

[Jones e Plassman 2000] JONES, M. T.; PLASSMAN, P. E. Unstructured mesh computations on networks of workstations. *Computer-Aided Civil and Infrastructure Engineering*, v. 15, n. 3, p. 196–208, 2000.

[Kohout et al. 2005] 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.

[Lämmer e Burghardt 2000] LÄMMER, L.; BURGHARDT, M. Parallel generation of triangular and quadrilateral meshes. *Advances in Engineering Software*, v. 31, n. 12, p. 929–936, 2000.

[Lawlor et al. 2006] LAWLOR, O. S.; CHAKRAVORTY, S.; WILMARTH, T. L.; CHOUDHURY, N.; DOOLEY, I.; ZHENG, G.; KALÉ, L. V. ParFUM: A parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, v. 22, n. 3-4, p. 215–235, 2006.

[Löhner 2001] LÖHNER, R. A parallel advancing front grid generation scheme. *International Journal for Numerical Methods in Engineering*, v. 51, n. 6, p. 663–678, 2001.

[Merks 1986] 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.

[Minyard e Kallinderis 2000] MINYARD, T.; KALLINDERIS, Y. Parallel load balancing for dynamic execution environments. *Computer Methods in Applied Mechanics and Engineering*, v. 189, n. 4, p. 1295–1309, 2000.

[Miranda et al. 1999] MIRANDA, A. C. 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*. Campinas, São Paulo, Brasil: IEEE Computer Society, 1999. p. 29–38.

[Moretti 2001] 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.

[MPI Forum] MPI Forum. The message passing interface (MPI) standard. Disponível em: <http://www.mcs.anl.gov/research/projects/mpi>.

[Okusanya e Peraire 1996] 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*. Mississipi, Estados Unidos: Mississippi State University, 1996. p. 719–729.

[Oliker e Biswas 1998] OLIKER, L.; BISWAS, R. PLUM: Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, v. 52, n. 2, p. 150–177, 1998.

[OpenMP Architecture Review Board] OpenMP Architecture Review Board. The OpenMP API specification for parallel programming. Disponível em: <http://www.openmp.org>.

[Owen 1998] OWEN, S. J. A survey of unstructured mesh generation technology. In: *Proceedings of the 7th International Meshing Roundtable*. Michigan, Estados Unidos: Sandia National Laboratory, 1998. p. 239–267.

[Persson 2004] PERSSON, P. O. PDE-based gradient limiting for mesh size functions. In: *Proceedings of the 13th International Meshing Roundtable*. Virgínia, Estados Unidos: Sandia National Laboratory, 2004. p. 377–387.

[Phongthanapanich e Dechaumphai 2004] PHONGTHANAPANICH, S.; DECHAUMPHAI, P. Adaptive Delaunay triangulation with object-oriented programming for crack propagation analysis. *Finite Element Analysis and Design*, v. 40, n. 13-14, p. 1753–1771, 2004.

[Preparata e Shamos 1991] PREPARATA, F. P.; SHAMOS, M. I. *Computational Geometry: An Introduction*. 2nd. ed. [S.l.]: Springer-Verlag, 1991.

[Ruppert 1999] RUPPERT, J. A new and simple algorithm for quality 2-dimensional mesh generation. In: *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms*. Texas, Estados Unidos: SIAM - Society for Industrial and Applied Mathematics, 1999. p. 83–62.

[Segerlind 1984] SEGERLIND, L. J. *Applied Finite Element Analysis*. 2nd. ed. [S.l.]: John Wiley and Sons, 1984.

[SGI - Silicon Graphics International] SGI - Silicon Graphics International. OpenGL - the industry's foundation for high performance graphics. Disponível em: <http://www.opengl.org>.

[Shewchuk 1997] SHEWCHUK, J. R. *Delaunay Refinement Mesh Generation*. Tese (Doutorado) — School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997.

[Shreiner et al. 2007] SHREINER, D.; WOO, M.; NEIDER, J.; DAVIS, T. *OpenGL Programming Guide*. 6th. ed. [S.l.]: Addison-Wesley, 2007.

[Tang et al. 1993] TANG, 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, Estados Unidos: Association for Computing Machinery, 1993. p. 112–121.

[Wilson e Topping 1998] 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.

[wxTeam] wxTeam. wxWidgets - cross-platform GUI library. Disponível em: <http://www.wxwidgets.org>.

[Yoshimura et al. 1999] YOSHIMURA, S.; WADA, Y.; YAGAWA, G. Automatic mesh generation of quadrilateral elements using intelligent local approach. *Computer Methods in Applied Mechanics and Engineering*, v. 179, n. 1, p. 125–138, 1999.

[Zagaris et al. 2009] ZAGARIS, G.; PIRZADEH, S. Z.; CHRISOCHOIDES, N. A framework for parallel unstructured grid generation for practical aerodynamic simulations. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. Flórida, Estados Unidos: AIAA - American Institute of Aeronautics and Astronautics, 2009.

Glossário

AFT	*Advancing front technique*
CAD	*Computer aided design*
CG	Computação gráfica
DAG	*Directed acyclic graph*
DDR	*Double Data Rate*
FEM	*Finite element methods*
GIS	*Geographical information systems*
HPC	*High performance computing*
HT	*Hyper-threading*
MEF	Método de elementos finitos
MIMD	*Multiple instruction, multiple data*
MISD	*Multiple instruction, single data*
MPI	*Message Passing Interface*
MPMD	*Multiple program, multiple data*
RAM	*Random Access Memory*
SIG	Sistemas de informações geográficas
SIMD	*Single instruction, multiple data*
SISD	*Single instruction, single data*
SPMD	*Single program, multiple data*