UFC

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DANIEL NASCIMENTO TEIXEIRA

UMA TÉCNICA DE DECOMPOSIÇÃO A PRIORI PARA GERAÇÃO PARALELA DE MALHAS BIDIMENSIONAIS

FORTALEZA, CEARÁ

2014

DANIEL NASCIMENTO TEIXEIRA

UMA TÉCNICA DE DECOMPOSIÇÃO A PRIORI PARA GERAÇÃO PARALELA DE MALHAS BIDIMENSIONAIS

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

Área de concentração: Computação Gráfica

Orientador: Prof. Dr. Joaquim Bento Cavalcante Neto

Coorientador: Prof. Dr. Creto Augusto Vidal

FORTALEZA, CEARÁ

2014

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

T265t Teixeira, Daniel Nascimento.
Uma técnica de decomposição a priori para geração paralela de malhas bidimensionais / Daniel Nascimento Teixeira. – 2014.
94 f. : il. color., enc. ; 30 cm.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2014.
Área de Concentração: Computação Gráfica.
Orientação: Prof. Dr. Joaquim Bento Cavalcante Neto.
Coorientação: Prof. Dr. Creto Augusto Vidal.

1. Computação de alto desempenho. 2. Estruturas de dados (Computação). 3. Algoritmos computacionais. I. Título.

CDD 005

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter-me dado forças e motivação para chegar até onde estou.

Aos meus pais Arimatéa e Doralice que tanto se esforçaram para que eu tivesse uma boa formação. Aos meus irmão Davi e Araceli pela companhia e pela ajuda que me ajudaram da forma que eles podiam para melhorar meu trabalho. Aos meus tios, primos e avós que sempre se importaram comigo e ajudaram na minha criação.

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 meus amigos de laboratório Yuri Lenon, Roberto, Rubens, Teófilo, Martha, Rafael Ivo, Jonas, Caio, Suzana, Rafael Siqueira, Laise, Lílian, Arnaldo, Ricado Lenz, Danilo, e aos outros nomes que tenha esquecido, pelas conversas e discussões sobre os mais variados temas. Em especial gostaria de agradecer ao Markos Freitas por acompanhar minha pesquisa desde a minha graduação e por me ajudar sempre que pôde. Agradeço também a todos os amigos que tanto torceram por mim e aos que estiveram presente na apresentação deste trabalho em especial a Fabrícia Timbó, Guilherme, Rafael Lima, entre outros que não foram lembrados aqui.

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 CENAPAD-UFC por permitir acesso ao seu *cluster* para rodar minha aplicação.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), e ao Programa de Orientação e Operacionalização da Pós-Graduação (PROPAG) pelo apoio financeiro para a realização deste trabalho.

RESUMO

Este trabalho descreve uma técnica de decomposição de domínios bidimensionais para geração em paralelo de malhas. Esta técnica funciona tanto para memória distribuída quanto compartilhada, além de permitir que se utilize qualquer estrutura de dados que gere regiões quadrangulares paralelas aos eixos para decompor o domínio dado como entrada. Pode se utilizar por exemplo, uma árvore quaternária (*quadtree*) ou uma partição binária do espaço (*bsp*). Além disso, qualquer processo de geração de malha que respeite os pré-requisitos estabelecidos pode ser empregado nos subdomínios criados, como as técnicas de Delaunay ou Avanço de Fronteira, dentre outras. A técnica proposta é dita *a priori* porque a malha de interface entre os subdomínios é gerada antes das suas malhas internas. A estimativa de carga de processamento associada a cada subdomínio é feita nesse trabalho com a ajuda de uma *quadtree* refinada, cujo nível de refinamento orienta a criação das arestas que são definidas a partir da discretização das fronteiras das células internas. Essa maneira de estimar carga produz resultados que representam, com boa precisão, o número de elementos a serem gerados em cada subdomínio. Isso contribui para um bom particionamento do domínio, fazendo com que a geração de malha em paralelo seja significativamente mais rápida do que a geração serial. Além disso, a qualidade da malha gerada em paralelo é qualitativamente equivalente àquela gerada serialmente, dentro de limites aceitáveis.

Palavras-chave: Decomposição de domínios. Geometria computacional. Geração em paralelo de malhas.

ABSTRACT

This work describes a technique of two-dimensional domain decomposition for parallel mesh generation. This technique works for both distributed and shared memory and has the freedom to use any data structure that manages rectangular regions parallel to the axes to decompose the domain given as input, such as a quaternary tree (quadtree) or a binary space decomposition (bsp), for example. Any process of mesh generation that respects the prerequisites established can be used in the subdomains created, for instance, Delaunay or Advancing Front, among others. This technique is called a priori because the mesh on the interface of the sub-domains is generated prior to the their internal meshes. The load estimation for each sub-domain in this work is performed with the aid of a refined quadtree, whose level of refinement guides the creation of edges that are defined from the bounderies of only inner cells. This way of estimate load produces results that accurately represent the number of elements to be generated in each subdomain. That contributes to a good partitioning of the domain, making the mesh generation in parallel be significantly faster than the serial generation. Furthermore, the quality of the generated mesh in parallel is qualitatively equivalent to that generated serially within acceptable limits.

Keywords: Domain decomposition. Computational geometry. Parallel mesh generation.

LISTA DE FIGURAS

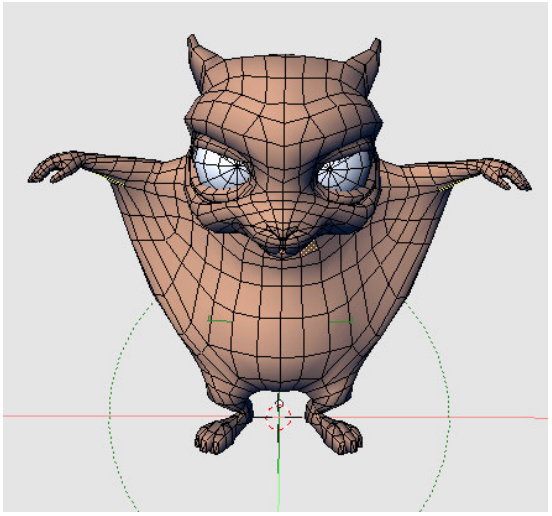
SUMÁRIO

1 INTRODUÇÃO

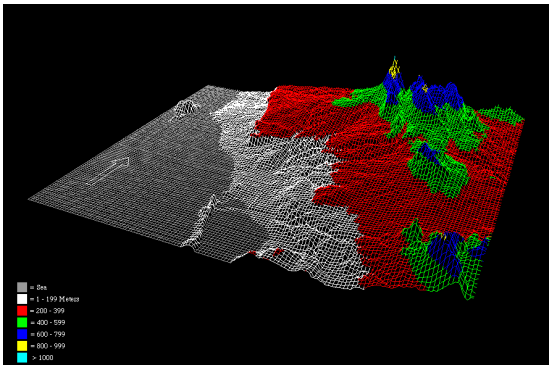
1.1 Motivação

O poder de processamento dos computadores vem crescendo muito nos últimos anos. Já é comum, inclusive, computadores pessoais terem mais de um processador. Desenvolver programas ou algoritmos que não utilizam completamente os recursos disponíveis nas máquinas resulta em um desperdício de capacidade de processamento.

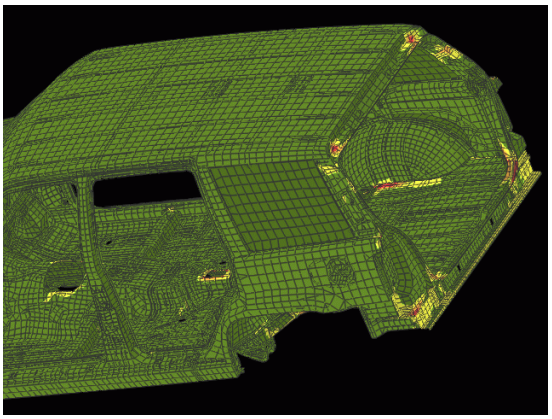
Ainda é preciso, entretanto, desenvolver técnicas que tentem alcançar uma boa escalabilidade, ou seja, ter um aumento no seu desempenho quando mais recursos computacionais forem disponíveis, mantendo ou melhorando os resultados obtidos pelo algoritmo sequencial.



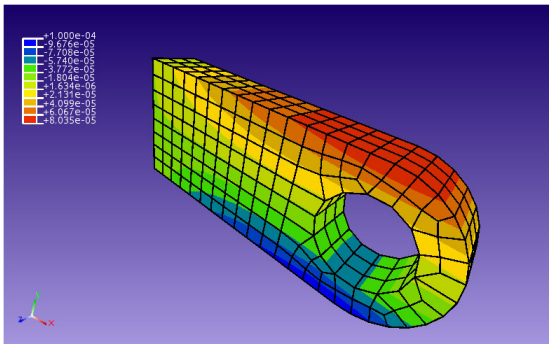
(a) Malha de computação gráfica. Fonte: www.yofrankie.org.



(b) Malha de sistemas de informação geográfica. Fonte: classics.uc.edu.



(c) Malha de projetos assistidos por computadores. Fonte: www.research.ibm.com.



(d) Malha utilizada na engenharia para MEF.

Figura 1.1: Exemplos de aplicações para malhas.

Malhas estão presentes no nosso dia a dia e são amplamente utilizadas para representar geometrias e outras informações. Alguns exemplos de aplicações seriam: em computação

gráfica (CG) (Figura 1.1a), sistemas de informações geográficas (SIG) (Figura 1.1b), projetos assistidos por computadores (CAD) (Figura 1.1c) e na engenharia, ajudando na análise e simulação de fenômenos físicos que utilizam métodos de elementos finitos (MEF) (Figura 1.1d).

Grandes malhas são necessárias em aplicações reais para obter bons resultados. Além do tamanho, a qualidade dos elementos da malha é fundamental em algumas aplicações como as da Engenharia. Então, para um bom programa de geração de malha executar, é necessário que ele também rode em paralelo e garanta que a malha gerada seja tão boa quanto a malha gerada sequencialmente. No MEF, por exemplo, se uma malha tiver uma grande quantidade de elementos ruins (polígonos, em malhas bidimensionais), é possível que este não convirja. Por necessitar de malhas bastante refinadas, é comum que as análises de elementos finitos usem malhas grandes, com milhões de elementos.

A geração de malha em paralelo geralmente envolve duas fases distintas, a decomposição ou partição do domínio em partes menores, que são enviadas para os processadores que estão disponíveis e a geração de malha em cada uma dessas partes, chamadas de subdomínios. Em geral se usa a mesma técnica de geração para cada um dos subdomínios usados, como triangulação de Delaunay ou Avanço de Fronteira, por exemplo, o que pode ser uma restrição para a geração. Além disso, deve-se procurar obter a melhor escalabilidade possível, a fim de utilizar corretamente todos os recursos disponíveis dos computadores.

1.2 Objetivos e Contribuições

O principal objetivo deste trabalho é descrever uma técnica *a priori* de subdivisão de domínios bidimensionais para geração paralela da malha. A técnica proposta é dita *a priori* porque a malha de interface entre os subdomínios é gerada antes das suas malhas internas. Essa técnica gera novos domínios e permite abstrair a técnica de geração de malha aplicada aos subdomínios, podendo-se combinar, por exemplo, as técnicas de Delaunay e Avanço de Fronteira, dentre outras. Além disso, a técnica proposta permite também abstrair o tipo de arquitetura de memória a ser usada (compartilhada ou distribuída).

A técnica foi projetada para atender aos seguintes requisitos:

1. Respeitar a fronteira de entrada, discretizada em segmentos, sem efetuar refinamentos;
2. Produzir bons elementos, evitando-se elementos com proporções ruins;
3. Proporcionar boas transições entre as regiões muito refinadas, com muitos triângulos pequenos, e regiões grosseiras, com poucos triângulos grandes, da malha;
4. Manter a compatibilidade das fronteiras (segmentos de interface) dos subdomínios criados com as de seus vizinhos;
5. Abstrair a técnica de geração de malha, podendo-se combinar mais de uma técnica;
6. Abstrair o tipo de arquitetura de memória a ser usada (compartilhada ou distribuída); e
7. Tentar gerar malhas da forma mais eficiente possível.

1.3 Organização do Trabalho

O restante deste trabalho está organizado em quatro capítulos. No Capítulo 2, apresentam-se alguns conceitos de Geometria Computacional e de Computação de Alto Desempenho necessários à compreensão dos capítulos subsequentes. No Capítulo 3, é apresentada uma visão geral das técnicas de subdivisão de domínios para geração paralela de malhas. No Capítulo 4, é apresentada a técnica de subdivisão de domínios bidimensionais desenvolvida, detalhando a técnica e a criação das estruturas de dados envolvidas no processo de estimativa da carga e particionamento do domínio. No Capítulo 5, são apresentados os resultados dos testes preparados para demonstrar a eficácia da técnica proposta, e os resultados obtidos são analisados. Por fim, no Capítulo 6, são apresentadas as conclusões sobre o trabalho e são propostas algumas ideias para trabalhos futuros.

2 CONCEITOS E DEFINIÇÕES

Neste capítulo, é apresentada uma revisão dos principais conceitos que serão úteis no decorrer da dissertação. Os conceitos sobre geometria computacional são apresentados na Seção 2.1. As técnicas de geração de malha, na Seção 2.2. Alguns conceitos de Computação de Alto Desempenho, bem como estratégias para administrar tarefas em paralelo, na Seção 2.4. Por fim, algumas estruturas de dados que serão utilizadas nesse trabalho são apresentadas, na Seção 2.5.

2.1 Geometria Computacional

A geometria computacional é a área da computação que estuda soluções e estruturas de dados para problemas geométricos. O seu enfoque é buscar soluções sob o ponto de vista da análise de complexidade de algoritmos. Entre os problemas estudados estão a construção de fechos convexos, triangulações, ordenação de pontos espaciais, interseções de retas/planos, malhas e outros mais.

2.1.1 Fecho Convexo

O fecho convexo de um conjunto finito de pontos é o menor conjunto convexo que contém tais pontos. Segundo (CARVALHO; FIGUEIREDO, 1991) um conjunto K do $\Re^n$, sendo n um inteiro não negativo, é convexo se quaisquer que sejam $x \in K$, $y \in K$ e $0 \leq \lambda \leq 1$, $\lambda \in \Re$, tem-se $\lambda x + (1-\lambda)y \in K$. Ou seja, todas as combinações convexas dos elementos de K pertencem a K. Um ponto p é dito ser combinação convexa dos pontos $p_i \in K$ se $p = \sum_{i=1}^{|K|} p_i \lambda_i$, sendo $\sum_{i=1}^{|K|} \lambda_i = 1$.

Um ponto $w \in K$ é um ponto extremo de K se não pode ser conectado a um elemento de K por um segmento de reta aberto pertencente a K. O fecho convexo de um conjunto finito C de pontos no $\Re^n$, sendo n um inteiro não negativo, é o conjunto de todas as combinações convexas de elementos de C. Em outras palavras pode-se dizer que o fecho convexo é um conjunto finito de pontos com menor área geométrica que engloba todos os pontos do conjunto (Figura 2.1).

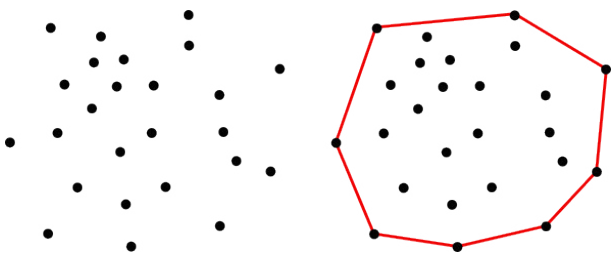


Figura 2.1: Conjunto de pontos e o seu fecho convexo.

2.1.2 Ponto de Steiner

No triângulo ABC, sejam:

- A' o simétrico de A em relação a G
- B' o simétrico de B em relação a G
- C' o simétrico de C em relação a G;

As três circunferências definidas pelos conjuntos de ternos de pontos AB'C', BA'C', CA'B' intersectam-se num ponto do circuncírculo - ponto de Steiner. A cada uma das três circunferências dá-se o nome de "círculo de Steiner", como mostra a figura 2.2.

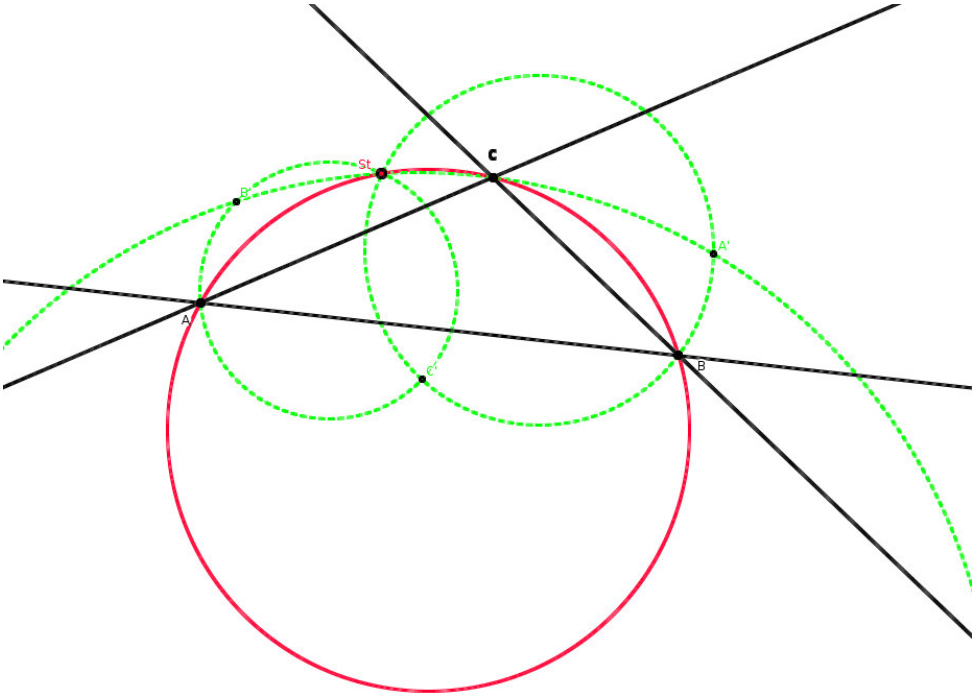


Figura 2.2: Ponto de Steiner para um triângulo ABC.

2.1.3 Triangulação e Malha

Triangulações são muitas vezes chamadas de malhas ou usadas como malhas, como no caso do método dos elementos finitos (MEF), já que malha é uma união de elementos, que podem ser triângulos, por exemplo. Pode-se definir uma malha M de uma maneira genérica como:

- $\Omega = \bigcup_{k \in M} k$, onde Ω é um domínio finito limitado,
- O interior de cada elemento k em M é não vazio,

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

Uma triangulação também é uma malha, mas nem toda malha é uma triangulação. Neste trabalho quando for mencionado malha, será sempre a malha que respeita as mesmas propriedades de uma triangulação. A definição de triangulação segundo (DæHLEN; HJELLE, 2006) diz que:

- Nenhum triangulo pertencente à triangulação pode ter pontos colineares.
- A interseção do interior de quaisquer dois triângulos pertencentes à triangulação é vazia.
- As bordas de 2 triângulos quaisquer só podem fazer interseção com vértices ou arestas.
- A união de todos os triângulos da triangulação é igual ao domínio.
- O domínio deve ser conectado.
- Não deve existir buracos na triangulação, a menos que eles sejam definidos como entrada.
- Se um triângulo está na borda da triangulação então ele faz interseção por aresta no máximo dois triângulos.

Existem malhas de diferentes geometrias e dimensões. Caso a topologia de elementos e vértices da malha siga alguma regra simples de indexação, essa malha será definida como estruturada, caso contrário, ela é definida como não-estruturada. Nas malhas estruturadas o conhecimento dos vizinhos de cada elemento não depende do armazenamento ou existência desta informação. Já nas não-estruturadas, para se ter conhecimento dos vizinhos, é necessário armazenar ou calcular estas informações. Existem ainda as malhas mistas, que combinam malhas estruturadas e não-estruturadas (Figura 2.3).

As malhas também podem ser classificadas de acordo com a geometria dos seus elementos, já que não são necessariamente formadas somente de triângulos.. Para malhas bidimensionais, por exemplo, elas podem ser de elementos triangulares ou quadrilaterais, por exemplo, como mostra a Figura 2.3.

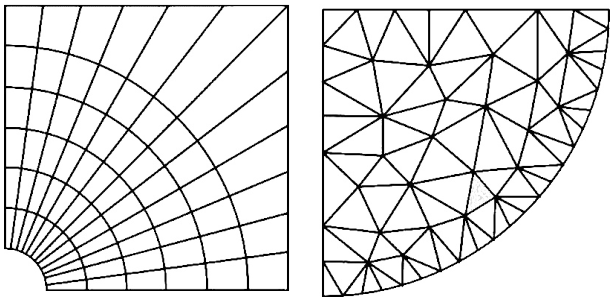


Figura 2.3: Exemplo de malha quadrilateral estruturada e não-estruturada triangular.

Para muitas aplicações, a qualidade dos elementos da malha é muito importante. Para classificar um elemento de uma malha triangular como bom ou ruim pode-se utilizar, dentre

outras, uma métrica que é definida como $\alpha = 2R_i/R_c$, 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. 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 de boa qualidade, como mostra a Figura 2.4.

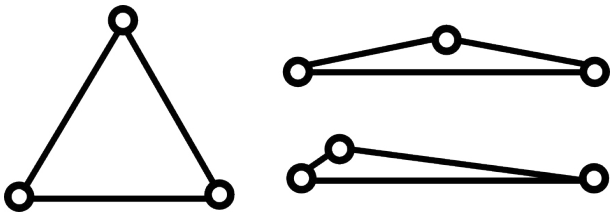


Figura 2.4: Exemplo de um elemento triangular de boa qualidade (esquerda) e dois de péssima qualidade (direita).

2.2 Geração de Malha

Nesta seção, são apresentadas as técnicas de geração de malhas triangulares mais conhecidas atualmente. Existem diversos algoritmos para geração de malhas, porém eles podem ser enquadrados uma das categorias a seguir:

- Avanço de fronteira, técnica em que a malha é gerada a partir da borda da região;
- Delaunay, técnica em que a malha é gerada procurando-se maximizar o menor ângulo dos triângulos gerados para um dado conjunto de pontos;
- Arbitrária, técnica em que a malha é gerada de maneira diferente das anteriores.

2.2.1 Avanço de Fronteira

Este é um dos métodos mais populares de geração de malhas e consiste em criar os elementos no interior do domínio progressivamente a partir de um contorno, especificando a região a ser preenchida (Figura 2.5a). Este contorno é chamado de fronteira inicial ou borda. Os elementos são gerados a partir dessa fronteira dada como entrada. Uma fronteira bidimensional é formada por um conjunto de arestas.

À medida que o algoritmo progride, a fronteira avança em direção ao interior, sempre removendo ou adicionando elementos de fronteira até que todo o domínio seja preenchido. O algoritmo chega ao fim quando não há mais fronteira, ou seja, o domínio foi totalmente triangularizado.

Há casos em que o algoritmo não consegue mais gerar elementos para uma determinada fronteira, isso indica que o algoritmo falhou. O caso de falha ocorre quando todos

os possíveis elementos a serem criados se sobrepõem a um elemento já existente. Por isso, é importante verificar se elementos se interceptam. Os casos de falha geralmente acontecem em modelos tridimensionais. Entretanto, já existem técnicas para contornar esses problemas e gerar malhas em modelos que falhariam.

Para gerar os novos triângulos no interior do domínio, é necessário criar novos pontos que não pertencem aos dados de entrada. Em geral, são utilizados os pontos de Steiner para isso (RUPPERT, 1999).

Um algoritmo de avanço de fronteira procede da seguinte maneira no caso 2D (Figura 2.5):

1. Selecione uma aresta da fronteira, a aresta base (fig. 2.5b);
2. Encontre um ponto ideal para a formação de um novo triângulo com a aresta base (fig. 2.5c);
3. Crie uma região de busca em torno desse ponto ideal (fig. 2.5d);
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, que pode ser um novo ponto ou um ponto já pertencente à malha;
5. Forme o novo triângulo com o ponto selecionado e adicione-o à malha (fig. 2.5e);
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.

Pelo fato da fronteira ser sempre respeitada, os algoritmos de avanço de fronteira têm facilidade em tratar regiões descontínuas, ou por conterem buracos, ou por serem regiões separadas. Como os elementos mais próximos da borda são gerados primeiro, em geral, eles têm uma boa qualidade. A boa qualidade da malha gerada provê estabilidade e precisão à aplicação de métodos numéricos (como os métodos dos elementos finitos).

Porém, nem sempre todos os elementos gerados têm boa qualidade. Ao contrário dos elementos mais próximos da borda, os elementos mais internos à malha nem sempre têm boa qualidade devido à região tornar-se menor à medida que a fronteira avança. Geralmente uma técnica de suavização ou otimização é aplicada na malha resultante do algoritmo para tratar esses casos.

2.2.2 Delaunay

Esta é uma técnica bastante conhecida na área de geração de malhas, cujo nome é uma homenagem ao matemático russo Boris Delaunay. A entrada para esse problema é um conjunto de pontos e, geralmente, não são utilizados os pontos de Steiner para formar os triângulos.

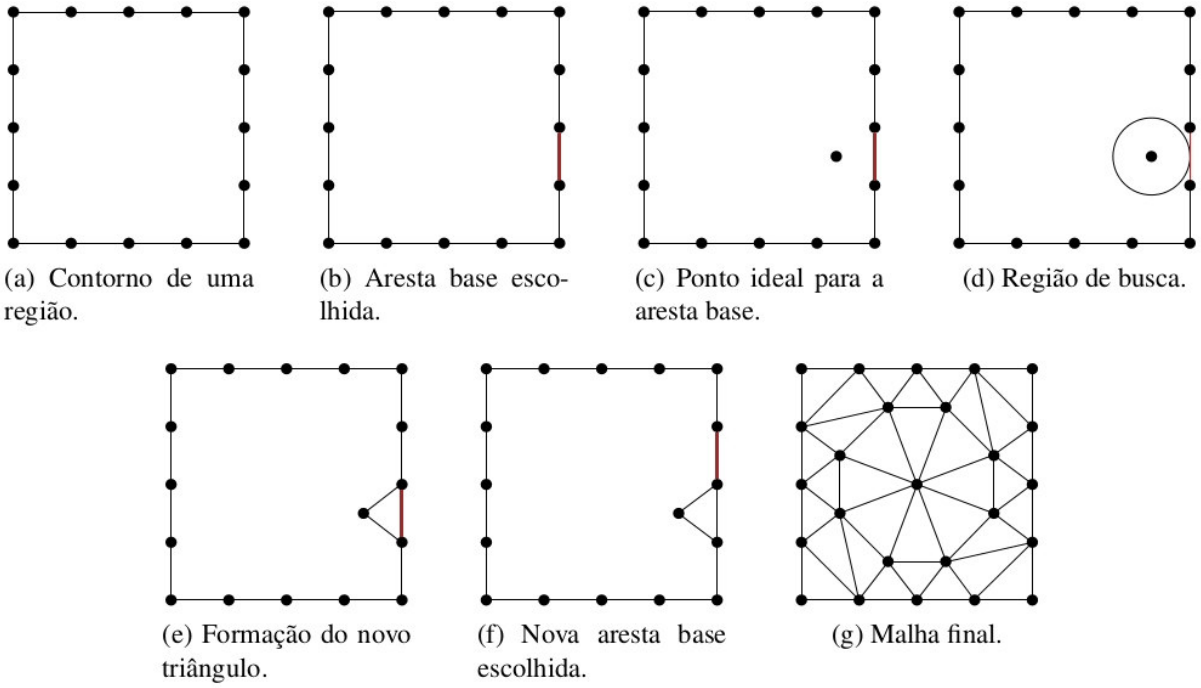


Figura 2.5: Avanço de fronteira (FREITAS, 2010).

O critério de Delaunay para a formação dos triângulos é que não exista nenhum outro ponto dentro do círculo que passa pelos três pontos desse triângulo (seu circuncírculo), critério este também chamado de "esfera vazia"(o circuncírculo desse triângulo, Figura 2.6). O critério de Delaunay em si não se constitui num método de geração de malhas, mas é uma forma de saber onde os pontos devem estar localizados no espaço.

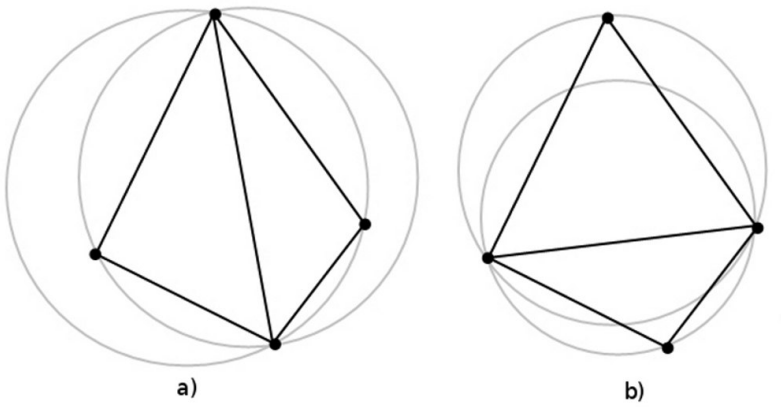
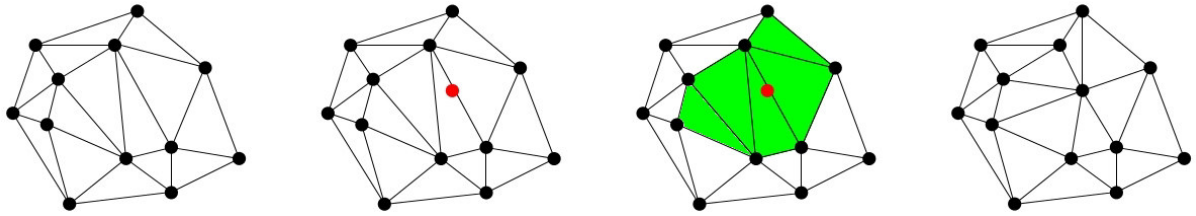


Figura 2.6: a) Critério de Delaunay falhando para os dois triângulos. b) Triangulação válida respeitando o critério de Delaunay.

A malha gerada por Delaunay visa maximizar os ângulos internos dos triângulos gerados, ou seja, dada uma aresta da triangulação de Delaunay, o ponto que forma o maior ângulo com essa aresta é o ponto que formará um triângulo de Delaunay com ela.

Existem algumas variações de algoritmos de Delaunay. Em uma delas, encontra-se uma aresta que faz parte da triangulação que é, em geral, uma aresta pertencente ao fecho convexo. A partir dela, é encontrado o ponto que formará um triângulo de Delaunay. Assim, com

as novas arestas, encontram-se novos triângulos, em um algoritmo parecido com o de avanço de fronteira. Uma outra variação é feita a partir de inserção de pontos. A entrada é uma malha triangular não necessariamente de Delaunay e se modifica essa malha (de apenas um subconjunto de pontos da entrada) pré-existente (Figura 2.7).



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

Figura 2.7: Triangulação por inserção de pontos (FREITAS, 2010).

Dependendo da disposição dos pontos da entrada, a triangulação final pode não ter boa qualidade, principalmente em regiões críticas, próximas à borda, gerando instabilidade em métodos numéricos. Uma alternativa para melhorar essa malha é fazer refinamentos e otimizações, que fazem uso de pontos de Steiner.

2.2.3 Arbitrária

As técnicas de geração de malha arbitrárias são aquelas que não se enquadram nem como Avanço de Fronteira e nem como Delaunay. As malhas são geradas em geral por algoritmos de varredura ou algum outro método.

Outro uso que essas malhas possuem é nas demonstrações de teoremas. O problema de ordenação de pontos pode ser reduzido ao problema de geração de malhas bidimensionais (CARVALHO; FIGUEIREDO, 1991). Prova-se por redução que pode ser gerada uma malha triangular a partir do fecho convexo de um conjunto de pontos em duas dimensões numa complexidade na ordem de $O(n \log n)$.

2.3 Geração de Malha em Paralelo

Na geração em paralelo é necessário dividir a entrada para realizar o processamento em paralelo das diversas partes. Existem duas formas de decompor o domínio. Na primeira forma, uma malha grosseira da região é rapidamente gerada, sequencialmente, e dividida entre os processadores. Essa forma, chamada de decomposição discreta do domínio, envolve ainda o problema de particionamento da malha. A segunda forma de decompor o domínio envolve dividir a região a partir de funções, segmentos, eixos inerciais, ou estruturas auxiliares, por isso chamada de decomposição contínua do domínio. Cada subdomínio é enviado a um processador, onde a malha será gerada.

Uma malha de interface é um conjunto de segmentos ou triângulos para o caso bidimensional ou, no caso tridimensional, um conjunto de triângulos ou tetraedros. Essa malha

de interface faz a conexão entre duas partições vizinhas e faz o papel de uma nova fronteira. A forma que ela é criada vai depender da técnica que está sendo utilizada para particionar o domínio.

O particionamento contínuo pode ainda, ser subdividido em duas categorias, dependendo da forma como é gerada a malha entre os subdomínios, chamada de malha de interface. Se essa malha for gerada antes da malha interna ao subdomínio, essa abordagem é chamada de *a priori*. Caso ela seja gerada depois, é chamada de *a posteriori*. A geração da malha de interface *a posteriori* geralmente requer sincronização entre processos.

2.4 Computação de Alto Desempenho

Computação de Alto Desempenho ou HPC (do inglês *High-performance computing*) se refere ao uso de *clusters* ou supercomputadores em tarefas que requerem grandes recursos de computação. O uso eficiente desses recursos é o principal foco de estudo nessa área. *Cluster* é um conjunto de computadores de alto desempenho interconectados por uma rede local que trabalham em conjunto como um único recurso de processamento.

2.4.1 Modelos de Arquiteturas

Em programas que executam sequencialmente não existe a preocupação que uma dada posição de memória seja alterada no mesmo tempo que ela esteja sendo lida. Em computação paralela há essa preocupação e existem várias técnicas para manter a ordem de leitura e escrita na memória. Basicamente há dois tipos de arquiteturas (Figura 2.8):

- Memória Compartilhada

  Engloba basicamente os sistemas UMA (Uniform Memory Acess), ou seja, o acesso à memória é feito de forma uniforme através de endereçamento direto. Assim, todos os processadores de um computador compartilham um mesmo espaço de memória e para isso deve haver um controle na leitura e escrita na memória.

- Memória Distribuída

  Cada um dos processadores têm acesso a um espaço único de endereçamento de memória privada. Cada módulo da memória pode ser acessado diretamente por apenas um dos processadores. A comunicação entre os processos ocorre através de troca de mensagens.

2.4.2 Balanceamento de Carga

Uma aplicação que executa em paralelo cria várias novas tarefas que devem ser distribuídas entre os processadores existentes. Quando a quantidade de tarefas se torna maior que a quantidade de processadores disponíveis, torna-se necessário um balanceamento de carga,

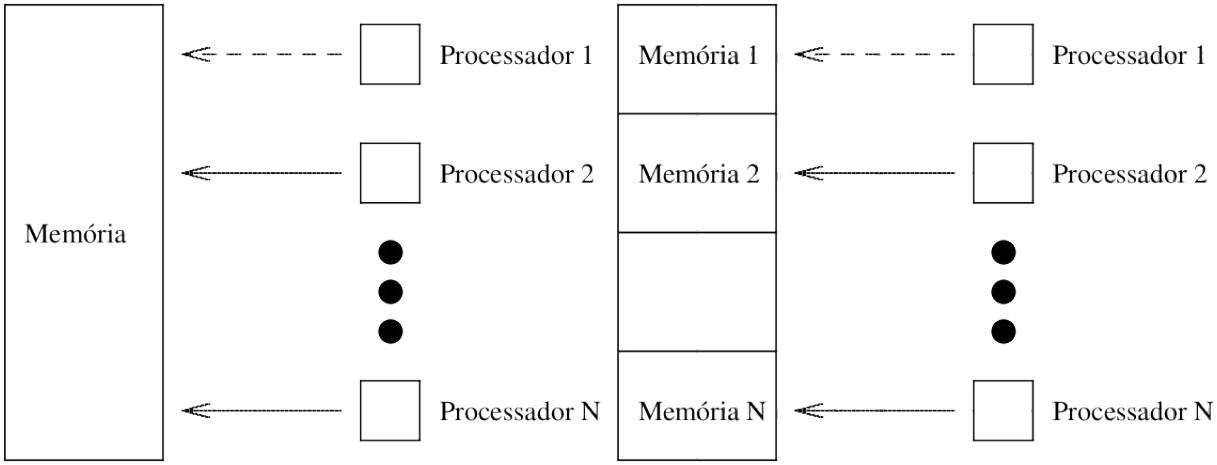


Figura 2.8: Estratégia de balanceamento não-centralizado onde todos os nós podem se comunicar entre si. Memória compartilhada à esquerda e memória distribuída à direita.

ou seja, distribuir o processamento entre os processadores de modo a obter a maior velocidade possível de execução. O principal objetivo é manter os processadores ocupados a maior parte do tempo possível evitando que alguns deles fiquem ociosos enquanto outros estão executando alguma tarefa.

Esse balanceamento pode ser estático (o balanceamento ocorre antes da execução de qualquer processo) ou dinâmico (é realizado durante a execução do processo). O principal problema de um balanceamento estático é a dificuldade em estimar com precisão os tempos de execução das várias partes do programa. O modelo de balanceamento dinâmico possui duas classificações:

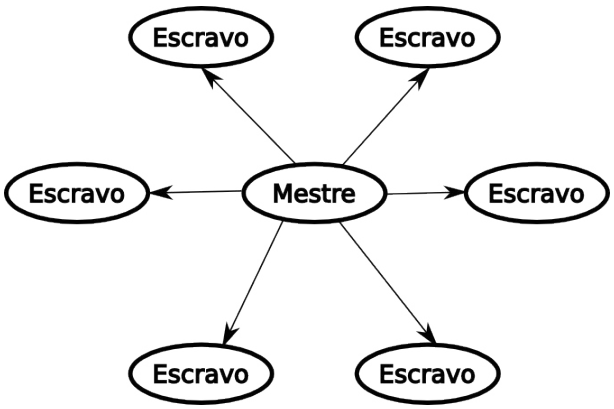


Figura 2.9: Estratégia de balanceamento centralizado mestre/escravo onde um processo controla todas as tarefas e os escravos solicitam e executam as mesmas.

1. Centralizado (Figura 2.9)

   - Todas as tarefas são manipuladas a partir de uma localização central.
   - Exemplo: Modelo mestre-escravo - o processo mestre mantém a coleção de tarefas a serem executadas e os processos escravos solicitam tarefas.

2. Não-centralizado (Figura 2.10)

    - Uma coleção de processos trabalhadores opera sobre um problema, e esses trabalhadores interagem entre si, reportando o resultado final a um único processo.
    - Exemplo: Algoritmo de *polling* aleatório - o processo P_i pede tarefas ao processo P_x, onde x é um número selecionado aleatoriamente entre 0 e $n-1$ (excluindo i).

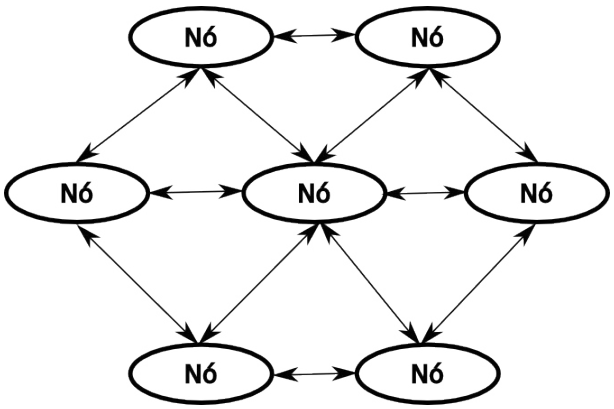


Figura 2.10: Estratégia de balanceamento não-centralizado onde os nós podem se comunicar entre si.

2.4.3 Métricas de Desempenho

Ao utilizar uma aplicação paralela, surge o interesse em saber o ganho de velocidade quando comparado a uma aplicação sequencial. Para isso existem algumas métricas utilizadas:

- Escalabilidade

  É a propriedade de um sistema que lhe confere a capacidade de aumentar seu desempenho sob uma determinada carga, quando mais recursos (processadores) são acrescentados a esse sistema. Ou seja, pode-se falar que um algoritmo é escalável se ele pode ser utilizado em uma grande quantidade de processadores sem que aconteça uma queda em sua velocidade.

  Pode-se dizer que um sistema é escalável quando ele resolve um problema de magnitude γ com um recurso R, e consegue resolver um problema de magnitude $n\gamma$ com um recurso nR. Ou seja, sempre que aumentar os recursos computacionais, aumentará proporcionalmente a capacidade de resolver problemas maiores.

- *Speed-up*

  Esta métrica mostra quantas vezes um programa paralelo é mais rápido que um serial. Para obter um *speed-up* linear tem-se que obter um programa com tempo de execução x vezes mais rápido quando aumentado em x o número de processadores. Já um *speed-up* super linear seria obter um ganho maior que x quando aumentado em x o número de processadores.

O *speed-up* S para p processadores é calculado pela seguinte formula: $S(p) = T_s/T_p$, onde T_s é o tempo de execução do programa sequencialmente e T_p é o tempo do programa executando em paralelo para p processadores.

Na prática um *speed-up* linear é difícil de se obter. À medida que a quantidade de processadores aumentam, a comunicação entre os processos aumenta e isso faz o tempo de execução cair, derrubando assim o *speed-up*.

2.5 Estrutura de Dados

Diversas estruturas de dados que foram criadas na área da computação são muito usadas em problemas de computação gráfica. No contexto desse trabalho, essas estruturas têm o objetivo de fazer uma decomposição espacial do domínio. Com essas decomposições, diversos cálculos são otimizados fazendo uma busca em qual parte da decomposição o objeto de interesse está e limitando os cálculos apenas aos elementos que pertencem a essa decomposição.

Essas estruturas são utilizadas em diversas aplicações como tratamento de colisão e renderização (Figura 2.11). Entre as estruturas de dados bidimensionais, as mais importantes são a *quadtree* e a *binary space partitioning* ou simplesmente BSP.

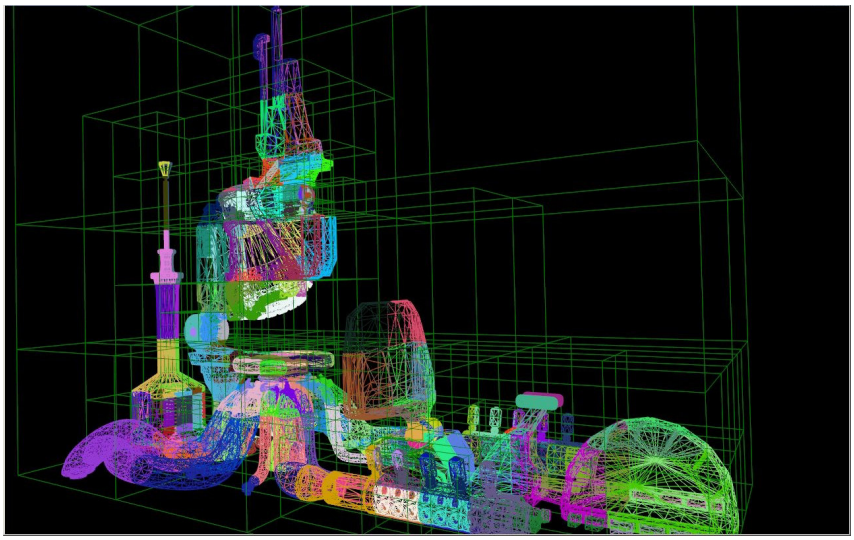


Figura 2.11: Exemplo de uma decomposição espacial feita para renderização e teste de colisão. Fonte: http://togeskov.net/

2.5.1 *Quadtree*

Uma *quadtree* é uma estrutura de dados baseada em árvore em que cada nó possui exatamente quatro filhos (Figura 2.12). Em geral *quadtrees* são utilizadas para decompor domínios bidimensionais em quatro regiões de mesmo tamanho. As *quadtrees* podem ser classificadas de acordo com o tipo do dado que elas representam (regiões, pontos, arestas, polígonos), isso vai depender do tipo de aplicação para o qual ele está sendo utilizada.

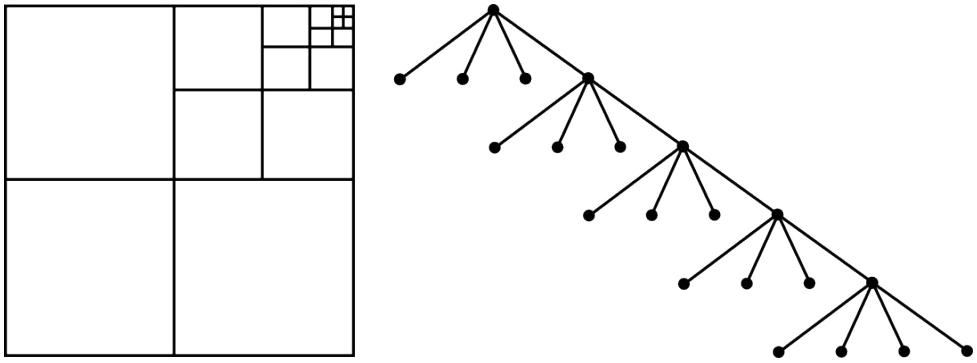


Figura 2.12: Uma subdivisão feita por *quadtree* com cinco níveis e sua representação em árvore.

2.5.2 *Binary Space Partitioning* (BSP)

BSP (particionamento binário espacial) é um processo genérico que, de forma recursiva, divide um domínio em duas partes, não necessariamente iguais, até que o particionamento do corte satisfaça um ou mais requisitos estabelecidos. Como resultado tem-se dois novos subespaços que podem ainda ser particionados recursivamente. O critério de posicionamento do corte e de parada no particionamento vai depender do objetivo que se deseja ao usar uma BSP.

Pode-se dizer que a BSP é um caso genérico da *quadtree*. A principal diferença entre elas basicamente é a quantidade de partições criadas (quatro para cada subdivisão na *quadtree* e duas na BSP) e a desvantagem está na hora de encontrar o melhor corte para a BSP, que pode ser bastante custoso se comparado com a *quadtree*.

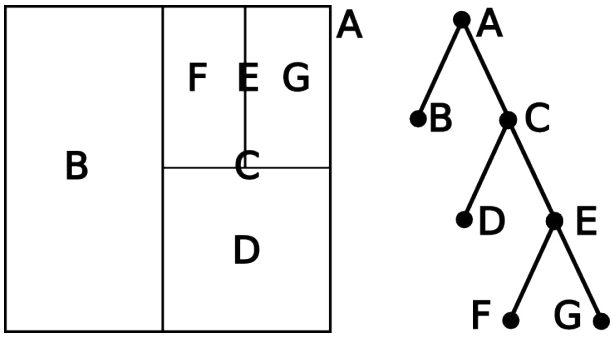


Figura 2.13: Uma subdivisão feita com BSP e sua representação em árvore.

3 TRABALHOS RELACIONADOS

Em Computação de Alto Desempenho é conhecido que, para um bom algoritmo paralelo executar, é preciso uma boa estratégia para a divisão da entrada e para a junção das várias soluções ao final.

Há ainda a preocupação com a distribuição das tarefas entre os processadores, levando em conta que, ao final, todos eles devam ter realizado uma quantidade similar de processamento, evitando que alguns processadores fiquem ociosos enquanto outros estão sobrecarregados. Se isto acontecer, significa que a carga foi devidamente balanceada entre os processadores.

Neste capítulo, é apresentado o que se tem feito nos últimos anos na área de subdivisão de domínios para geração em paralelo de malha, mostrando as vantagens e desvantagens de cada técnica apresentada. Alguns trabalhos são tridimensionais, mas que podem ser adaptados para duas dimensões e por isso são mencionados neste capítulo.

3.1 Decomposição Baseada na Distância/Volume/Centro de Massa

Em (PIRZADEH; ZAGARIS, 2009), é descrita uma técnica baseada em Avanço de Fronteiras e Avanço de Camadas com decomposição contínua *a priori*.

Inicialmente, é gerada uma malha de superfície nos pontos dados como entrada. Logo em seguida, é feita uma estimativa de carga nos subdomínios utilizando uma *octree* que usa a informação da quantidade de faces para subdividir o domínio. Se necessário, serão criados planos de partições que dividem o domínio em regiões com cargas aproximadamente iguais. As posições destes planos são definidas através do centro de densidade da malha. Esse centro indica onde a massa efetiva do sistema está concentrada.

Em seguida, são identificadas as faces que interceptam o plano de partição e uma malha parcial é gerada na região do plano de corte. Depois disso, para cada lado da partição são agrupadas as faces dos novos subdomínios. Esse processo é repetido até que um número máximo de subdivisões tenha ocorrido. Ao final da execução, tem que ser realizada uma junção de todas as submalhas. A Figura 3.1 ilustra os principais passos dessa técnica para o caso bidimensional.

Como vantagens pode-se citar que a utilização de avanço de camadas entre as partições faz com que a malha gerada seja praticamente idêntica a uma malha gerada sequencialmente, ou seja, não são gerados padrões entre as partições do domínio. Outra vantagem é que não é necessário nenhum pré-processamento custoso para definir ou construir as partições. Além disso, a construção da *octree* para estimar a carga é automática e de baixo custo.

Uma das desvantagens desse método é que nem sempre é fácil gerar as malhas nas partições, especialmente em três dimensões. Além disso, a qualidade dessas malhas pode ser ruim, prejudicando assim a qualidade da malha gerada no modelo todo. Basear a quantidade de subdivisões num número máximo não é uma boa métrica para controlar a geração dos subdomínios quando não se tem uma boa estimativa de carga, isso pode gerar subdomínios em excesso, aumentando a comunicação entre os processos.

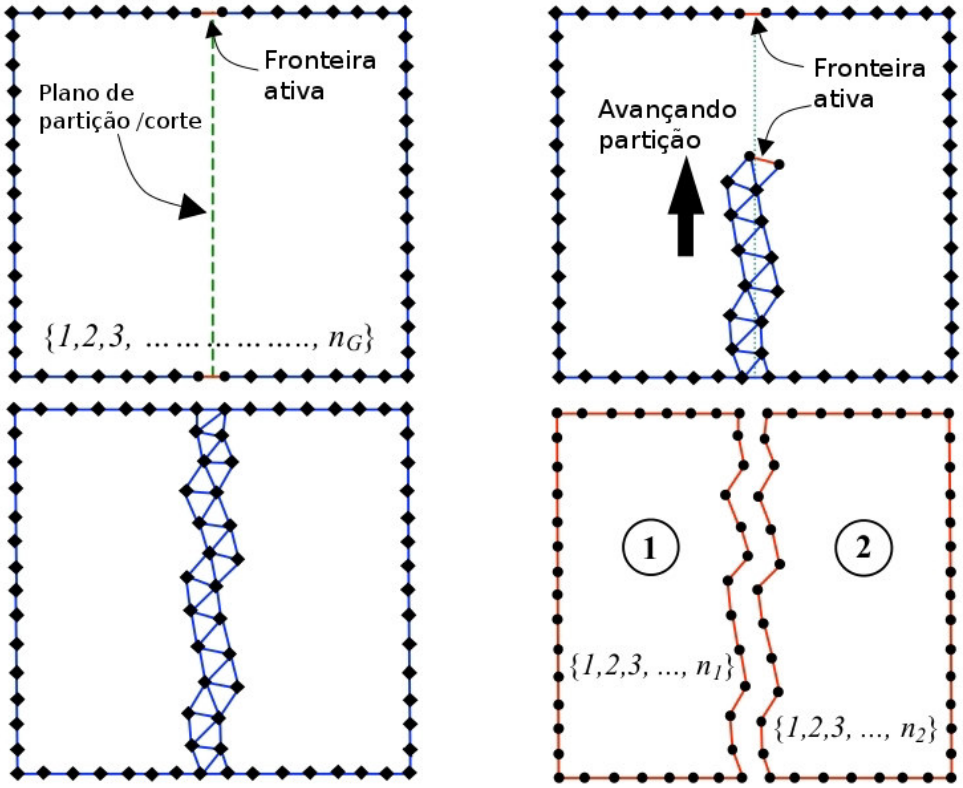


Figura 3.1: Os principais passos da técnica de (PIRZADEH; ZAGARIS, 2009) para gerar os segmentos de interface.

Em (IVANOV; ANDRÄ; KUDRYAVTSEV, 2006), foi desenvolvido um algoritmo baseado em Delaunay para geração de malhas tetraédricas em que o posicionamento do plano de corte é definido pelo centro de massa e pela matriz de inércia. O plano de corte é um plano perpendicular a um eixo que segue uma das três definições:

- Planos criados são equidistantes;
- Volume entre os planos são iguais;
- Passa pelo centro de massa.

A escolha do critério utilizado para criar os subdomínios depende da geometria da entrada. Assim, dependendo da entrada, um critério pode ser melhor que outro, mas isso depende do conhecimento do usuário. Na Figura 3.2, as três formas de decomposição são apresentadas.

Após ter o plano de corte definido, é feita uma suavização da seção de corte e a sua triangulação para, posteriormente, serem geradas as malhas nos subdomínios. Um problema bem visível nesse método é que para se ter um bom plano de corte é preciso ter um modelo com uma geometria bem comportada, sem forma côncava, alongada ou afinada. Nesse trabalho a quantidade de subdomínios gerados é maior que a de processos para tentar melhorar o balanceamento dinâmico, uma vez que não se tem uma boa precisão na estimativa da carga.

Uma solução parecida é a apresentada em (LÄMMER; BURGHARDT, 2000), em que o plano de corte é traçado da mesma maneira, porém em duas dimensões, ou seja, um eixo

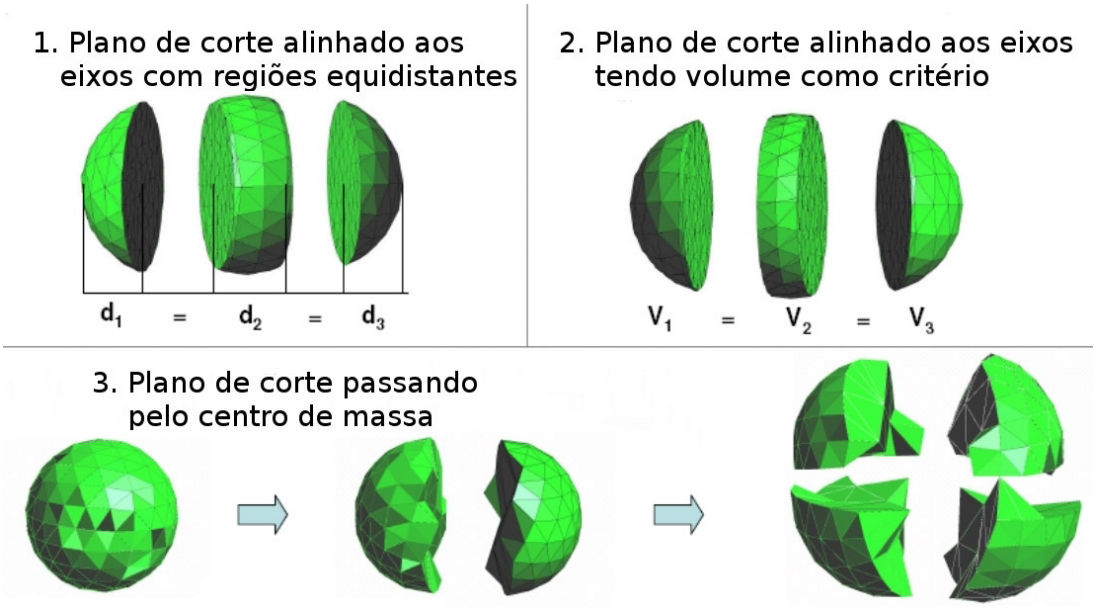


Figura 3.2: As três formas de particionar. 1 - planos equidistantes. 2 - volume dos subdomínios iguais. 3 - centro de massa (IVANOV; ANDRÄ; KUDRYAVTSEV, 2006).

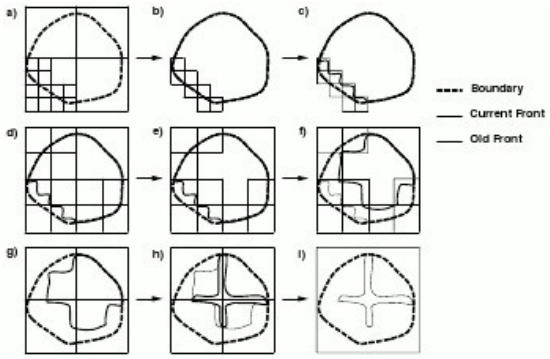
de corte. Este eixo é usado para dividir o domínio recursivamente. A partir do eixo, uma aresta é formada, e os valores nos seus pontos extremos são interpolados dos valores dados como entrada. Quando o número de subdomínios for igual ao número de processadores, uma malha de Delaunay é gerada em cada interior concorrentemente. Essa técnica não apresenta uma boa escalabilidade nos seus resultados.

3.2 Decomposição Baseada em Estruturas de Dados tipo Árvore

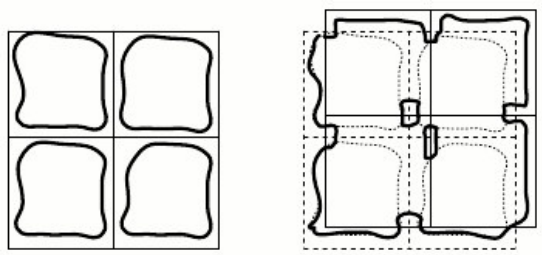
Em (DECOUGNY; SHEPHARD, 1999), a entrada do algoritmo é o contorno de um objeto. Cada processador fica com parte de uma *octree* distribuída, que define planos de corte do domínio. A malha das células internas é gerada concorrentemente com *templates*. A região entre o contorno e as células internas é preenchida por uma técnica de Avanço de Fronteira, onde são gerados os elementos internos a uma região delimitada pelos planos de corte. Por último é feita a conexão das malhas dos dois lados de cada plano e de suas intersecções. Essa técnica gera muitas partições, já que a cada subdivisão oito novos subdomínios são criados, e, por usar *templates*, esta técnica pode gerar uma quantidade excessiva de elementos, além de possivelmente gerar elementos de qualidade ruim nas regiões próximas ao contorno.

Na técnica de (LÖHNER, 2001), é gerada uma *octree* grosseira com relação ao contorno dado como entrada. Após essa geração, 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 possa cruzar as extremidades da célula que a contém. Cada octante sofre então um pequeno deslocamento na diagonal com o intuito de gerar mais elementos. Esse deslocamento elimina quase todas as faces entre duas ou mais células e diminui o tamanho da fronteira para o

próximo passo. Desse modo a nova fronteira é encontrada e uma nova *octree* é construída para ela, e o procedimento é repetido, até que não seja mais possível gerar malha. Na Figura 3.3, são mostrados os passos do algoritmo e os deslocamentos que são realizados.



(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 3.3: Técnica de (LÖHNER, 2001).

O deslocamento de cada octante é feito seguindo-se sempre o mesmo processo, e isso pode não ser o ideal para certos tipos de modelos, onde maneiras distintas de deslocamento poderiam ser mais eficientes (deslocamentos na diagonal, na direção dos eixos principais, entre outros).

Em (LARWOOD et al., 2003), é apresentada uma técnica de decomposição de domínio que tem como entrada uma triangulação de borda. Para saber quais subdomínios devem ser divididos, a técnica usa um critério baseado na quantidade de faces por subdomínio. A decomposição é feita recursivamente usando uma *octree* caso seja tridimensional ou uma *quadtree* caso seja bidimensional, verificando sempre se o número de faces de um subdomínio é menor do que o limite estipulado, e, enquanto a verificação for falsa, a decomposição ocorre. A quantidade máxima de subdivisões está limitada por uma constante maior que o número de processadores disponíveis. Isso evita a criação excessiva de partições e permite que um processador possa receber mais de uma tarefa ao longo da execução.

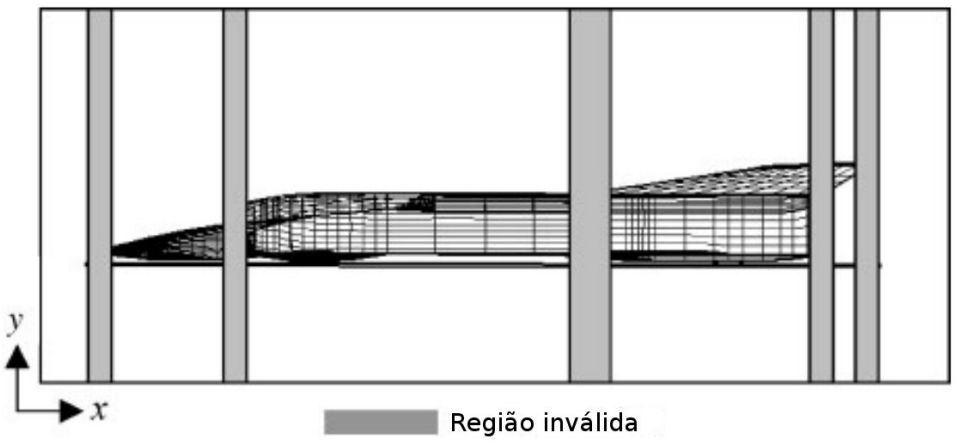


Figura 3.4: Regiões de corte inválidas em cinza (LARWOOD et al., 2003).

Para evitar a criação de elementos ruins, é feita uma verificação no corte baseada

no ângulo do vetor normal do plano de corte com a normal dos triângulos, de forma que o plano de corte não possa passar por triângulos com ângulo menor do que uma tolerância. Caso essa verificação falhe, a *octree* (*quadtree*, em 2D) sofre um deslocamento em um dos eixos. A Figura 3.4 mostra um exemplo onde alguns planos de corte falham nos testes.

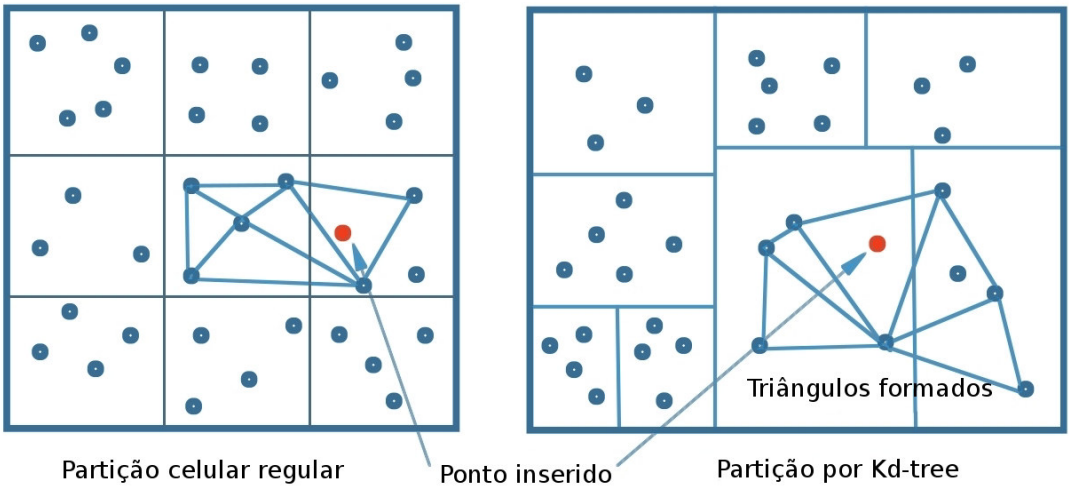


Figura 3.5: Pontos organizados em células. À esquerda por partição regular e à direita por *kd-tree* (LO, 2012).

Em (LO, 2012), uma técnica bidimensional para uma triangulação de Delaunay por inserção de pontos é apresentada. Uma *kd-tree* é utilizada para organizar os pontos da entrada em células. Estas células são agrupadas em zonas e distribuídas entre os processadores. A vantagem de usar uma *kd-tree* é que cada célula tem uma quantidade de pontos aproximadamente igual e a busca espacial é facilitada na hora de fazer a inserção de pontos em uma região. A Figura 3.5 mostra a organização de um conjunto de pontos por partição regular e por *kd-tree*.

A quantidade de subdomínios criados é compatível com a quantidade de processadores disponíveis, ou seja, cada processador terá que ficar responsável por uma zona. A inserção dos pontos em cada zona é feita em paralelo, sendo totalmente independente das outras zonas, e a malha gerada em cada zona também será independente.

Os triângulos gerados nas bordas das zonas têm pontos de uma zona vizinha. Isso irá gerar uma camada a mais de triângulos nas zonas, que é necessário para obter uma malha sem buracos entre elas. Ao final, tem de ser feita a junção de todas as malhas geradas em uma só e, para isso, é preciso eliminar as redundâncias (triângulos repetidos entre duas zonas). Essa junção das malhas pode ser um processo complexo em determinados modelos e isso pode prejudicar o desempenho do algoritmo. A Figura 3.6 mostra o fluxograma da triangulação em paralelo.

O trabalho de (FREITAS et al., 2013) apresenta uma técnica bidimensional que recebe como entrada uma fronteira e utiliza uma *quadtree* para particionar e estimar a carga. As células folhas da *quadtree* de particionamento são divididas entre os processadores disponíveis, onde são geradas as malhas internas. Depois de gerar as malhas nos subdomínios iniciais, a fronteira é atualizada e as células da *quadtree* são deslocadas nos eixos cartesianos a fim de gerar mais malha. Esse processo de deslocamento e geração de malha é feito até que não seja mais possível gerar malha. Um processo mestre fica responsável por finalizar a geração da malha e

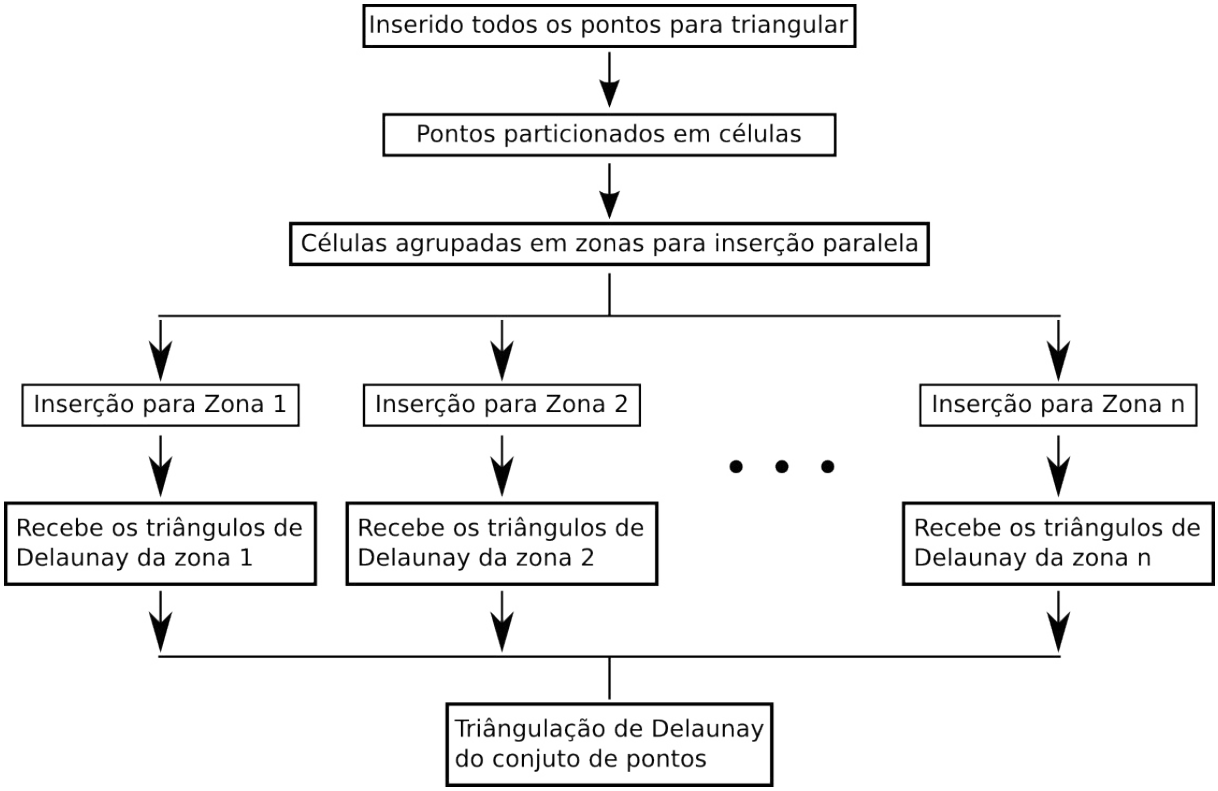


Figura 3.6: Fluxograma da triangulação em paralelo (LO, 2012).

fazer a melhoria na mesma. Este trabalho é classificado como contínuo *a posteriori*. A Figura 3.7 ilustra o passo de deslocamento da *quadtree* no eixo X.

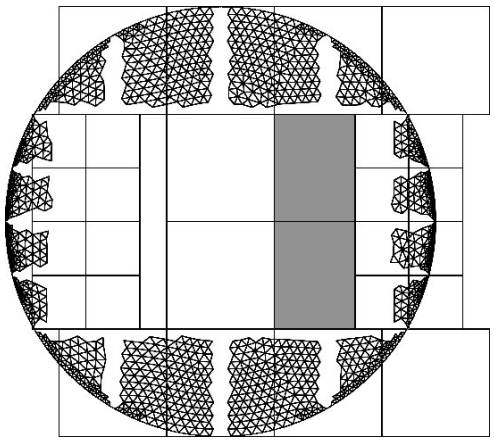
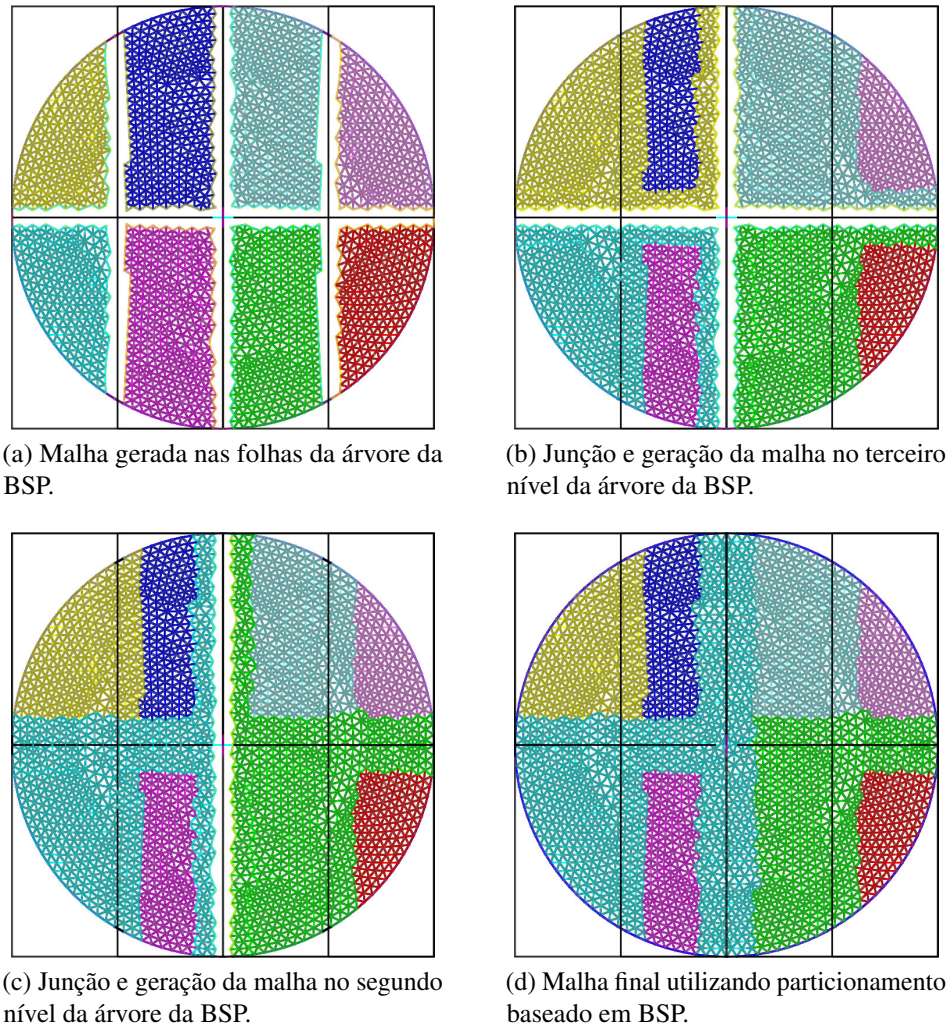


Figura 3.7: Malha gerada, espalhada entre os processos escravos, e as células da *quadtree* de decomposição deslocadas para a direção +X (FREITAS et al., 2013).

Em (FREITAS; CAVALCANTE-NETO; VIDAL, 2014), é apresentada uma evolução da técnica anterior que pode ser tanto bidimensional como tridimensional, por Avanço de Fronteira, com subdivisão baseada em BSP, que recebe como entrada uma superfície de faces triangulares ou uma lista de arestas, para o caso bidimensional. Inicialmente uma *octree* é construída para estimar a carga no domínio de acordo com o tamanho das faces da superfície. As células internas da *octree* têm o tamanho definido de acordo com a maior e a menor face

da superfície de entrada. Uma BSP é utilizada para particionar o domínio de tal forma que a quantidade de subdomínios criados seja igual à quantidade de processadores disponíveis.

Após o particionamento, cada subdomínio pertencente a uma folha da árvore BSP gera sua malha por avanço de fronteira até que não seja mais possível avançar (quando chega no plano criado pela BSP), como mostra a Figura 3.8a. Quando os dois filhos de um nó da BSP terminam de gerar a malha nos seus respectivos subdomínios, o nó pai fica encarregado de juntar as duas malhas e, se necessário, terminar a geração da malha no novo subdomínio criado pela junção dos dois filhos, como mostra as Figuras 3.8b, 3.8c e 3.8d.



(a) Malha gerada nas folhas da árvore da BSP.

(b) Junção e geração da malha no terceiro nível da árvore da BSP.

(c) Junção e geração da malha no segundo nível da árvore da BSP.

(d) Malha final utilizando particionamento baseado em BSP.

Figura 3.8: Passos da geração da malha no trabalho de (FREITAS; CAVALCANTE-NETO; VIDAL, 2014). Cada cor representa a malha gerada por um processador.

A grande vantagem dessa técnica está na utilização da BSP, que acaba permitindo a geração de subdomínios com cargas mais equilibradas e no ganho de velocidade ao se evitar os deslocamentos que aconteciam anteriormente. Por essa abordagem necessitar de uma sincronização e junção da malha em cada nível da BSP, há uma perda na velocidade apesar dessas junções serem feitas em paralelo por processos diferentes.

3.3 Decomposição Baseada no Eixo Mediano

O eixo mediano é uma maneira de descrever a forma de um objeto, e pode ser utilizado para garantir uma decomposição de domínio com separadores que formam bons ângulos com as bordas dos modelos.

Em (LINARDAKIS; CHRISOCHOIDES, 2006), é apresentada uma técnica bidimensional que utiliza a triangulação de Delaunay por divisão e conquista para um conjunto de pontos dados como entrada. Primeiramente, é feita uma triangulação utilizando apenas os pontos da borda, que é utilizada para a geração de um grafo ponderado, onde o peso de uma aresta é igual ao raio da circunferência circunscrita do triângulo que a contém. Em seguida é feita uma contração desse grafo, onde os vértices do grafo representam a área a ser triangularizada (futuros subdomínios), e as arestas representam a conexão entre essas áreas.

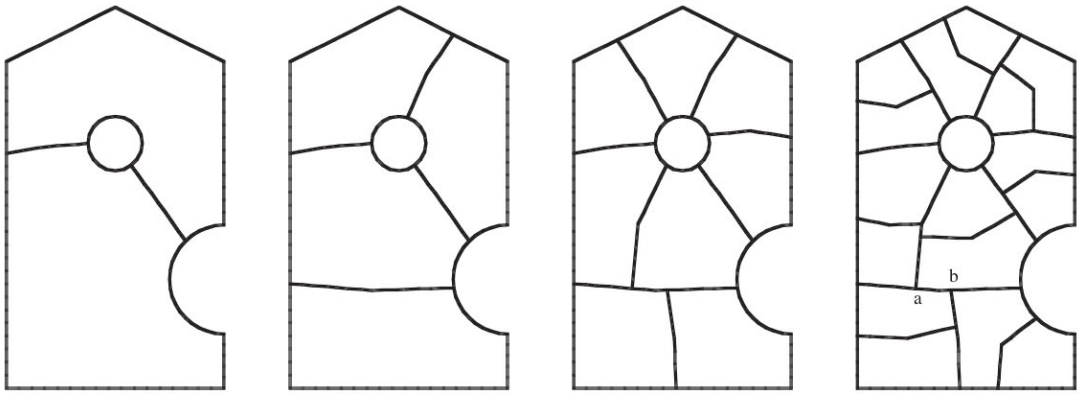


Figura 3.9: Partições em 2, 4, 8 e 16 (LINARDAKIS; CHRISOCHOIDES, 2006).

Através do grafo formado, os planos de corte são posicionados e os subdomínios formados. A Figura 3.9 mostra o resultado de decomposições para quantidades diferentes de subdomínios. Esse processo de subdivisão ocorre até que a quantidade de subdomínios criados seja suficientemente grande. Isso acontece para tentar melhorar o balanceamento de carga. Após isso, a geração da malha interna poderá ser realizada. A desvantagem dessa técnica é a ausência de uma estimativa de carga eficiente, precisando gerar vários subdomínios para melhorar o balanceamento de carga.

3.4 Decomposição Baseada em *Bounding Box*

Em (GLUT; JURCZYK, 2008), é apresentada uma técnica para malhas tridimensionais com uma abordagem baseada na decomposição geométrica onde a entrada é uma malha de superfície. Nesse trabalho são descritas duas técnicas baseadas na *bounding box* gerada a partir da entrada.

A seleção do separador do domínio deve garantir um custo de corte baixo, ou seja, encontrar e posicionar o plano de corte não pode ter um custo computacional alto. Além disso, deve garantir um bom balanceamento de carga e minimizar os elementos conectados por múltiplos subdomínios.

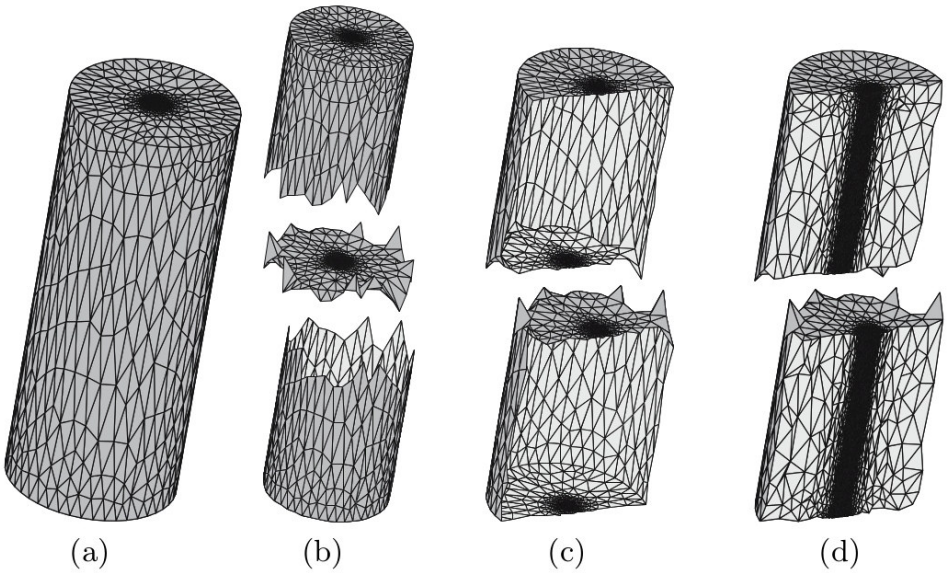


Figura 3.10: Passos da técnica baseada na malha de superfície. (a) malha de superfície; (b) corte; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008).

A primeira técnica é baseada na malha de superfície. Para o plano de corte ser criado, é preciso a localização do contorno da malha de superfície e do separador. O contorno é então projetado no separador usando uma função 2D de controle espacial baseada no tamanho das arestas (Figura 3.10).

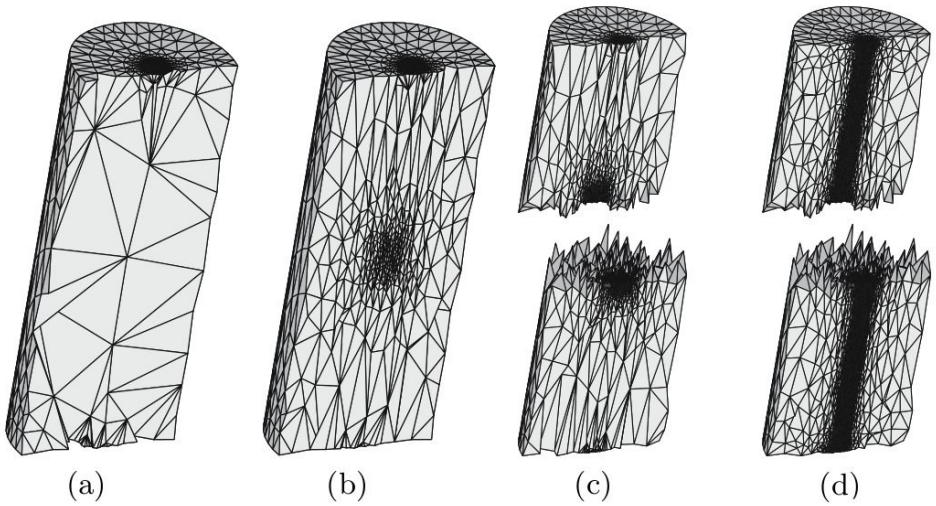


Figura 3.11: Passos da técnica baseada na malha volumétrica grosseira. (a) malha volumétrica grosseira; (b) refinamento da seção transversal; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008).

A segunda técnica se baseia numa malha volumétrica grosseira. Primeiramente, é feita a geração de uma malha 3D grosseira utilizando alguma função de controle espacial. O posicionamento do plano de corte é feito parecido com a técnica anterior, porém utilizando a malha volumétrica como função espacial (Figura 3.11).

Essa técnica depende muito da geometria da entrada já que são utilizadas informações da *bounding box* dessa entrada. Isso afeta diretamente a criação dos planos de corte, e por consequência, a malha gerada ao final.

3.5 Considerações Finais

As técnicas que foram discutidas neste capítulo apresentam bons resultados, mas o balanceamento de carga pode ser um problema em muitas delas, fazendo o desempenho do algoritmo paralelo cair. A forma com que os cortes são posicionados nessas técnicas dependem bastante do formato do objeto de entrada. Além disso, algumas técnicas necessitam de intervenção de um usuário, ou seja, o processo de geração da malha não é totalmente automático. Uma boa abordagem de balanceamento e de decomposição de domínio é essencial para algoritmos de subdivisão de domínios, caso contrário, o tempo para geração de malha pode ser prejudicado e a qualidade da malha ser afetada.

4 TÉCNICA PROPOSTA

Nesse capítulo é apresentada a técnica proposta de subdivisão de domínios bidimensionais para geração de malhas. Ela foi projetada para atender a alguns requisitos:

1. Respeitar a fronteira de entrada, discretizada em segmentos, sem efetuar refinamentos;
2. Produzir bons elementos, evitando-se elementos com proporções ruins;
3. Proporcionar boas transições entre as regiões muito refinadas, com muitos triângulos pequenos, e regiões grosseiras, com poucos triângulos grandes, da malha;
4. Manter a compatibilidade das fronteiras (segmentos de interface) dos subdomínios criados com as de seus vizinhos;
5. Abstrair a técnica de geração de malha, podendo-se combinar mais de uma técnica;
6. Abstrair o tipo de arquitetura de memória a ser usada (compartilhada ou distribuída); e
7. Tentar gerar malhas da forma mais eficiente possível em termos de tempo de processamento.

O primeiro requisito é muito importante em muitos problemas, tais como aqueles encontrados nas simulações em que o domínio contém regiões com diferentes materiais e/ou buracos. Nesses problemas, geralmente é desejável que a malha esteja em conformidade com uma discretização de contorno já existente nessas regiões.

Quanto ao segundo requisito, embora a técnica proposta dependa de uma *quadtree* refinada para a geração de elementos de boa qualidade, é feita uma série de testes e refinamentos a fim de garantir que os elementos gerados serão de boa qualidade.

Com relação ao terceiro requisito, em muitas aplicações, a diferença de tamanho entre os elementos em uma região refinada e aqueles de uma região grosseira é maior do que duas ordens de magnitude. Assim, para se obter uma boa transição deve-se evitar uma diferença de magnitude maior do que dois.

De acordo com o quarto requisito, é necessário manter a conformidade da malha assim como facilitar a junção da malha de cada partição para obter a malha final. Para isso acontecer é necessário que os mesmos segmentos gerados em um subdomínio tenham os seus simétricos gerados nos seus vizinhos, isso garante que a malha que será gerada nos dois lados serão compatíveis quando for feita a junção das malhas geradas em cada subdomínio.

O quinto requisito diz que a técnica proposta abstrai o tipo de malha e o gerador que será utilizado, contanto que respeite todos os pré-requisitos aqui citados. Isso permite que possam ser combinadas diferentes técnicas de geração de malha, como as de Delaunay e Avanço de Fronteira, por exemplo.

Para atingir o sexto requisito, a técnica proposta pode utilizar uma arquitetura de computador de memória distribuída ou compartilhada, com uma capacidade de balanceamento de carga elevada. Atualmente esses tipos de arquitetura têm sido amplamente usados com o barateamento dos computadores e *clusters*.

O último requisito é também muito importante, já que a velocidade e a qualidade da malha são atualmente um dos principais objetivos de estudo na área da geometria computacional.

4.1 Descrição Geral

Na técnica proposta a entrada é uma lista de segmentos que definem a fronteira de um ou mais objetos, que podem conter ou não buracos. Uma *quadtree* será refinada de acordo com o tamanho das arestas dadas como entrada, levando em consideração que o tamanho das células internas desta *quadtree* não pode ser maior que o maior segmento da entrada. Esta *quadtree* é chamada de *quadtree* de densidade ou de carga.

Com a *quadtree* de densidade devidamente criada, a estimativa de carga é realizada. Este processo de estimativa é feito baseado na quantidade de células que são internas ao domínio, tal como descrito na próxima se. O próximo passo é a decomposição da entrada em diversas partes. Para realizar a decomposição do domínio, uma estrutura de dados que gere regiões retangulares deve ser utilizada para que a técnica de geração de subdomínios proposta nesse trabalho funcione perfeitamente. Isso será mostrado na seção 4.3.

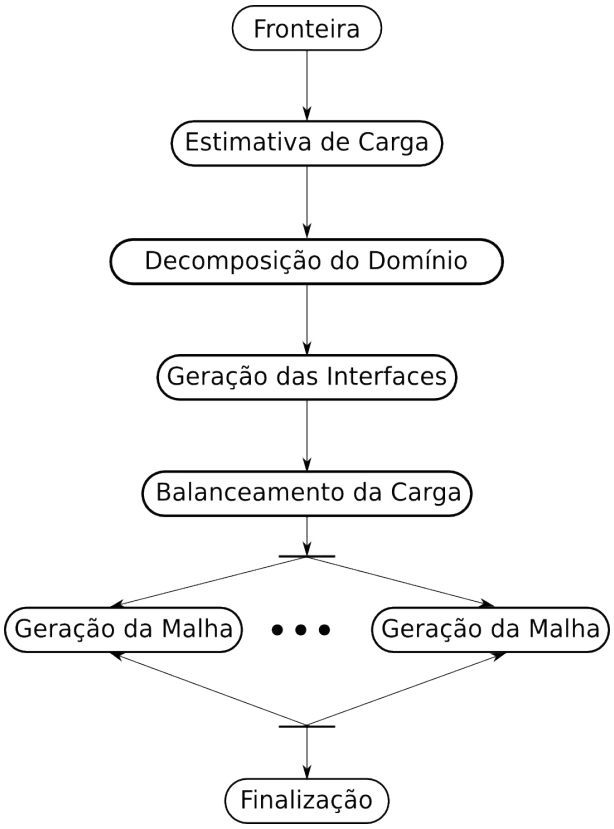


Figura 4.1: Visão geral da técnica paralela.

O algoritmo de geração das malhas de interfaces dos subdomínios recebe o resultado da decomposição do domínio e cria os novos subdomínios, tendo cada um deles sua nova fronteira compatível com as de seus vizinhos.

Após a criação de todos subdomínios, o balanceamento de carga é feito caso o número de processos criados seja maior que a quantidade de processadores disponíveis para executar a geração da malha. A malha é então gerada em cada um dos subdomínios e ao final é feita uma junção dos pedaços da malha e realizado algumas melhorias. A Figura 4.1 mostra o fluxograma de como a técnica procede.

4.2 Estimativa de Carga

Em Computação de Alto Desempenho (CAD), a carga é uma medida da quantidade de trabalho a ser realizado em um subdomínio ou um conjunto de subdomínios. Em problemas de geração de malha, a carga está relacionada com o número de elementos que serão gerados em cada subdomínio. Portanto, a carga está relacionada com as seguintes questões, que devem ser levadas em conta em sua estimativa (Figuras 4.2 e 4.3):

- O nível de discretização da malha, geralmente especificado pelo usuário ou por outro software, através de parâmetros de entrada. Quanto maior a discretização da malha, maior será a carga (Figura 4.2).
- Em regiões com o mesmo nível de discretização, uma região maior gera mais elementos do que uma região menor. Assim, quanto maior for a região, maior será a carga (esquerda da Figura 4.3).
- Em domínios com diferentes níveis de discretização, regiões de mesmo tamanho podem gerar um número de elementos diferente, dependendo do nível de refinamento de cada região. Assim, quanto mais refinada for a malha na região, maior será a carga (direita da Figura 4.3).

A malha da direita da Figura 4.2 sugere uma carga maior que a malha da esquerda da mesma figura. A mesma discretização de borda foi utilizada nos dois casos. Na Figura 4.3, a carga associada às regiões cercadas por linhas contínuas é maior que a carga associada às regiões cercadas por linhas tracejadas.

4.2.1 Construção da *Quadtree* de Estimativa de Carga

A construção da *quadtree* de estimativa de carga ou *quadtree* de densidade é utilizada tanto para estimar a carga como para auxiliar a geração das malhas de interface. Sua criação é iniciada com a construção de um quadrado de tamanho mínimo que engloba toda a fronteira dada como entrada (Figura 4.4a); este quadrado é a célula-raiz da *quadtree*. Esta *quadtree* será subdividida até que todos os pontos médios das arestas da entrada estejam dentro de uma célula

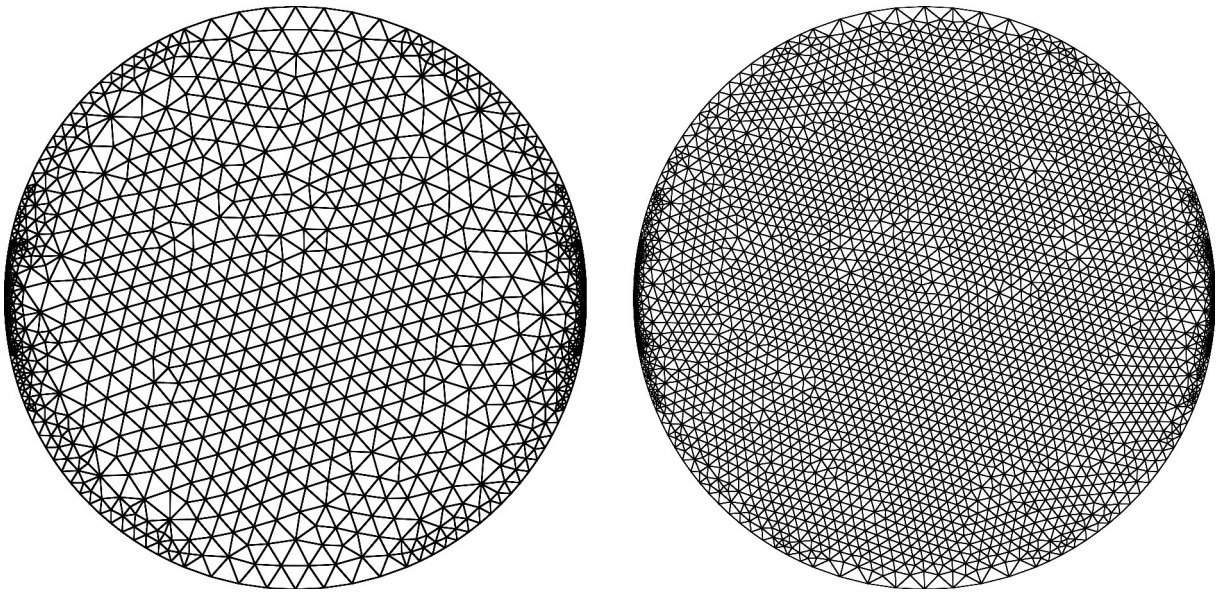


Figura 4.2: Estimativa de carga: refinamento menor (esquerda) e maior (direita).

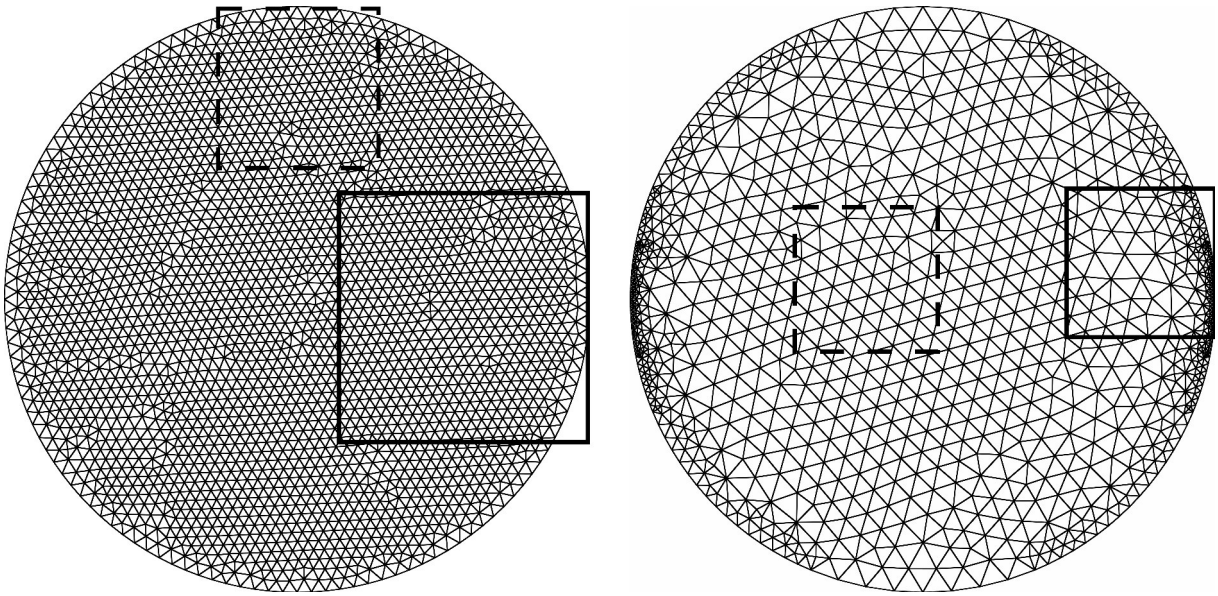
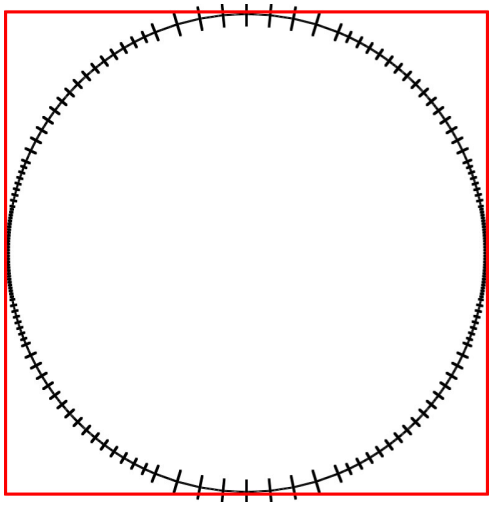


Figura 4.3: Estimativa de carga: malhas uniforme (esquerda) e não-uniforme (direita).

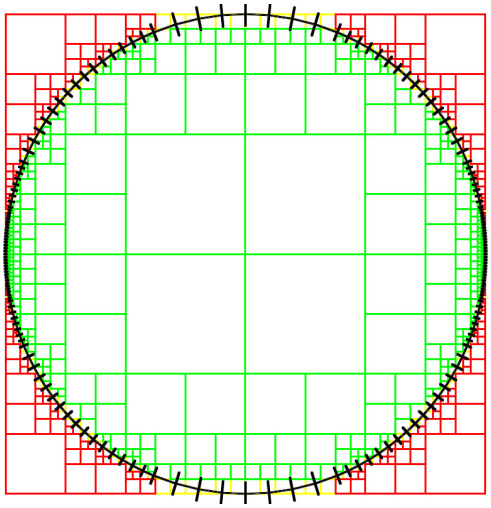
da *quadtree* de densidade (Figura 4.4b). O tamanho destas células tem de ser menor que o comprimento da aresta à qual o ponto médio pertence, multiplicada por uma constante. Essa constante tem valor $\sqrt{3}/2$, que equivale a 0,85, que é uma aproximação da altura de um triângulo equilátero.

Dois refinamentos são então aplicados. O primeiro é para garantir que todas as células da *quadtree* de densidade não sejam maiores que a maior célula que contém um ponto médio da borda de entrada (Figura 4.4c). Com esse refinamento é garantido um tamanho máximo para as células de acordo com as arestas da fronteira. Assim, os elementos que serão gerados no interior do domínio terão proporções iguais aos da borda.

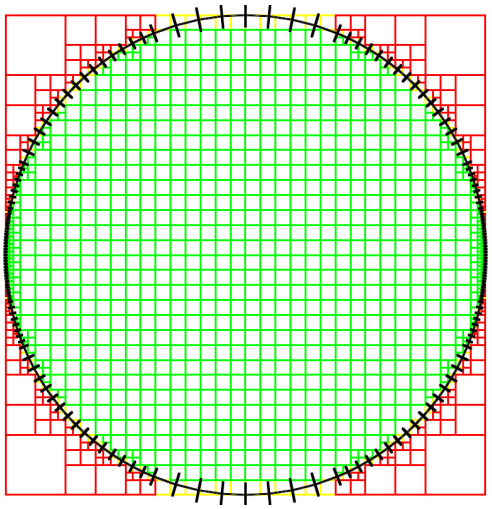
O segundo refinamento, também conhecido na literatura como refinamento 2:1, é



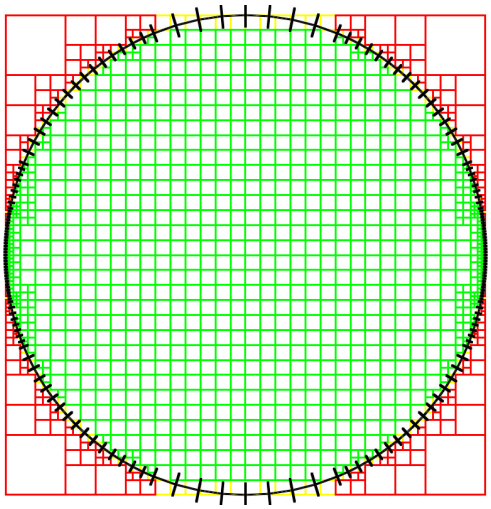
(a) Fronteira de entrada juntamente com sua caixa delimitadora (em vermelho).



(b) *Quadtree* de densidade inicialmente gerada.



(c) Primeiro refinamento na *quadtree* de densidade.



(d) Segundo refinamento na *quadtree* de densidade.

Figura 4.4: Passos da geração da *quadtree* de densidade.

para garantir que a diferença de níveis da *quadtree* não seja maior que 2 para células vizinhas (que compartilham um lado), como mostrado na Figura 4.4d. Este refinamento é feito para garantir uma transição suave entre elementos grandes e pequenos. Mais detalhes de como essa *quadtree* é construída podem ser encontrados em (MIRANDA; CAVALCANTE-NETO; MARTHA, 1999) (a construção de sua versão tridimensional, uma *octree*, pode ser encontrada em (CAVALCANTE-NETO et al., 2001)).

4.2.2 Classificação das Células

Uma classificação das células da *quadtree* de densidade contribui para otimizar buscas e para evitar o desperdício de memória. Pode-se classificar uma célula como interna (caso a célula esteja totalmente interna ao domínio), externa (caso a célula esteja totalmente externa ao

domínio) ou sobre a fronteira do domínio (caso a célula faça interseção com alguma aresta do domínio). O algoritmo de classificação é mostrado em (FREITAS, 2010). A Figura 4.5 mostra as células de uma *quadtree* de densidade classificadas para um dado domínio circular, onde as cores atribuídas às células indicam a sua classificação (células verdes - dentro do domínio, células vermelhas - fora do domínio, células amarelas - sobre a fronteira).

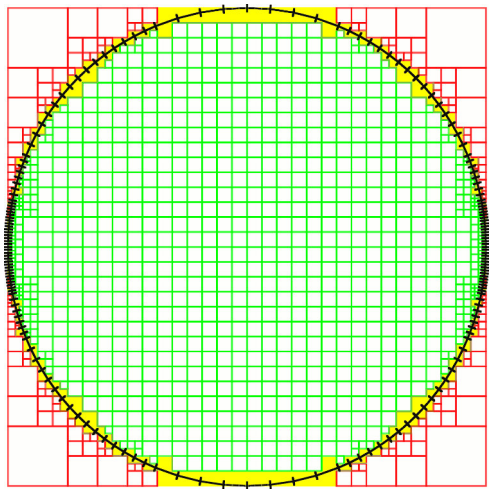


Figura 4.5: *Quadtree* de densidade com as células devidamente classificadas (células verdes - dentro do domínio, células vermelhas - fora do domínio, células amarelas - sobre a fronteira).

4.2.3 Cálculo da Carga

A *quadtree* de densidade é utilizada para fazer uma estimativa da carga neste trabalho. Com a sua construção é possível ter noção do tamanho dos elementos pertencentes ao domínio do objeto. Assim, uma ideia geral sobre a malha desejada é conhecida desde o início. A classificação das células folhas da *quadtree* de densidade é utilizada para selecionar apenas as células que não estão fora do domínio de entrada, ou seja, as células que estão no interior do domínio e as que interceptam a borda (células verdes e amarelas na Figura 4.5). A quantidade dessas folhas não externas é considerada como a carga total para este domínio.

4.3 Decomposição do Domínio

A técnica proposta neste trabalho tem apenas um pré-requisito quanto à estrutura de decomposição espacial para particionar o domínio dado como entrada. A estrutura de decomposição deve gerar regiões que sejam paralelas às células da *quadtree* de densidade. Estruturas baseadas em árvore como *quadtree*, BSP e *Kd-tree* geram regiões que possibilitam a perfeita execução do algoritmo de geração dos subdomínios.

Vale lembrar que para um bom particionamento é preciso uma boa estimativa de carga e para se obter uma malha de boa qualidade depende-se de um bom particionamento do domínio. Para gerar exemplos e até mesmo para comparar os resultados dos diferentes tipos de

estruturas de dados, esta seção descreve como é feita a criação de uma *quadtree* e de uma BSP para decompor a entrada.

4.3.1 Decomposição por *Quadtree*

O domínio é decomposto pela chamada *quadtree* de decomposição, que é guiada pela *quadtree* de estimativa de carga descrita na Seção 4.2. Essa *quadtree* de decomposição pode ser construída de diversas formas.

- Subdivisão baseada na carga. Cada célula folha não pode ter uma carga maior que a carga média, calculada como a carga total dividida pela quantidade de processadores disponíveis;
- Subdivisão baseada na diferença mínima de carga total de cada um dos processadores; e
- Subdivisão baseada na quantidade de processadores disponíveis.

Vale lembrar que, na segunda estratégia, é necessário que seja feita uma boa escolha para o valor da diferença entre as cargas totais dos processadores. Um valor muito alto pode acabar gerando um número excessivo de partições.

Entre as melhores maneiras de criar a *quadtree* de decomposição, a que será detalhada é aquela que faz a carga associada com cada uma das folhas da *quadtree* ser menor que uma carga máxima pré-definida. Este limiar de carga é uma função da carga total e do número de processadores escravos disponíveis.

Se L representa a carga total, idealmente, N subdomínios teriam carga L/N cada um. É desejável que o número de subdomínios gerados seja, pelo menos, igual ao número de processos escravos P. Assim, a carga ideal para cada processador seria de L/P.

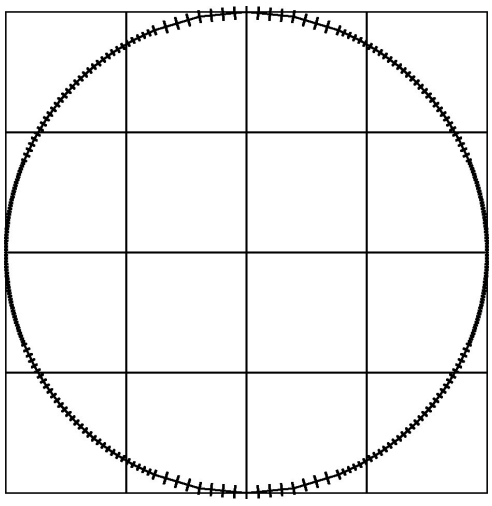


Figura 4.6: Exemplo de uma *quadtree* de decomposição.

A *quadtree* de decomposição se inicia com sua raiz definida como um quadrado envolvendo o domínio, e com sua carga definida como a carga total estimada. Se a carga desta

célula for maior que o limiar de carga, a célula é subdividida em quatro células de mesmo tamanho.

A carga associada com uma dada célula da *quadtree* de decomposição é estimada como o número de células-folha da *quadtree* de estimativa de carga que não estiverem fora do domínio e que estiverem dentro da célula da *quadtree* de decomposição. A verificação da carga máxima é feita recursivamente, para cada célula-folha da *quadtree* de decomposição e, sempre que a carga de uma célula exceder o limiar de carga, esta célula é subdividida em quatro (Figura 4.6). Cada folha da *quadtree* de decomposição que cruzar a fronteira ou estiver dentro do domínio é considerada um subdomínio.

Após a geração da quadtree de decomposição, os segmentos da fronteira de entrada são divididos entre os subdomínios existentes. Assim, se os dois vértices de um segmento estiverem dentro de um subdomínio, o segmento é dito pertencer exclusivamente a esse subdomínio. No entanto, se o segmento atravessar dois ou mais subdomínios, ou tocar os limites de alguns subdomínios, esse segmento é dito pertencer a todos os subdomínios tocados ou cruzados.

4.3.2 Decomposição por BSP

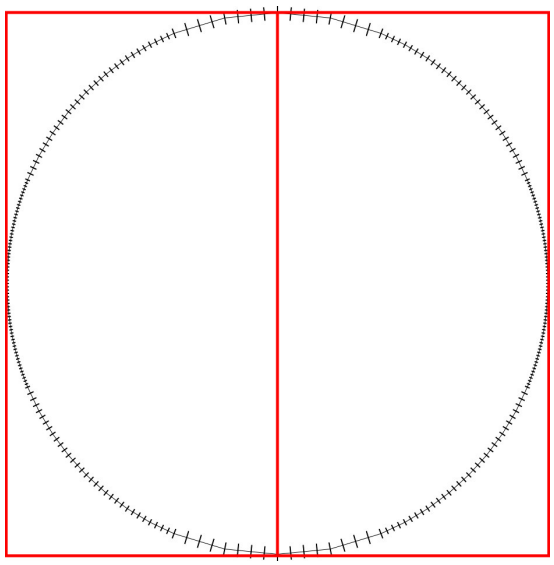
O domínio também pode ser particionado por uma BSP chamada de BSP de decomposição, tendo a sua criação guiada pela *quadtree* de estimativa de carga descrita na Seção 4.2. A utilização desta estrutura foi baseada no trabalho de (FREITAS; CAVALCANTE-NETO; VIDAL, 2014), e mais detalhes de construção e implementação podem ser encontrados nele. A BSP de decomposição pode ser gerada de forma parecida com a *quadtree* de estimativa de carga, ou seja:

- Subdivisão baseada na carga. Cada célula folha não pode ter uma carga maior que a carga média;
- Subdivisão baseada na diferença mínima de carga total de cada um dos processadores; e
- Subdivisão baseada na quantidade de processadores disponíveis.

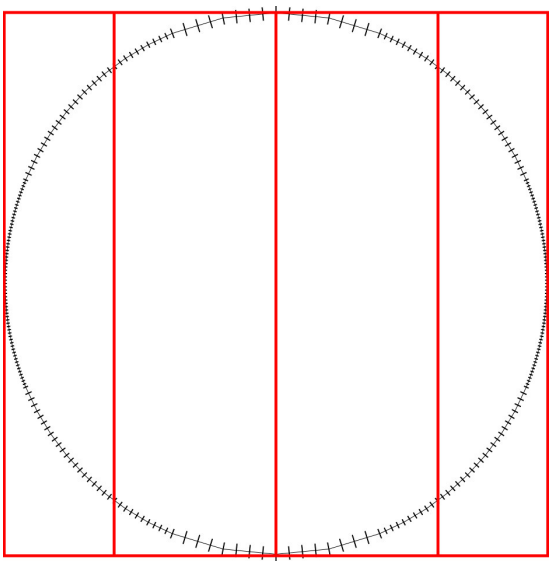
As principais diferenças entre uma BSP e uma *quadtree* são: a quantidade de subdivisões realizadas; e a liberdade que a BSP tem de subdividir o domínio em regiões de diferentes tamanhos. A possibilidade de gerar a quantidade de regiões que sejam necessárias com diferentes tamanhos torna o particionamento muito mais preciso que a *quadtree*. A Figura 4.7 mostra uma BSP de particionamento feita para dez processadores, ou seja, dez subdomínios foram criados.

Para gerar a BSP de decomposição a forma mais eficiente é a que faz a carga associada com cada uma das folhas ser menor que uma carga máxima pré-definida L/P. Com a BSP, é possível obter $N = P$ com uma carga de L/P para cada P.

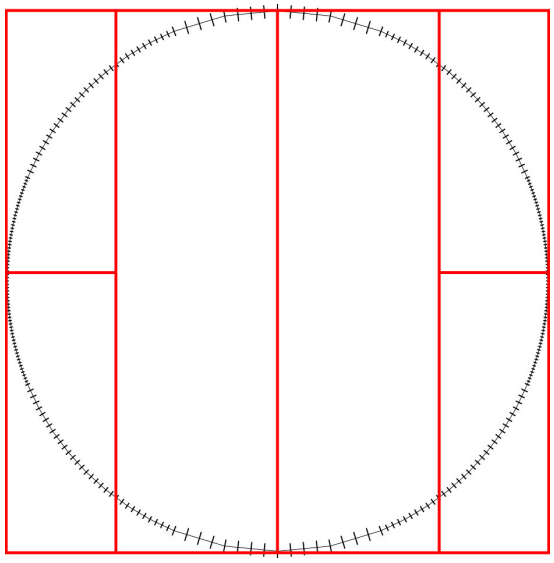
Inicia-se a BSP de decomposição com sua raiz definida como um quadrado envolvendo o domínio. A subdivisão é feita posicionando a partição no centro geométrico dessa célula, no eixo X. Se essa partição obtiver cargas iguais para os dois filhos, essa partição será a melhor



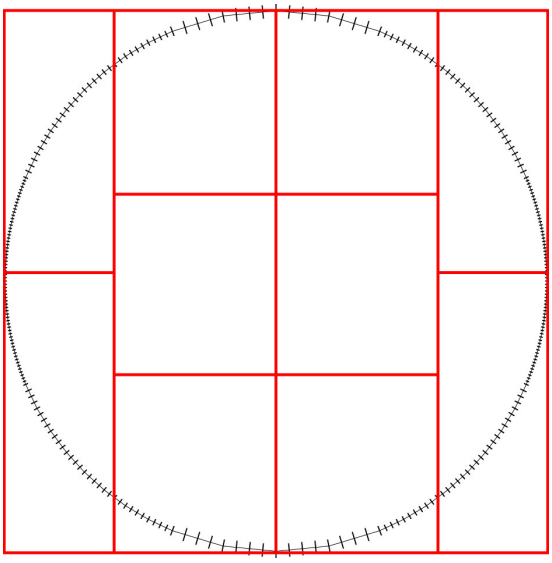
(a) Primeiro corte criado com pesos 5 para cada lado.



(b) Segundo corte criado com pesos 2 e 3 para cada lado.



(c) Terceiro corte criado gerando dois subdomínios.



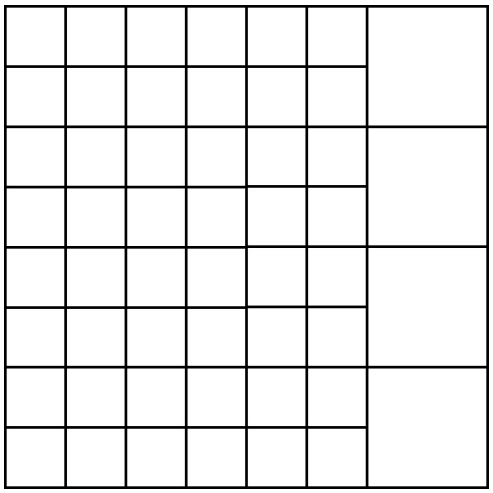
(d) Último corte criado finalizando os 10 subdomínios.

Figura 4.7: Exemplo da criação de uma BSP para 10 processadores.

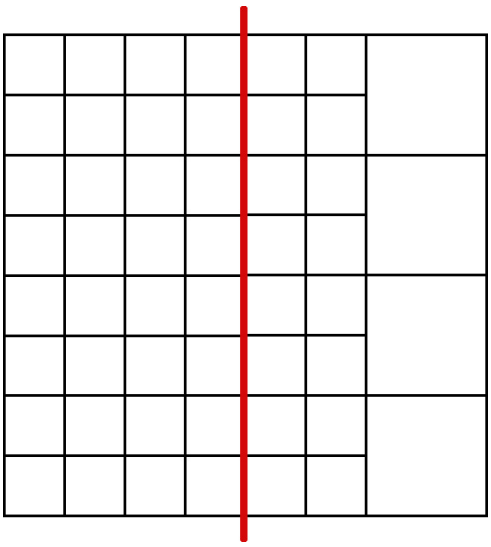
para o eixo X. Caso contrário, seleciona-se a célula mais pesada e reposiciona-se o plano de partição para a metade dessa célula, no mesmo eixo. Esse procedimento é feito recursivamente até que as cargas dos dois subdomínios sejam iguais ou, quando atinge-se uma célula-folha da *quadtree*, ou seja, é impossível subdividir mais ao meio.

Esse procedimento é realizado nos dois eixos e ao final é selecionado o particionamento no eixo que melhor divide a carga nas duas novas células criadas. A Figura 4.8 mostra o passo a passo da seleção da melhor subdivisão para o eixo X. Com essa subdivisão no eixo X, a diferença alcançada é de 4 (Figura 4.8d); porém, fazendo a decomposição no eixo Y a diferença seria de 0. O particionamento selecionado vai ser o que obtiver a menor diferença e então será

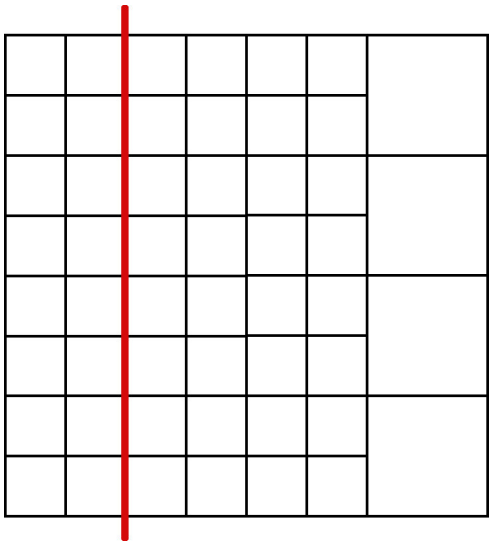
selecionada a subdivisão feita no eixo Y.



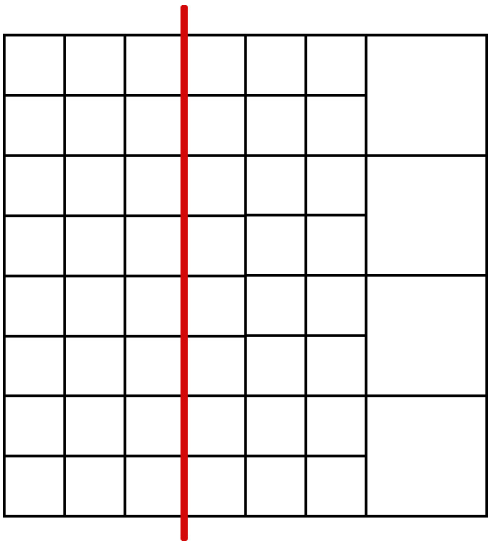
(a) *Quadtree* de estimativa carga. Carga total: 52. Carga ideal de cada subdomínio: 26.



(b) Posição inicial do plano de partição (em vermelho). Cargas dos subdomínios: 32 e 20. Diferença: 12.



(c) Deslocamento do plano de partição para o próximo nível da esquerda. Cargas dos subdomínios: 16 e 36. Diferença: 20.

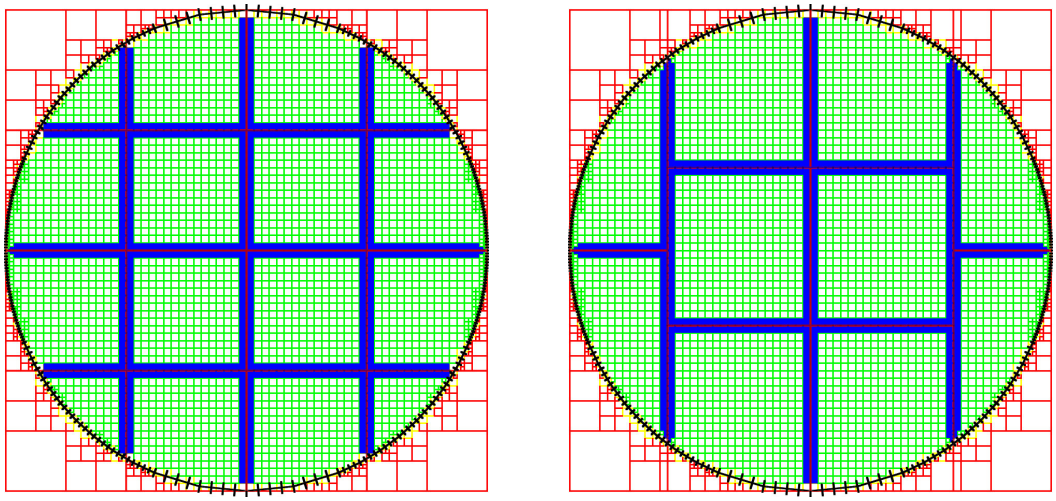


(d) Deslocamento do plano de partição para o próximo nível da direita. Cargas dos subdomínios: 24 e 28. Diferença: 4.

Figura 4.8: Passos de uma decomposição de domínio por BSP no eixo X para dois processadores.

4.4 Geração das Interfaces

Na geração das interfaces dos subdomínios, é necessário ter a *quadtree* de densidade e a estrutura de particionamento devidamente criadas. Qualquer estrutura de decomposição espacial que gere regiões paralelas aos eixos pode ser utilizada nessa técnica de geração de subdomínios. A Figura 4.9 mostra um exemplo da junção da *quadtree* de densidade com uma *quadtree* de particionamento (Figura 4.9a) e com uma BSP (Figura 4.9b).

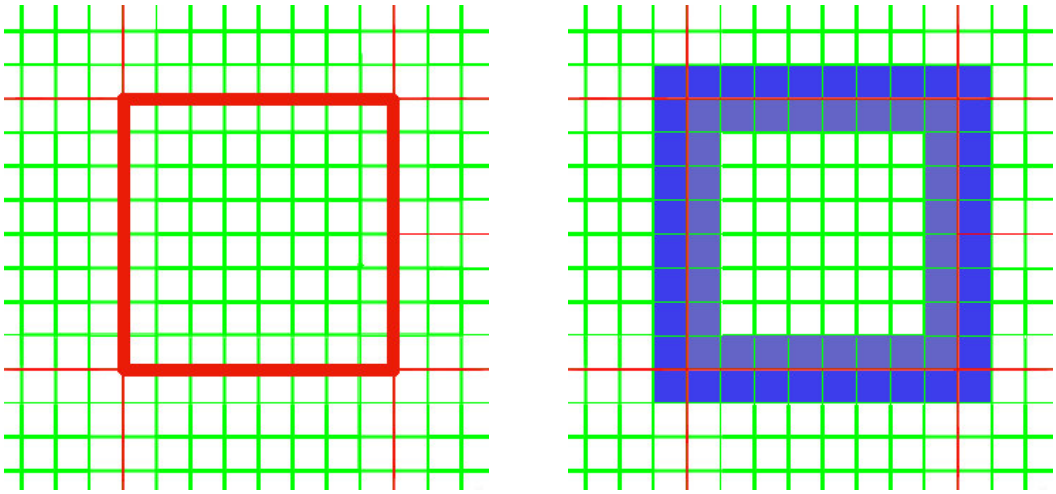


(a) *Quadtree* de decomposição junto com a *quadtree* de densidade.

(b) BSP de decomposição junto com a *quadtree* de densidade.

Figura 4.9: Células da *quadtree* de densidade em azul serão utilizadas para guiar a criação da borda dos novos subdomínios.

Para cada célula folha da estrutura de particionamento é criada uma caixa delimitadora. Esta caixa é utilizada para fazer interseção com as células da *quadtree* de densidade (Figura 4.10a). Como resultado dessa interseção, é obtido um conjunto de células da *quadtree* de densidade que cruzam ou apenas tangenciam a partição. Essas células guiarão a criação das fronteiras dos seus respectivos subdomínios (células em azul na Figura 4.10b). A *quadtree* de densidade possui informações de tamanho que são baseadas nas arestas da borda, essas informações ajudam na criação da malha de interface compatível com a discretização do domínio.



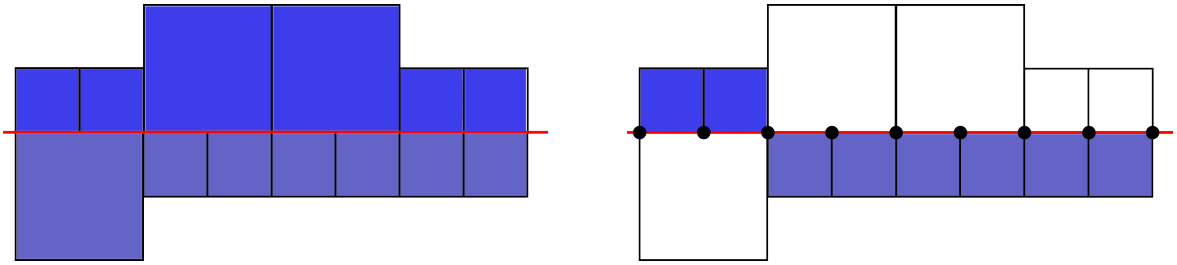
(a) Caixa delimitadora de uma célula da estrutura de particionamento selecionada (células verdes - *quadtree* de densidade, células vermelhas - estrutura de particionamento).

(b) Interseção da caixa delimitadora da estrutura de particionamento com a *quadtree* de densidade.

Figura 4.10: Processo de seleção das células que guiarão a criação das arestas de interface.

Células que estão totalmente fora da fronteira do domínio, ou seja, que não estão

dentro e nem sobre a fronteira do domínio, mas que também fazem interseção com a estrutura de particionamento, não devem fazer parte desse conjunto. Como essas células estão totalmente fora do domínio, não haverá criação de arestas nessas regiões; logo, elas devem ser retiradas desse conjunto.



(a) Exemplo onde a *quadtree* de densidade possui células de tamanhos diferentes.

(b) Células em azul foram as selecionadas para geração das arestas de interface.

Figura 4.11: Processo de criação das arestas de interface.

Para facilitar a criação das arestas de interface, é dividido o conjunto das células que fazem interseção com a caixa delimitadora da estrutura de particionamento do subdomínio em quatro grupos, um para cada lado da caixa delimitadora (esquerdo, direito, superior e inferior), e para cada grupo é feita a classificação de suas células como interna ou externa ao subdomínio. Essa classificação em grupos permite fazer a criação dos segmentos de interface em paralelo. Na Figura 4.10b as célula em azul escuro são externas ao subdomínio e as azul claro são internas.

Em seguida, as células em cada um desses quatro grupos são ordenadas de acordo com o seu tamanho. Essa ordenação ajuda na criação das arestas da interface, de tal modo que as arestas criadas sejam sempre as de menor tamanho possível, de acordo com as células selecionadas. A Figura 4.11 mostra um exemplo de como o algoritmo de criação das arestas se comporta.

A construção das arestas de cada fronteira dos subdomínios se dá pela verificação desses quatro grupos, tomando-se sempre as células com menor comprimento dentre as que são internas ou externas, e quando tiver apenas células internas ao subdomínio, a criação utilizará somente essas células.

Algumas partições estarão sobre a fronteira do domínio e algumas de suas arestas terão que se conectar à fronteira. Para fazer essa ligação, não se utilizam as células da *quadtree* de densidade, pois como elas estão sobre a fronteira, as arestas que seriam criadas ultrapassariam a fronteira do domínio. Por isso, na criação dos conjuntos de células que interceptam a caixa delimitadora dos subdomínios, são eliminadas as células que cruzam ou apenas tangenciam a fronteira do domínio. A ligação das subfronteiras com a borda do domínio é feita por uma busca do ponto do domínio que está mais próximo do ponto da última aresta criada em cada lado da partição que tiver interseção com a borda do domínio. Ao final desse processo cada partição terá sua fronteira construída e estará pronta para geração da malha. A Figura 4.12 mostra um exemplo de geração das arestas dos subdomínios usando uma *quadtree* como estrutura de particionamento.

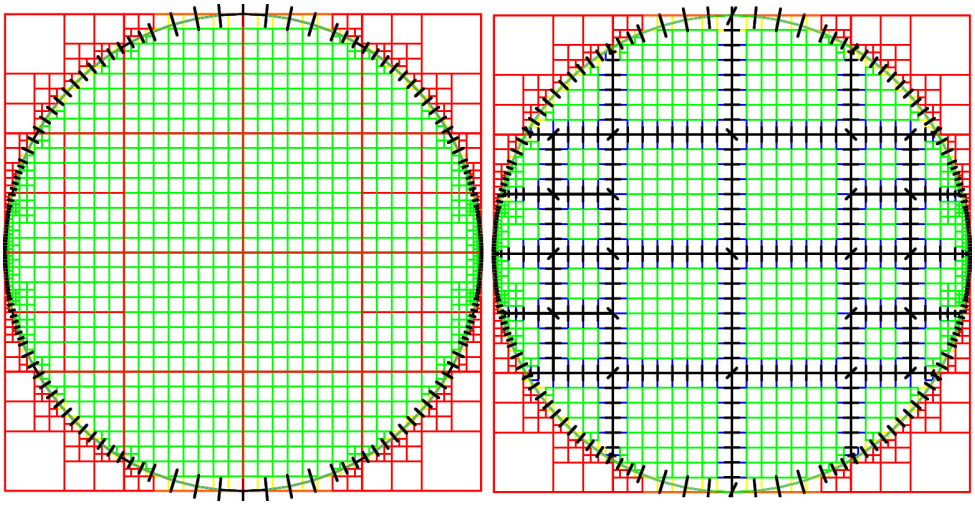


Figura 4.12: *Quadtree* de carga juntamente com a *quadtree* de partição (à esquerda) e as fronteiras dos subdomínios criadas (à direita).

4.4.1 Teste de Proximidade

Um dos problemas de se utilizar estruturas que geram regiões paralelas aos eixos para particionar domínios é a possibilidade das partições estarem posicionadas em regiões onde os elementos a serem gerados sejam de má qualidade ou até mesmo em posições em que não é possível gerar elementos. A Figura 4.13 ilustra o caso onde a fronteira dada como entrada e a célula da partição se tornam tão próximas que acabam tendo uma região onde praticamente se tangenciam.

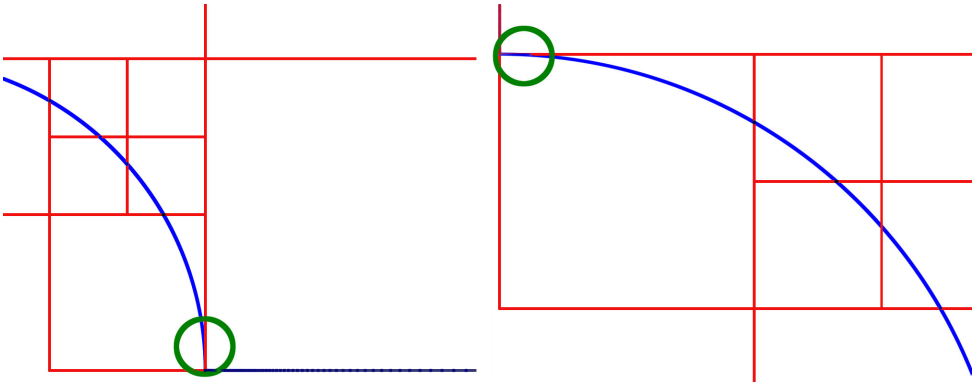


Figura 4.13: Casos onde a fronteira de entrada e a partição têm uma região que se tangenciam.

O simples fato de se buscar o vértice da fronteira mais próximo não garante a qualidade do subdomínio criado, pois, dependendo do refinamento da entrada e do posicionamento da caixa da *quadtree* de partição, o vértice mais próximo pode ter uma posição ruim para gerar malha. Quanto mais refinada for a entrada, maiores as chances de acontecer casos como o da Figura 4.14, onde há casos onde células muito próximas da fronteira que serão utilizadas na criação dos subdomínios. Se casos como esses não forem tratados, o algoritmo de geração de malha possivelmente falhará ou gerará elementos de baixa qualidade.

Para tratar os casos citados, foram implementados testes de proximidade que utilizam

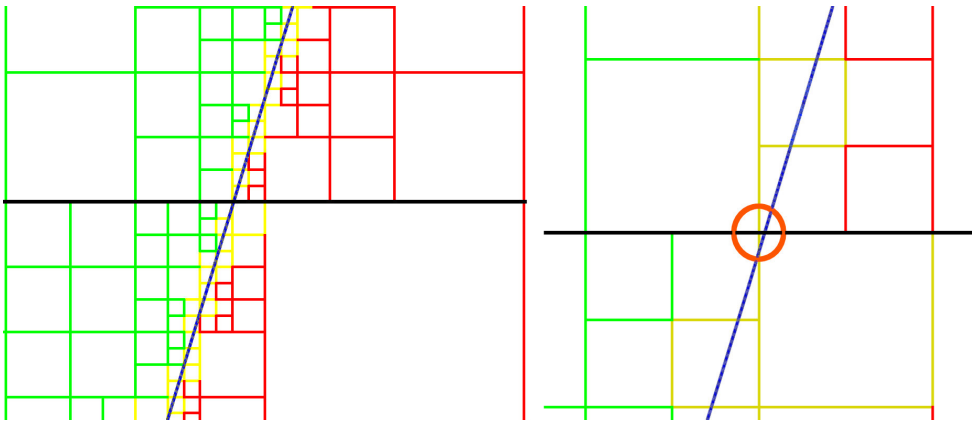


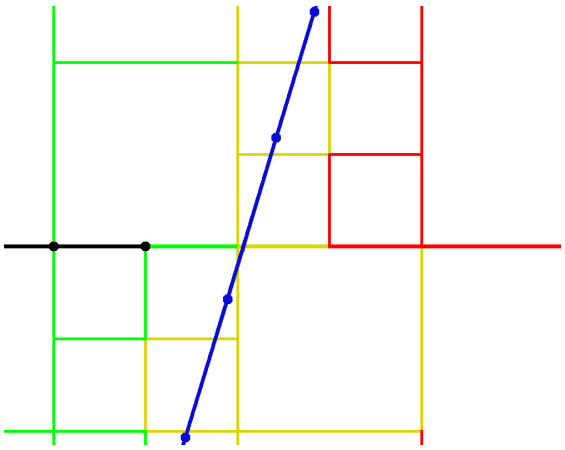
Figura 4.14: Exemplo onde uma célula da *quadtree* de densidade está muito perto da fronteira e possivelmente os elementos ali gerados serão ruins. À direita é mostrado um *zoom* da imagem da esquerda.

as informações das células da *quadtree* de densidade. Com esses testes a qualidade da malha melhora e garante a perfeita execução do algoritmo de geração de malha.

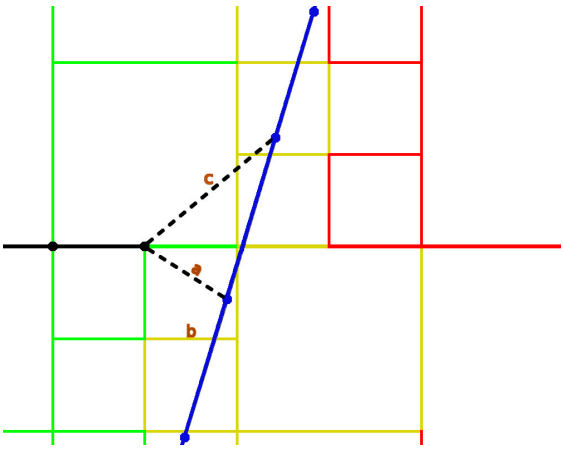
O teste de proximidade é realizado quando a partição que está sendo considerada tem alguma célula da *quadtree* de densidade que intercepta a fronteira de entrada em algum dos lados (fig. 4.15a). Se isso ocorrer, será feita uma verificação se a célula da *quadtree* de densidade que está sendo testada é classificada como **sobre** a fronteira do domínio. Em caso afirmativo, será realizada uma busca pelo vértice mais próximo da fronteira e calculadas as suas distâncias para os dois vértices mais próximos, tomando-se a menor delas (Figura 4.15b, onde as distâncias são a e c e a menor é a). Se essa distância for menor que o tamanho da célula da *quadtree* que intercepta a fronteira, será gerada a aresta que liga os dois vértices (Figura 4.15c, onde distância $a < b$). Caso contrário, será criado um novo vértice que seria normalmente considerado de acordo com a célula da *quadtree* de carga.

Esse teste é baseado na ideia de que para um bom elemento ser gerado ele precisa de uma área mínima. Neste trabalho é assumido que esta distância mínima deve ser o tamanho da célula da *quadtree* de densidade. Com estes testes evita-se a criação de arestas com ângulos muito pequenos (fig. 4.15d) e melhora-se a qualidade dos elementos que serão gerados.

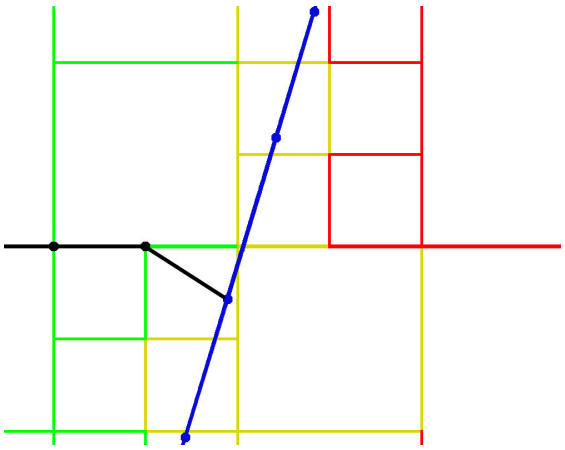
Esse mesmo teste se aplica a outros casos; um deles é quando a célula da partição tem uma região da sua célula tangente à fronteira de entrada, como mostrado na esquerda da Figura 4.16b. Neste caso o teste de proximidade será efetuado várias vezes de tal forma que a medida que as arestas vão entrando no teste de proximidade, a região que está muito próxima da borda é transferida para a partição vizinha. Como consequência disso, parte da malha que seria gerada por um processador, será agora tratada pelo vizinho como mostrado na direita da Figura 4.17.



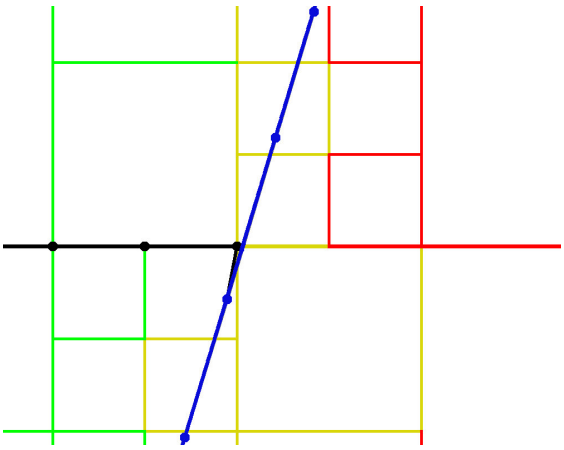
(a) Caso em que o tratamento de proximidade é realizado.



(b) Busca pelo vértice da fronteira mais próximo ao último vértice criado. É feita a comparação dessa distância com o tamanho da célula que está sobre a fronteira.



(c) Aresta criada com o teste de proximidade.



(d) Aresta que seria criada caso o teste de proximidade não fosse realizado.

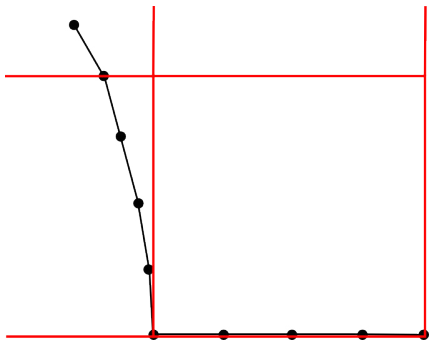
Figura 4.15: Possíveis casos no teste de proximidade.

4.4.2 Tratamento de Buracos

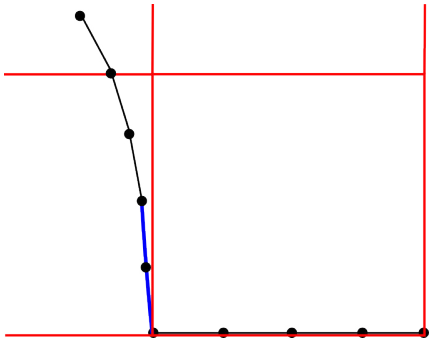
Alguns modelos possuem buracos em seus domínios, e, nesses casos, é preciso realizar um teste simples para evitar que sejam criadas arestas que atravessem a fronteira. Esse tratamento é realizado apenas quando uma célula da *quadtree* de partição intercepta algum buraco, como mostra a Figura 4.18.

Quando se tem um buraco que intercepta a partição que está sendo criada, existirão células classificadas como fora da partição entre as que estão classificadas como dentro ou que cruzam a fronteira. O algoritmo tentará gerar uma aresta entre duas células classificadas como internas, mas essa aresta irá cruzar células classificadas como externas ao domínio.

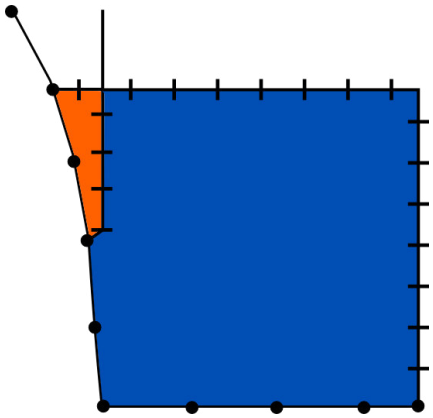
Quando é detectada uma possível colisão de uma aresta com células externas, serão criadas duas arestas em vez de uma. A primeira liga o último vértice criado ao vértice mais próximo da primeira interseção do buraco com a *quadtree* (Figura 4.19c), e a segunda liga o vértice mais próximo da segunda interseção com o buraco a um vértice que será criado de acordo



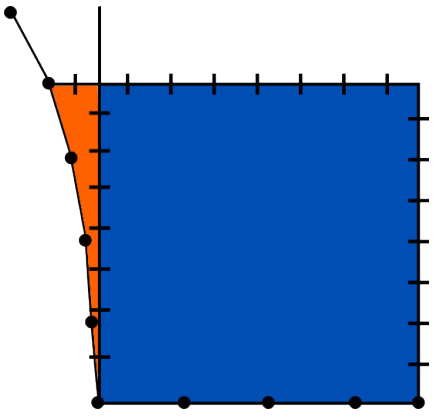
(a) Célula da estrutura de particionamento (em vermelho) juntamente com a borda de entrada.



(b) Arestas em azul entram no teste de proximidade.



(c) Após o teste de proximidade ser aplicado, o subdomínio azul fica responsável pela região onde as arestas falharam nos testes.



(d) Subdomínios que seriam gerados sem a realização dos testes de proximidade.

Figura 4.16: Modificações nas regiões realizadas pelo teste de proximidade para evitar elementos ruins.

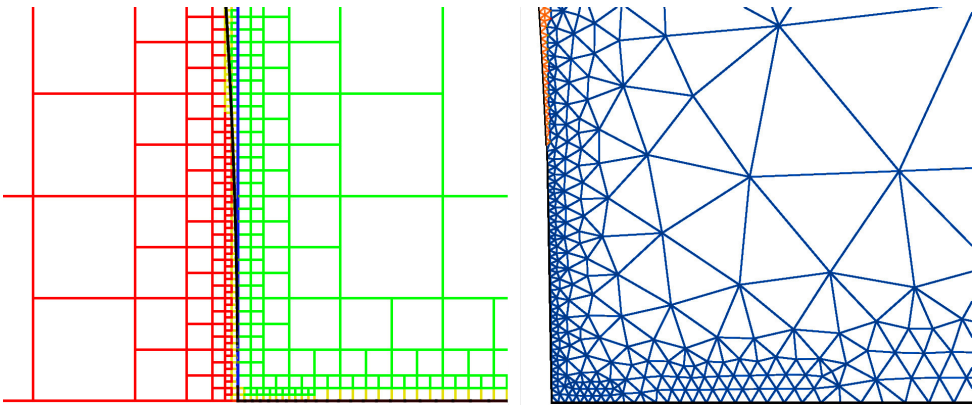


Figura 4.17: célula da *quadtree* de partição tem uma região que se torna tangente com a fronteira de entrada (à esquerda) e resultado final, onde malha que seria gerada pelo processador de cor laranja foi gerado pelo processador de cor azul (à direita).

com células classificadas como internas (Figura 4.19d).

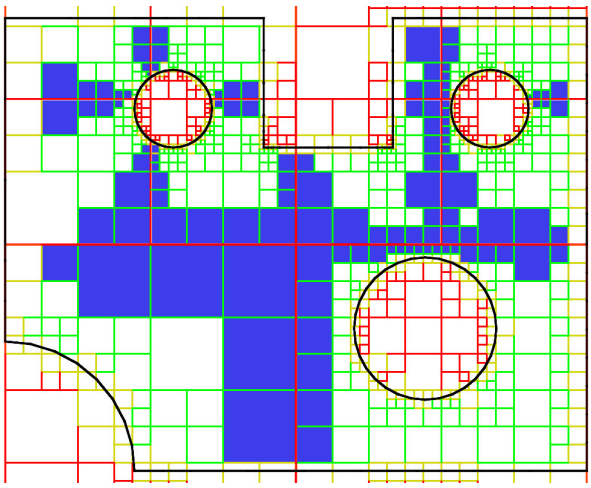
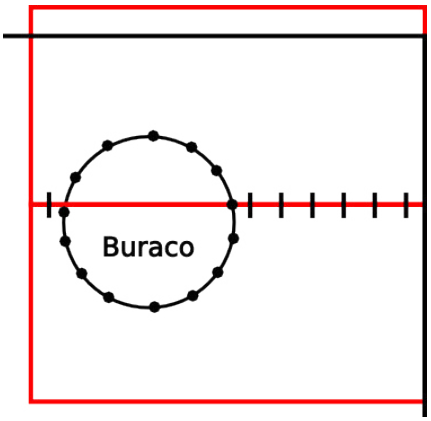
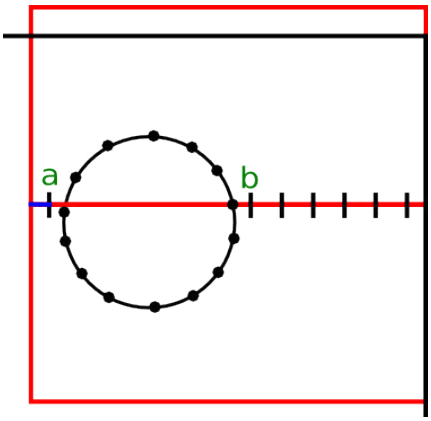


Figura 4.18: Exemplo onde o domínio possui buracos cortando uma partição.

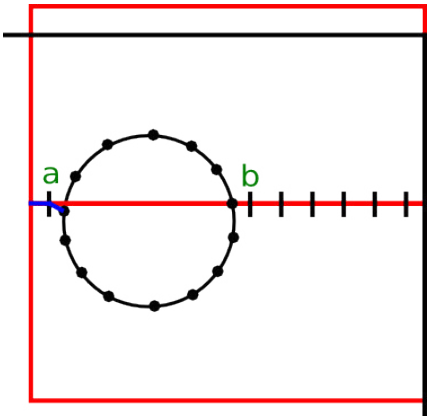


(a) Célula da estrutura de particionamento (em vermelho) juntamente com a borda de entrada que contém um buraco.

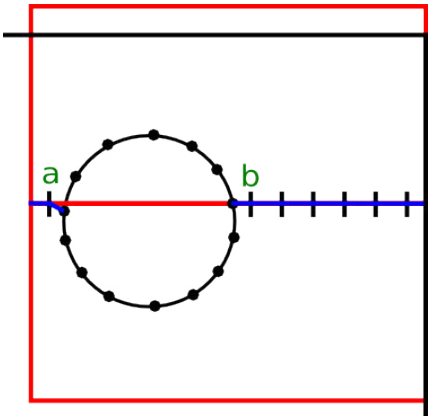


(b) Detecta-se que arestas que seria criada de a para b passa por um buraco.

Figura 4.19: Passos feitos nos tratamentos de buracos.



(c) Teste feito para selecionar a melhor aresta para a.



(d) Teste feito para selecionar a melhor aresta para b.

Figura 4.19: Passos feitos nos tratamentos de buracos (continuação).

4.5 Balanceamento de Carga

Após todos os subdomínios estarem devidamente criados, eles devem ser distribuídos entre os processadores disponíveis para que as submalhas sejam geradas por alguma técnica de geração de malha. Essa distribuição das tarefas deve ser feita de tal forma que todos os processadores gastem a mesma quantidade de tempo, evitando que algum processador fique sobrecarregado ou ocioso.

De acordo com o modo como o domínio é particionado, pode-se ter como resultado diversas tarefas para serem distribuídas (caso se utilize a *quadtree*) ou então uma quantidade de tarefas igual à de processadores (caso se utilize a BSP). Por isso duas estratégias de balanceamento são utilizadas neste trabalho, uma por demanda, quando utilizada a *quadtree*, e outra por balanceamento não-centralizado, quando utilizada a BSP.

Fazendo o particionamento com *quadtree*, os subdomínios criados são enviados aos processos escravos sob demanda. Inicialmente, o processo mestre retém todos os subdomínios. Um processo escravo requisita ao processo mestre um subdomínio e, assim que essa requisição for atendida, o processo escravo começa seu trabalho de geração de malha, enquanto que o processo mestre espera por outra requisição. Quando um processo escravo termina de trabalhar em um subdomínio, ele envia ao mestre outra requisição. Assim que todos os subdomínios tiverem sido enviados aos processos escravos, o processo mestre aguardará os resultados.

Quando uma BSP for usada para particionar, cada processo receberá uma única tarefa, onde cada tarefa corresponde a uma célula folha da BSP. Quando duas células que pertencem a um mesmo nó terminarem de fazer sua computação, um dos processadores dessas células ficará responsável por juntar as duas submalhas e fazer a melhoria na malha resultante. Esse processo é feito para cada par de células que pertence a um mesmo nó da BSP até chegar no nó raiz.

4.6 Geração de Malha

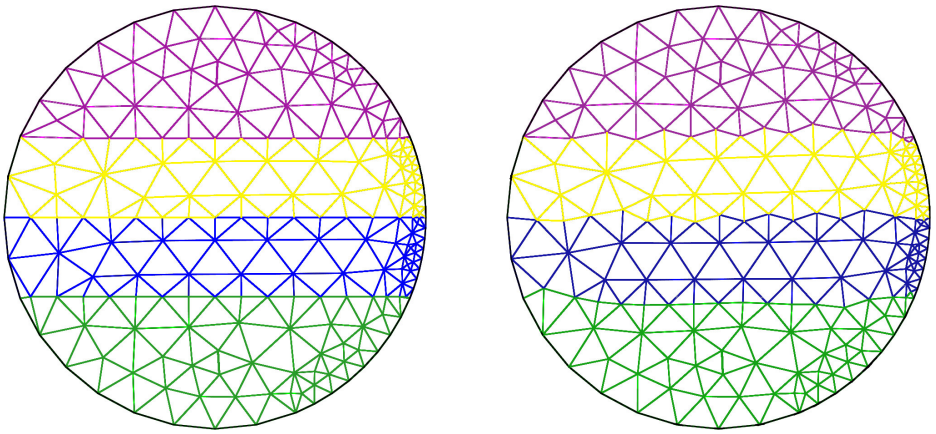
Cada processador tem a possibilidade de aplicar algoritmos diferentes se desejado, podendo aplicar qualquer técnica de geração de malha que respeite os pré-requisitos citados no início desta seção. Foi selecionado uma técnica de Avanço de Fronteira para realizar os testes.

O algoritmo de Avanço de Fronteira utilizado foi desenvolvido em (MIRANDA; CAVALCANTE-NETO; MARTHA, 1999) e (CAVALCANTE-NETO et al., 2001). Estruturas de dados geométricas foram utilizadas para acelerar a busca por vértices candidatos para a geração de um novo triângulo e para a busca de possíveis arestas intersectantes, garantindo uma rápida execução do procedimento de geração de malha.

4.7 Finalização da Malha

A finalização da malha é a junção dos diversos segmentos da fronteira dos subdomínios e melhorias (suavização Laplaciana) nas partes da malha. Estas partes consistem

de segmentos da fronteira juntamente com certas camadas de elementos adjacentes destes segmentos.



(a) Junção das malhas geradas nos diversos processadores.

(b) Malha resultante depois do último refinamento.

Figura 4.20: Finalização da malha: junção das malhas geradas pelos processadores (à esquerda) e malha refinada final (à direita).

A melhoria da malha é feita da região onde a malha de interface foi construída, juntamente com algumas camadas de triângulos adjacentes. Essas camadas são os elementos adjacentes à fronteira. Foi verificado em (ITO et al., 2007) que duas camadas de elementos são suficientes para uma boa malha. A camada 0 consiste dos próprios segmentos da fronteira, e a camada N compreende os elementos presentes na camada $N-1$ mais seus elementos adjacentes. Depois dessa etapa, a malha está completamente gerada (Figura 4.20b).

5 EXEMPLOS E RESULTADOS

Neste capítulo são apresentados os resultados obtidos pela técnica proposta neste trabalho. É também mostrada a diferença nos resultados que as estruturas de dados utilizadas no particionamento fazem, comparando a *quadtree* com a BSP. Os resultados para a técnica descrita neste trabalho são nomeados, como *a priori* e comparados com os resultados para os mesmos modelos no trabalho de (FREITAS et al., 2013) que são nomeados de *a posteriori*, pois essa técnica segue uma abordagem de particionamento diferente.

O mesmo critério de particionamento, que é baseado na carga média, foi adotado para os exemplos que utilizaram a *quadtree* como estrutura de particionamento. O algoritmo de geração de malha utilizado foi o mesmo em todos os testes para uma comparação justa entre os resultados.

A técnica descrita neste trabalho foi implementada 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. As versões com interface foram ainda implementadas com a biblioteca de renderização OpenGL (com suas bibliotecas glu, glut e glew), e a biblioteca de interfaces wxWidgets.

O computador utilizado para rodar os testes da implementação foi o cluster do CENAPAD-UFC com 48 nós, cada um com dois processadores Intel Westmere 6 cores X5650 EP e 24 GB de RAM, totalizando 576 núcleos de processamento e 1152GB de RAM. Os testes foram feitos utilizando 8 nós com memória distribuída e o gerador de malha apresentado na seção 4.6.

5.1 Modelos

Ao todo, foram executados 3 modelos de diferentes geometrias, alguns dos quais com buracos, para avaliar o comportamento da técnica em diferentes entradas. A Figura 5.1 mostra as fronteiras utilizadas pelo procedimento de geração em paralelo de malhas.

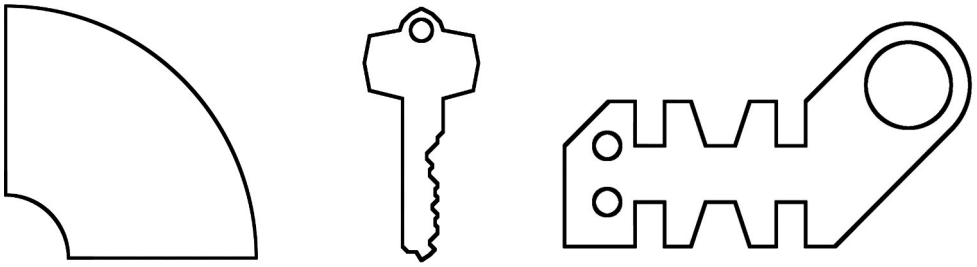


Figura 5.1: Modelos utilizados para testes: Cilindro, Chave e Placa.

5.2 Tamanho das Malhas

A Figura 5.2 mostra os tamanhos das malhas geradas sequencialmente, onde todos os resultados usam os mesmos modelos para a validação das ideias propostas neste trabalho. Como pode ser visto, os tamanhos das malhas variam aproximadamente de 1,2 milhões a 1,8 milhões de elementos. Os tamanhos das malhas e a quantidade de processadores utilizados são razoáveis para os teste aqui realizados, ilustrando bem o comportamento da técnica nestes modelos.

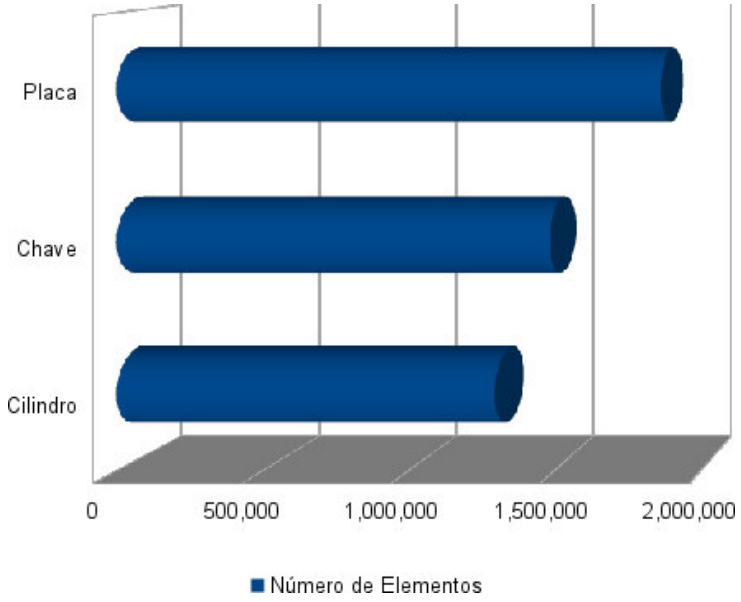


Figura 5.2: Quantidade de elementos gerados nos exemplos.

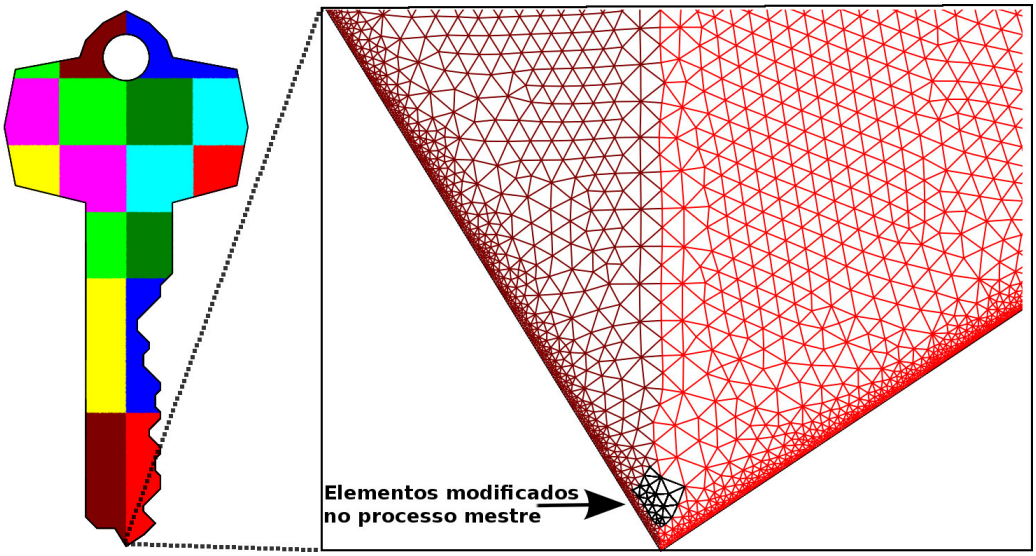


Figura 5.3: Região entre submalhas.

A Figura 5.3 destaca uma região entre dois subdomínios do modelo Chave, com o objetivo de mostrar como as malhas geradas em paralelo se adequam perfeitamente (cada

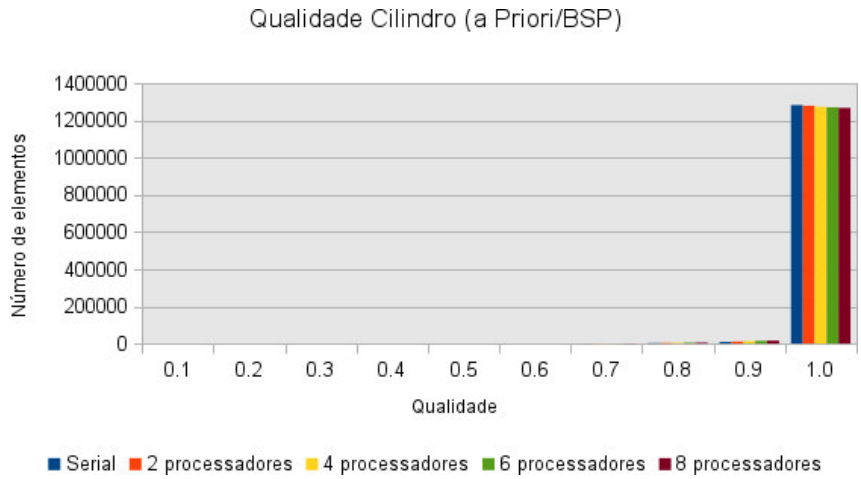
cor indica um processo diferente). Como pode ser visto, as funções de tamanho e forma nos triângulos gerados foram respeitadas, mesmo que eles tenham sido gerados por diferentes processos em diferentes subdomínios, ou mesmo pelo processo mestre, no passo de finalização da malha (elementos em preto).

5.3 Qualidade

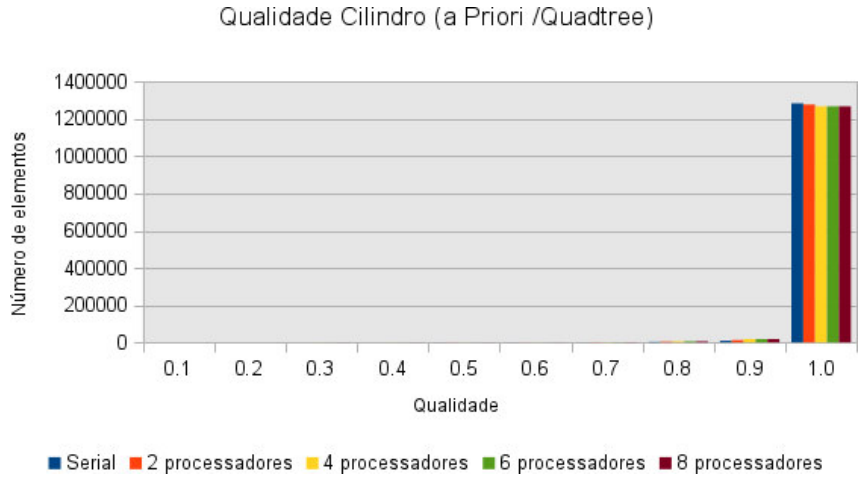
Para avaliar a qualidade das malhas geradas é utilizada uma métrica que usa $\alpha = 2R_i/R_c$, onde R_i e R_c são os raios dos círculos inscrito e circunscrito a um dado triângulo da malha, respectivamente. Esta métrica α tem valor 1,0 para um triângulo equilátero. Quanto pior for a qualidade de um elemento, mais próximo de 0,0 é o valor de α. Um elemento com $\alpha \leq 0{,}1$ é dito ter uma qualidade muito pobre, enquanto que elementos com $\alpha \geq 0{,}7$ têm boa qualidade.

A qualidade da malha de cada um dos três modelos é mostrada, respectivamente, nas figuras 5.4, 5.5 e 5.6. Nessas figuras, os elementos gerados recebem uma classificação entre 0,1 e 1,0. Pode-se ver que existe um grande espaço vazio no início dos gráficos e logo após uma grande quantidade de elementos de boa qualidade, com $\alpha \geq 0{,}7$. É visível que praticamente todos os elementos gerados se encontram numa estreita faixa próxima ao valor 1,0 (máxima qualidade) e que a quantidade de elementos nessa faixa é praticamente a mesma das malhas malhas geradas de forma sequencial. No geral, esse gráfico mostra que, para as três abordagens(*a priori* com particionamento por BSP, *a priori* com particionamento por *quadtree* e *a posteriori* com particionamento por *quadtree*) a malha resultante é bastante boa.

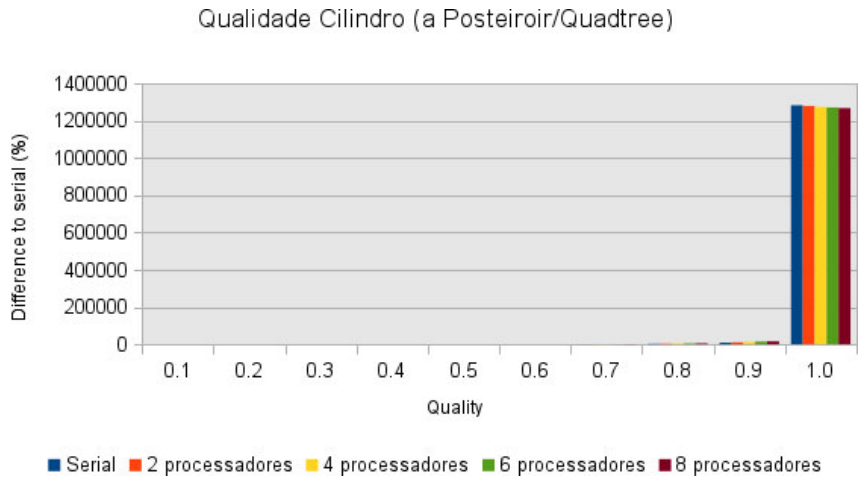
As Figuras 5.7 e 5.8 mostram a diferença em porcentual da qualidade da malha gerada sequencialmente com a gerada em paralelo. Para a técnica *a priori* houve um crescimento na qualidade dos elementos gerados em paralelo na faixa de 0,6 a 0,9 e um decaimento na faixa próxima do valor 1,0. Ou seja, alguns elementos classificados como 1,0 reduziram sua qualidade e foram para uma outra faixa de valor um pouco mais abaixo, mas, mesmo assim, continuam com uma boa qualidade. Como visto na figura, a variação de qualidade para melhor ou pior não passa de 1,2% para a abordagem *a priori* com *quadtree*, 0,7% para a abordagem *a priori* com BSP e de 0,06% para para a abordagem *a posteriori* com *quadtree*. A explicação disso é que em técnicas de particionamento *a priori* são inseridos elementos que cortam a borda para gerar novos subdomínios, e esses cortes podem ficar em regiões geometricamente ruins, podendo influenciar diretamente nos elementos que serão gerados. Já nas técnicas de decomposição *a posteriori* não é gerado nenhuma nova fronteira na criação dos subdomínios, fazendo assim a nova malha ser bem mais parecida com a sequencial. Isso, entretanto, não é um problema sério pois a qualidade da malha não varia muito como foi mostrado e esse aspecto ocorre, em geral, em regiões internas do modelo, onde isso não é crucial na maioria dos casos.



(a) Qualidade para a técnica *a priori* com particionamento por BSP para o Cilindro.

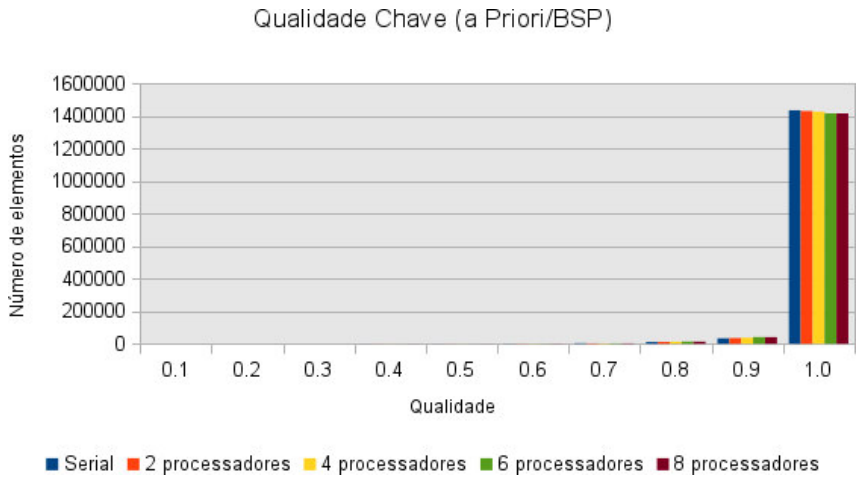


(b) Qualidade para a técnica *a priori* com particionamento por *quadtree* para o Cilindro.

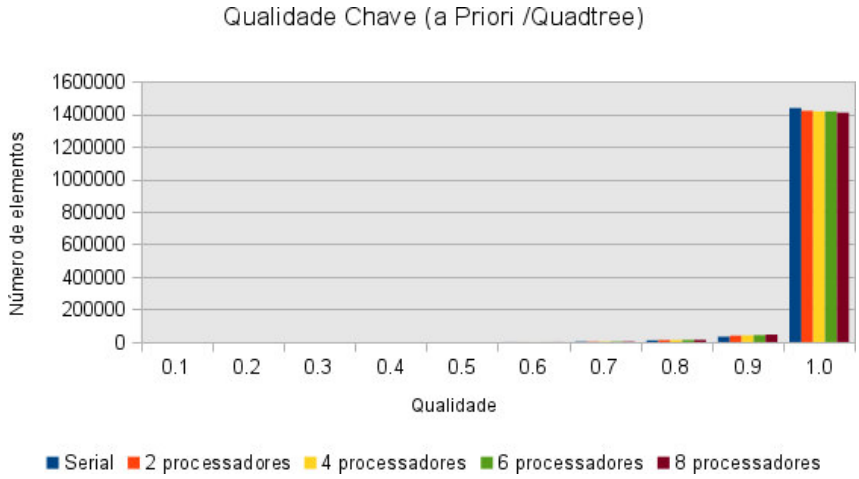


(c) Qualidade para a técnica *a posteriori* com particionamento por *quadtree* para o Cilindro.

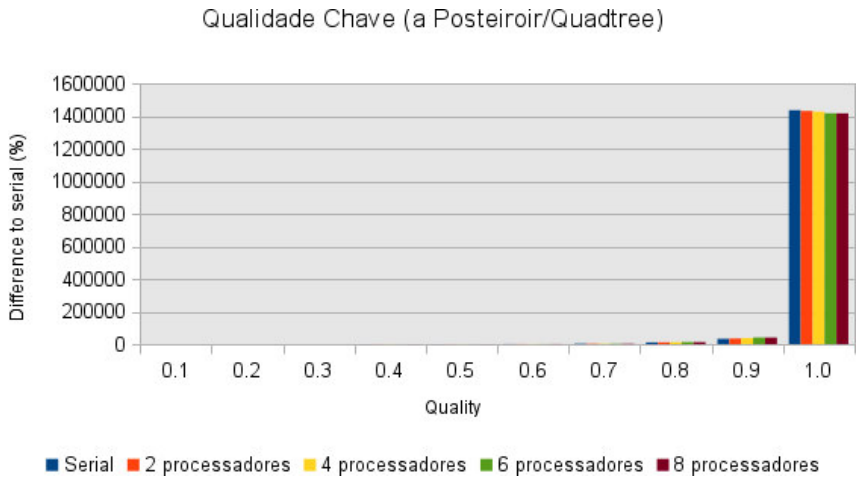
Figura 5.4: Qualidade para o Cilindro nas três abordagens.



(a) Qualidade para a técnica *a priori* com particionamento por BSP para a Chave.

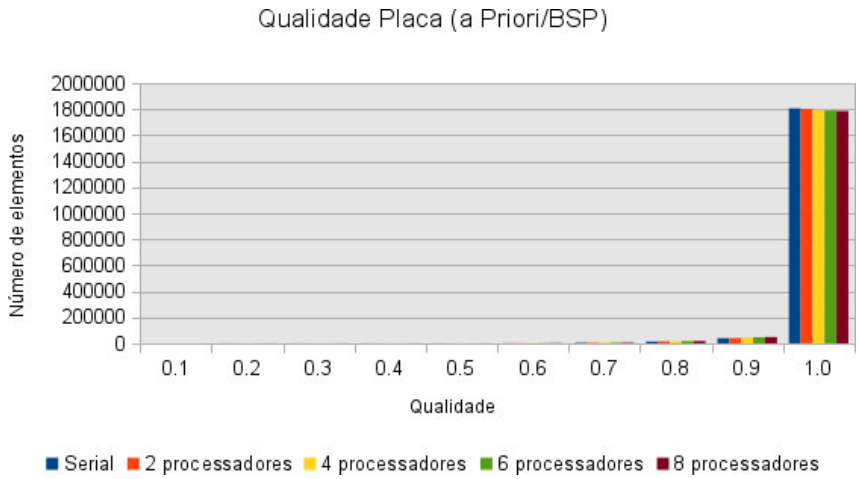


(b) Qualidade para a técnica *a priori* com particionamento por *quadtree* para a Chave.

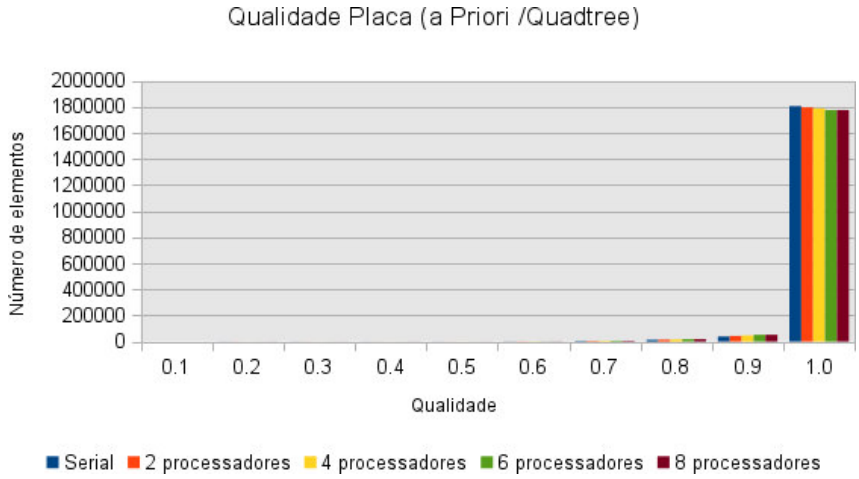


(c) Qualidade para a técnica *a posteriori* com particionamento por *quadtree* para a Chave.

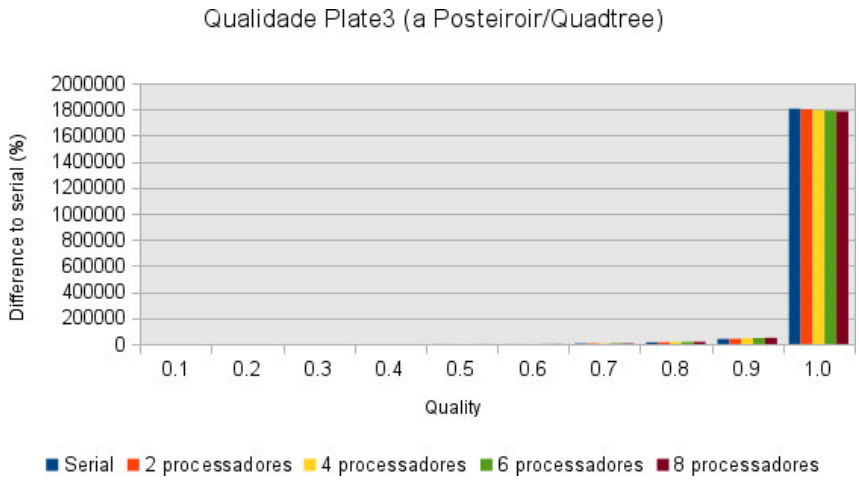
Figura 5.5: Qualidade para a Chave nas três abordagens.



(a) Qualidade para a técnica *a priori* com particionamento por BSP para a Placa.



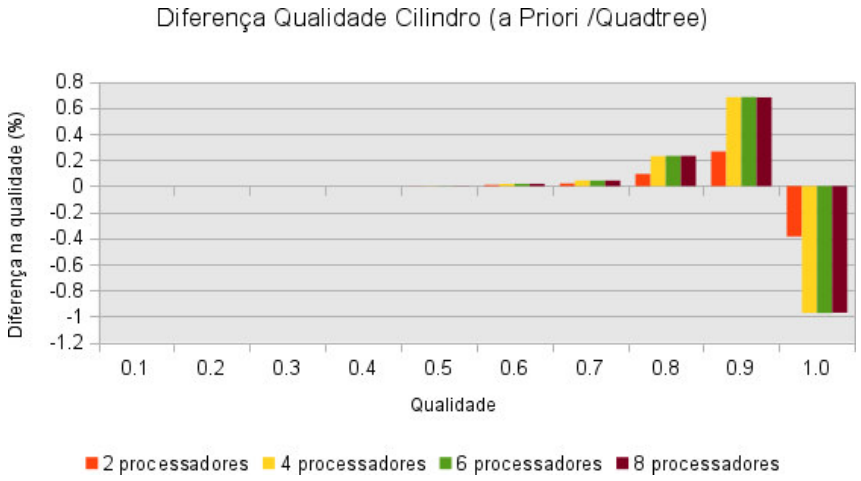
(b) Qualidade para a técnica *a priori* com particionamento por *quadtree* para a Placa.



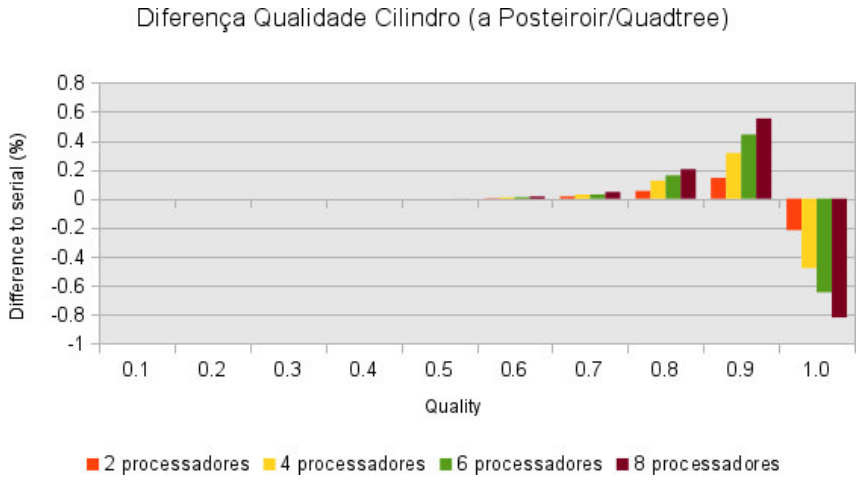
(c) Qualidade para a técnica *a posteriori* com particionamento por *quadtree* para a Placa.

Figura 5.6: Qualidade para a Placa nas três abordagens.

Os gráficos de diferença de qualidade ajudam a comparar as malhas geradas em paralelo com as sequenciais. Nesses gráficos, barras acima de zero indicam que mais elementos naquela faixa de valor da métrica de qualidade foram gerados na versão paralela do que na sequencial; barras abaixo de zero indicam que existem mais elementos gerados sequencialmente naquela determinada faixa de valor do que na versão paralela. Algo interessante de se observar é que à medida que mais processos participam da geração da malha, mais essa diferença aumenta. Isso ocorre porque quando se adiciona um novo processo na geração da malha, se adiciona também um novo subdomínio para esse processo. Quanto mais subdivisões forem feitas, mais a malha final vai divergir da malha sequencial.

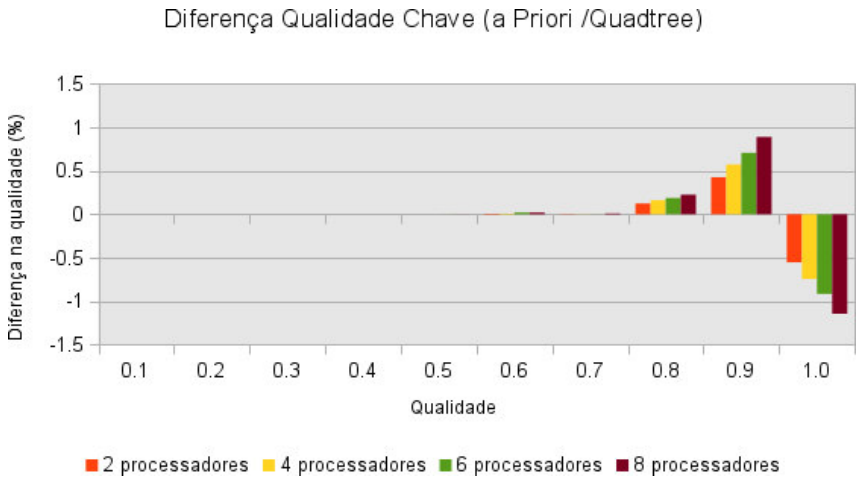


(a) Diferença de qualidade para a técnica *a priori* com particionamento por *quadtree* para o Cilindro.

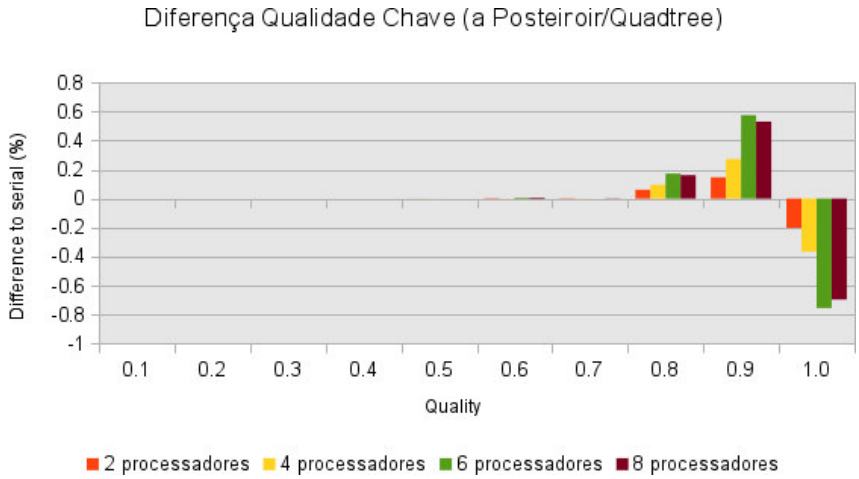


(b) Diferença de qualidade para a técnica *a posteriori* com particionamento por *quadtree* para o Cilindro.

Figura 5.7: Diferença de qualidade para as técnica *a priori* e *posteriori*, utilizando *quadtree* para particionamento.

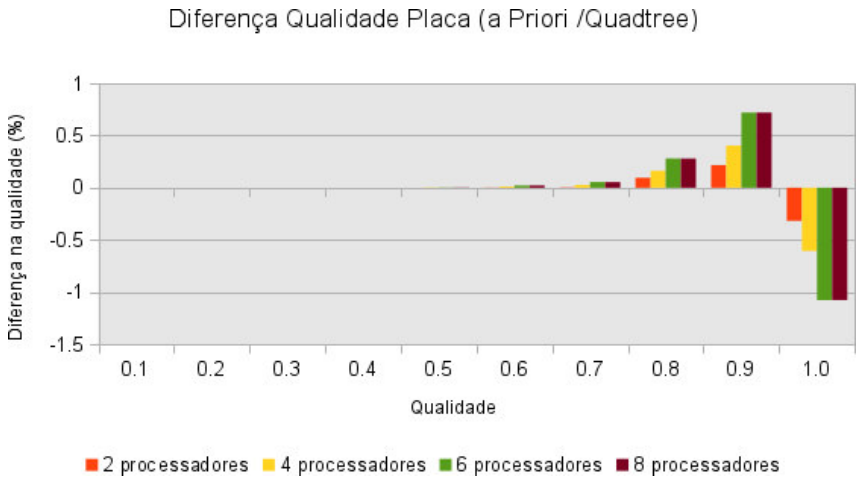


(c) Diferença de qualidade para a técnica *a priori* com particionamento por *quadtree* para a Chave.

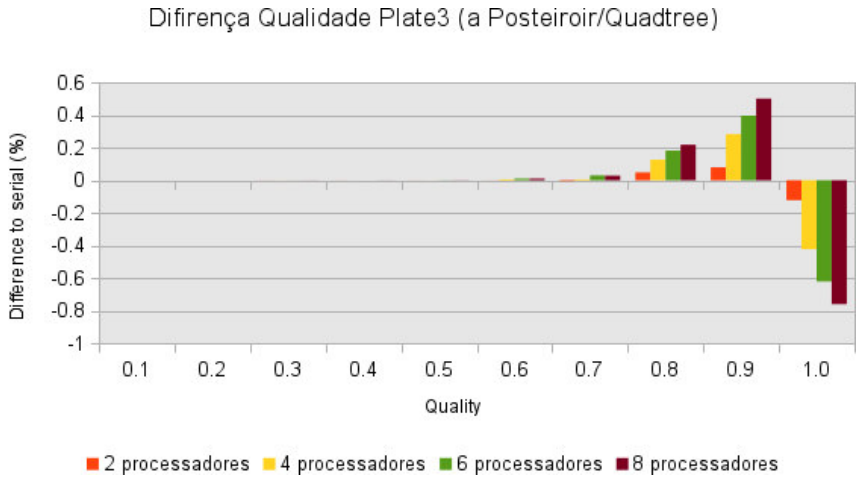


(d) Diferença de qualidade para a técnica *a posteriori* com particionamento por *quadtree* para a Chave.

Figura 5.7: Diferença de qualidade para as técnica *a priori* e *posteriori*, utilizando *quadtree* para particionamento (continuação).



(e) Diferença de qualidade para a técnica *a priori* com particionamento por *quadtree* para a Placa.

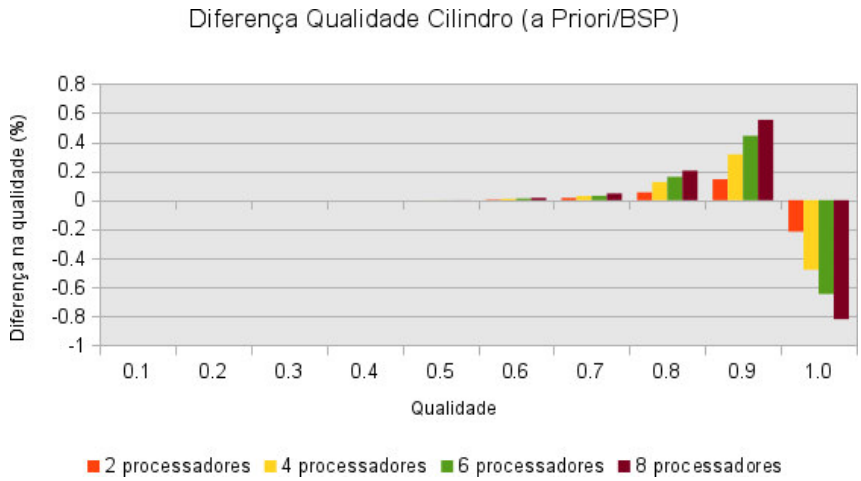


(f) Diferença de qualidade para a técnica *a posteriori* com particionamento por *quadtree* para a Placa.

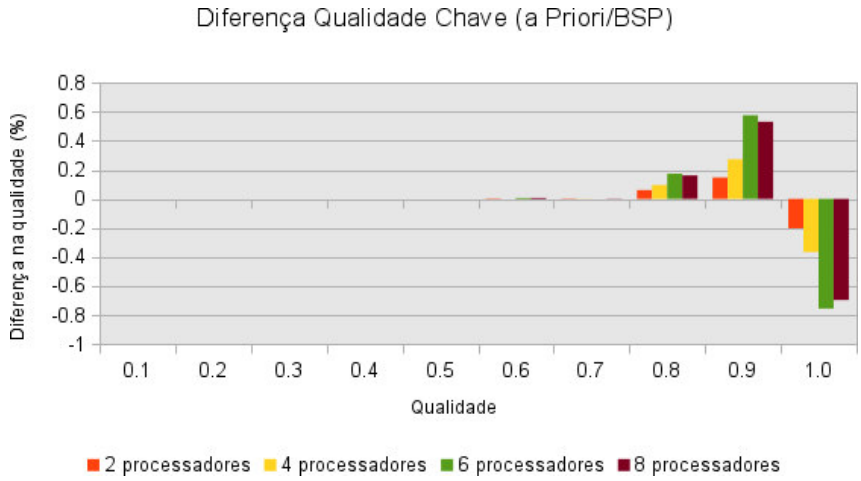
Figura 5.7: Diferença de qualidade para as técnica *a priori* e *posteriori*, utilizando *quadtree* para particionamento (continuação).

Como pode ser visto nas figuras desta seção, em todos os casos, as malhas geradas apresentam boa qualidade. Somente uma pequena porcentagem de elementos apresentam qualidade muito diferente de 1,0. Outro fato que se observa é que à medida que se aumenta o número de processadores a qualidade dos elementos diminui, e isso ocorre por conta da quantidade de subdomínios que aumenta à medida que novos processadores são usados. A utilização do particionamento por BSP melhora a qualidade, mas, mesmo assim, não apresenta uma qualidade superior à da técnica *a posteriori*.

No entanto, a técnica paralela descrita neste trabalho gera uma malha de qualidade aproximadamente igual à da malha gerada sequencialmente. Note também que um número maior de subdomínios criados geralmente leva a diminuição da qualidade da malha.

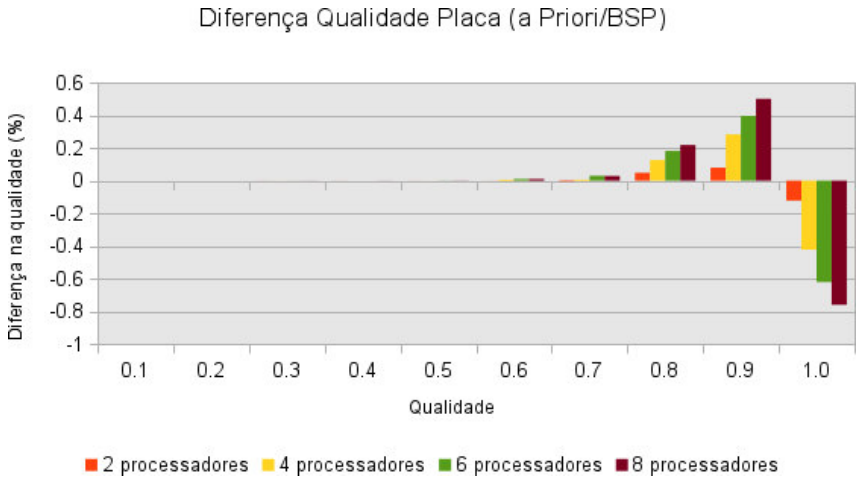


(a) Diferença de qualidade para a técnica *a priori* com particionamento por BSP para o Cilindro.



(b) Diferença de qualidade para a técnica *a priori* com particionamento por BSP para a Chave.

Figura 5.8: Diferença de qualidade dos elementos em porcentagem para as técnica *a priori* utilizando BSP para particionamento.



(c) Diferença de qualidade para a técnica *a priori* com particionamento por BSP para a Placa.

Figura 5.8: Diferença de qualidade dos elementos em porcentagem para as técnica *a priori* utilizando BSP para particionamento (continuação).

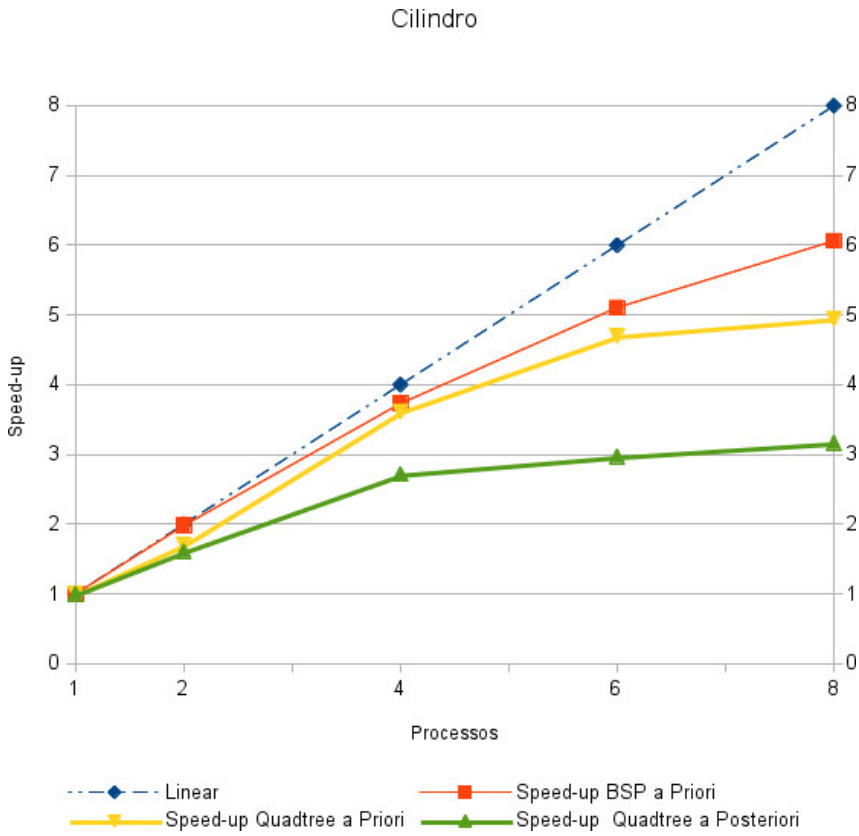
5.4 Tempo de execução e *speed-up*

Os gráficos da Figura 5.9 mostram o *speed-up* atingido pela implementação das duas técnicas mudando somente a estrutura de particionamento para a técnica *a priori*. Como dito anteriormente na Seção 2.4.3, o *speed-up* indica quantas vezes a implementação foi mais rápida quando uma determinada quantidade de processos foi utilizada. Idealmente, uma implementação paralela teria um *speed-up* linear, significando que n processos a tornam n vezes mais rápida. Na prática, isso é difícil de ser obtido, por causa das inevitáveis porções sequenciais presentes no algoritmo paralelo, além do tempo de comunicação entre os processos, que não existe no caso sequencial.

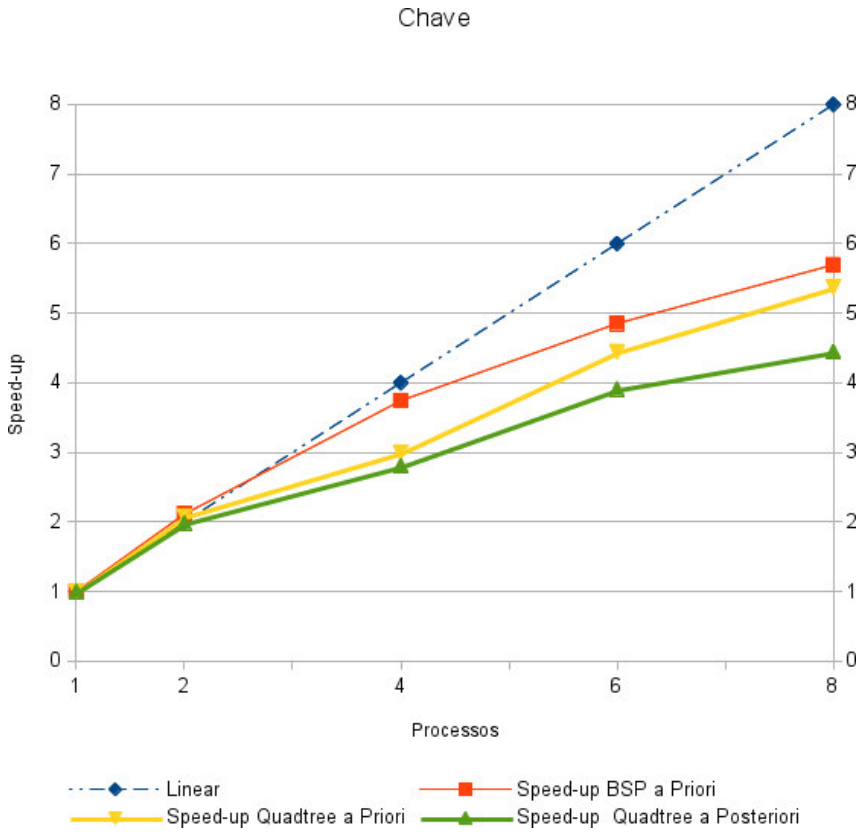
Do lado esquerdo dos gráficos é mostrado o valor do *speed-up*. No eixo X é mostrada a variação na quantidade de processos utilizados na geração da malha. Pode-se notar que o *speed-up* da técnica descrita neste trabalho foi bem superior ao da técnica *a posteriori*. A utilização da BSP para particionamento melhorou o *speed-up* quando comparado com as outras duas abordagens.

Na Figura 5.10, é mostrado o tempo de execução atingido pela implementação das duas técnicas mudando somente a estrutura de particionamento para a técnica *a priori*. Do lado esquerdo dos gráficos é mostrado o tempo de execução em segundos. A abordagem *a priori* com decomposição baseada na BSP obteve os melhores resultados em todos os exemplos.

A técnica *a priori* é mais rápida que *a posteriori* além de apresentar uma escalabilidade melhor. O uso da BSP melhora os resultados e isso se deve a uma melhor adaptação desta estrutura de dados ao domínio dado como entrada. Os próximos resultados que serão exibidos não apresentam mais a técnica *a posteriori* pois a técnica *a priori* mostrou-se mais rápida em todos os modelos. Então, a partir de agora, são comparadas somente as técnicas *a priori* com particionamento feito por BSP e por *quadtree*.

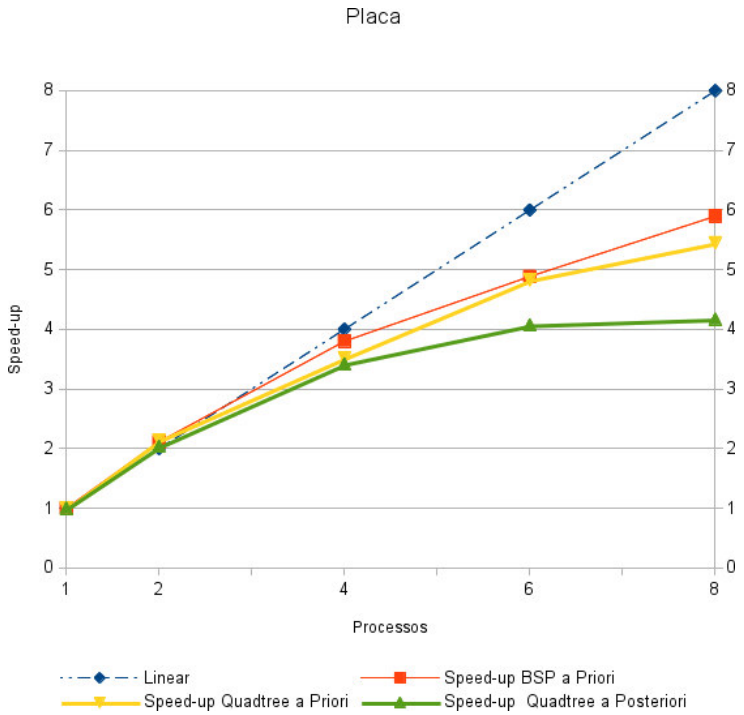


(a) *Speed-up* para as três abordagens no modelo Cilindro.



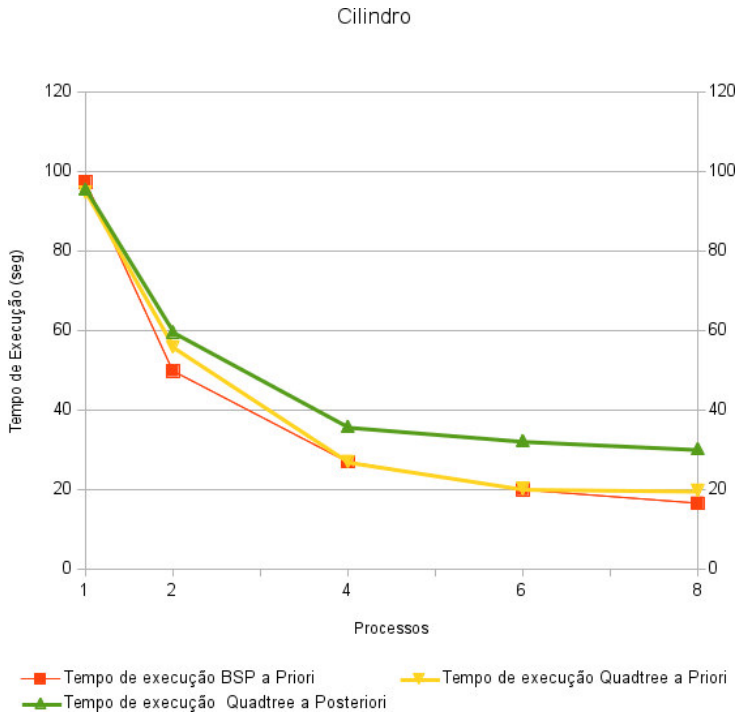
(b) *Speed-up* para as três abordagens no modelo Chave.

Figura 5.9: *Speed-up* para as três abordagens.



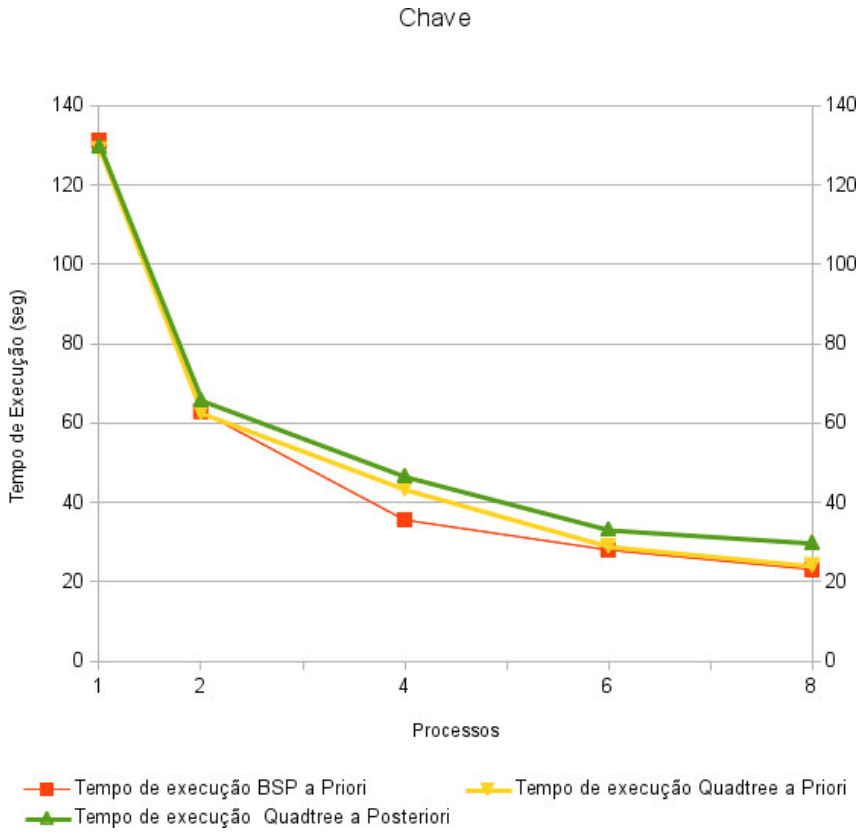
(c) *Speed-up* para as três abordagens no modelo Placa.

Figura 5.9: *Speed-up* para as três abordagens (continuação).

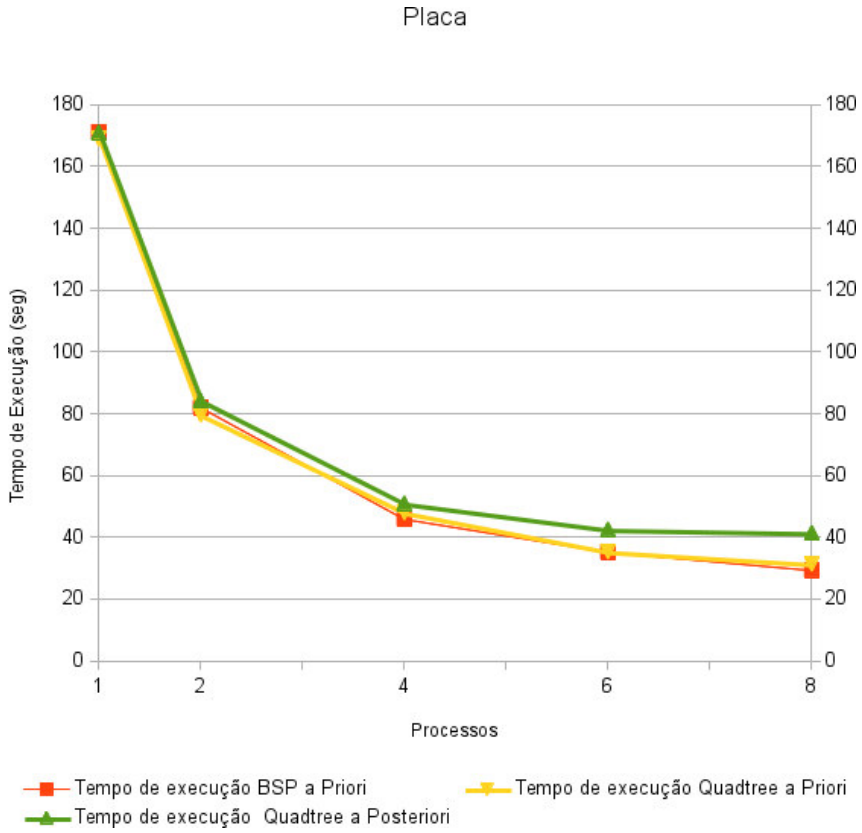


(a) Tempo de execução para as três abordagens no modelo Cilindro.

Figura 5.10: Tempo de execução para as três abordagens.



(b) Tempo de execução para as três abordagens no modelo Chave.



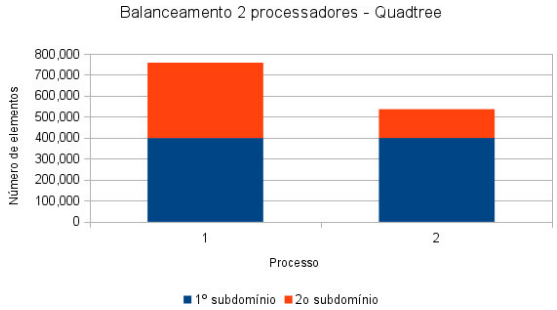
(c) Tempo de execução para as três abordagens no modelo Placa.

Figura 5.10: Tempo de execução para as três abordagens (continuação).

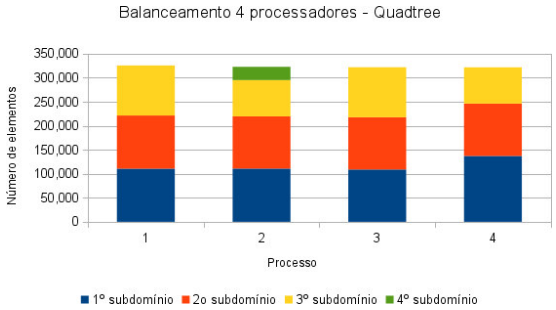
5.5 Balanceamento de Carga

Nesta seção, é mostrado o balanceamento da carga, levando em conta o número de elementos gerados em cada subdomínio, que é considerado neste trabalho como carga. Os gráficos das Figuras 5.11, 5.13 e 5.15 mostram que, para os três exemplos, a técnica, que utiliza particionamento por *quadtree*, tem uma carga bem distribuída entre os processos. A esquerda dos gráficos fica a quantidade de elementos gerados, no eixo X é exibida a quantidade de processos e nas barras de valores ficam os subdomínios representados por diferentes cores. Utilizando um particionamento com *quadtree*, a quantidade de subdomínios criados pode ser maior que a quantidade de processos disponíveis, e por isso aparecem processos aos quais estão atribuídos dois ou mais subdomínios.

Nas figuras 5.12, 5.14 e 5.16, estão exibidos os gráficos para o balanceamento utilizando particionamento por BSP. Além da quantidade de elementos, é exibido o tempo de execução em segundos, representados pelas barras laranja e azul respectivamente. Pode-se observar que o tempo de execução é diretamente proporcional à quantidade de elementos gerados e que tanto o tempo de execução como o número de elementos se mantêm balanceados nos gráficos. Até no exemplo da Chave, que possui uma geometria bastante complexa, as barras de tempo de execução e a quantidade de elementos se mantiveram niveladas.

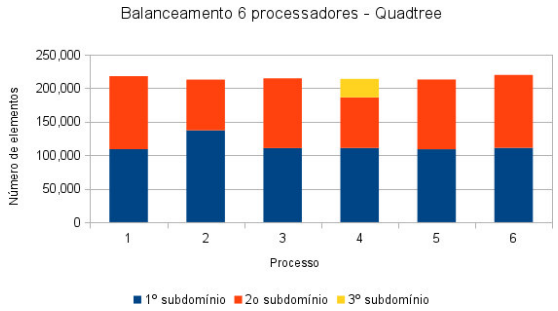


(a) Balanceamento pelo número de elementos do exemplo Cilindro com 2 processos com particionamento por *quadtree*.

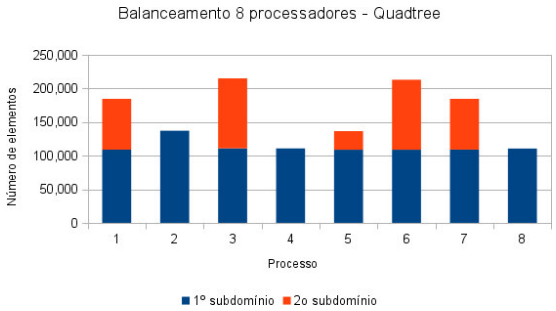


(b) Balanceamento pelo número de elementos do exemplo Cilindro com 4 processos com particionamento por *quadtree*.

Figura 5.11: Balanceamento pelo número de elementos do exemplo Cilindro para 2, 4, 6 e 8 processos com particionamento por *quadtree*.

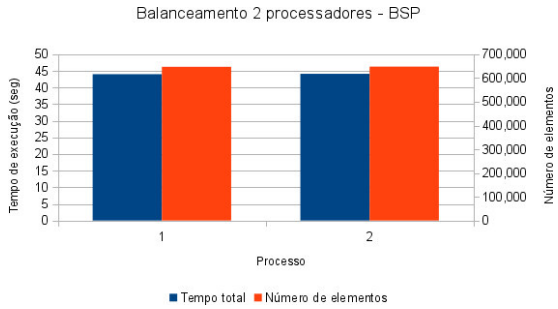


(c) Balanceamento pelo número de elementos do exemplo Cilindro com 6 processos com particionamento por *quadtree*.

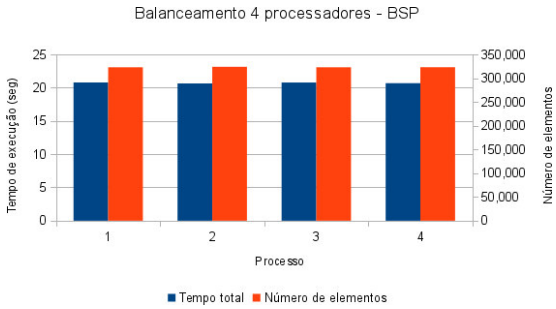


(d) Balanceamento pelo número de elementos do exemplo Cilindro com 8 processos com particionamento por *quadtree*.

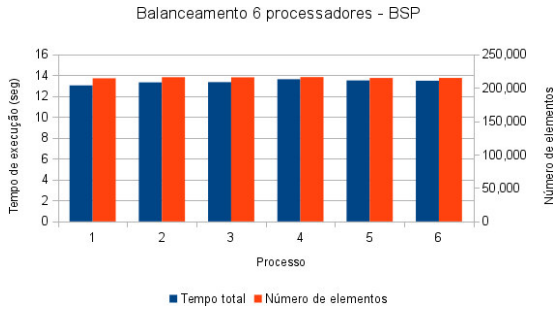
Figura 5.11: Balanceamento pelo número de elementos do exemplo Cilindro para 2, 4, 6 e 8 processos com particionamento por *quadtree* (continuação).



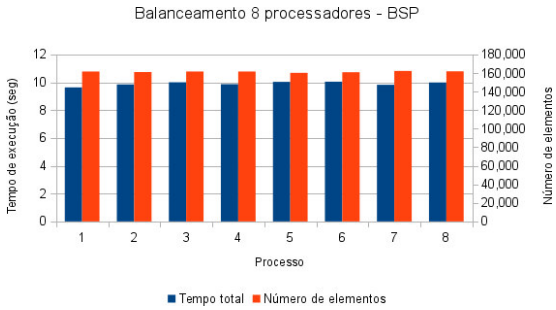
(a) Balanceamento pelo número de elementos do exemplo Cilindro com 2 processos com particionamento por BSP.



(b) Balanceamento pelo número de elementos do exemplo Cilindro com 4 processos com particionamento por BSP.

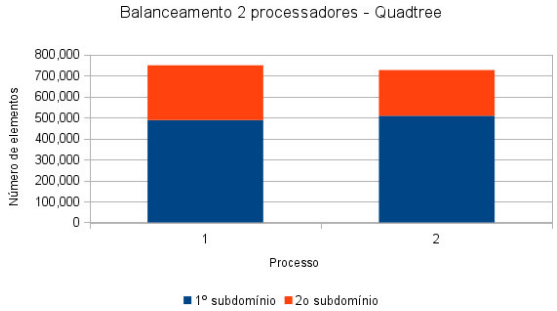


(c) Balanceamento pelo número de elementos do exemplo Cilindro com 6 processos com particionamento por BSP.

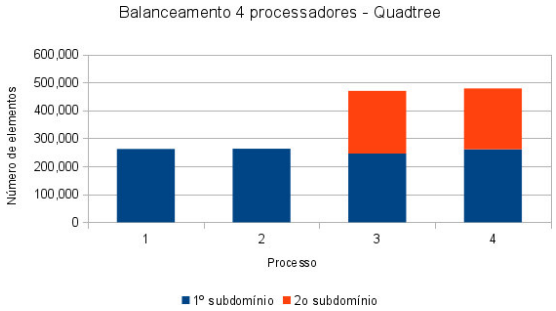


(d) Balanceamento pelo número de elementos do exemplo Cilindro com 8 processos com particionamento por BSP.

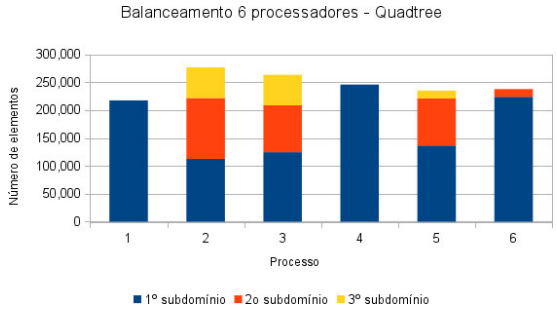
Figura 5.12: Balanceamento pelo número de elementos do exemplo Cilindro para 2, 4, 6 e 8 processos com particionamento por BSP.



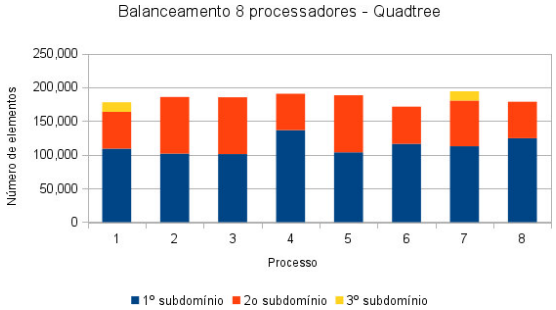
(a) Balanceamento pelo número de elementos do exemplo Chave com 2 processos com particionamento por *quadtree*.



(b) Balanceamento pelo número de elementos do exemplo Chave com 4 processos com particionamento por *quadtree*.

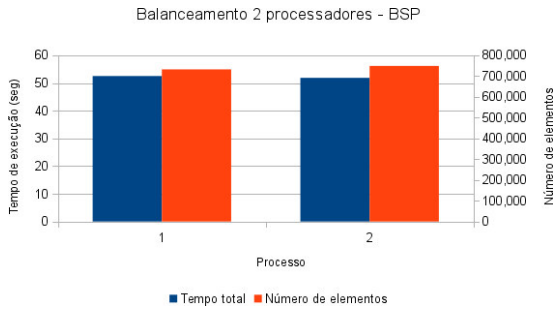


(c) Balanceamento pelo número de elementos do exemplo Chave com 6 processos com particionamento por *quadtree*.

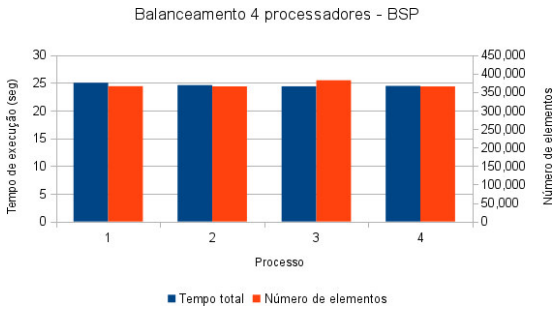


(d) Balanceamento pelo número de elementos do exemplo Chave com 8 processos com particionamento por *quadtree*.

Figura 5.13: Balanceamento pelo número de elementos do exemplo Chave para 2, 4, 6 e 8 processos com particionamento por *quadtree*.

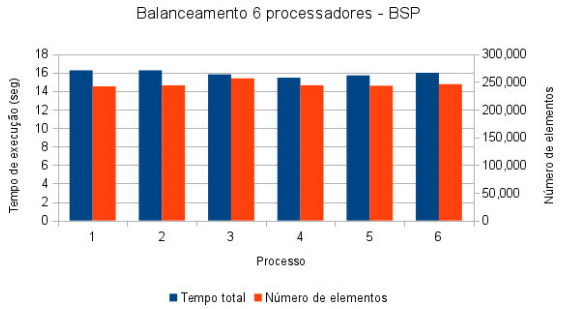


(a) Balanceamento pelo número de elementos do exemplo Chave com 2 processos com particionamento por BSP.

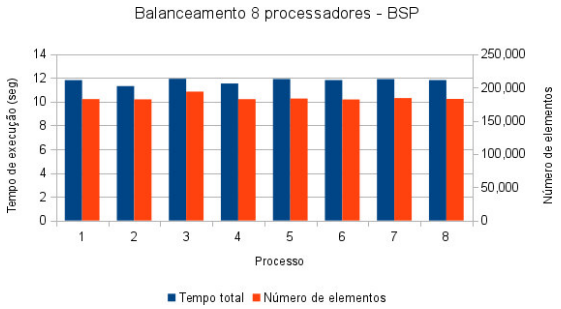


(b) Balanceamento pelo número de elementos do exemplo Chave com 4 processos com particionamento por BSP.

Figura 5.14: Balanceamento pelo número de elementos do exemplo Chave para 2, 4, 6 e 8 processos com particionamento por BSP.

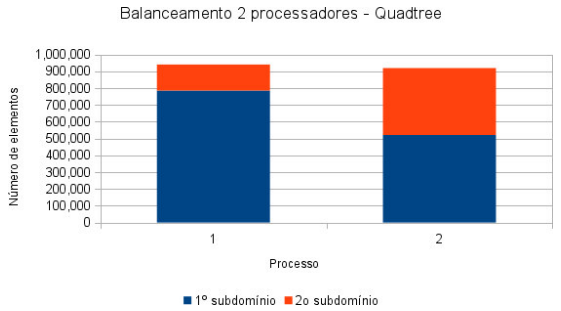


(c) Balanceamento pelo número de elementos do exemplo Chave com 6 processos com particionamento por BSP.

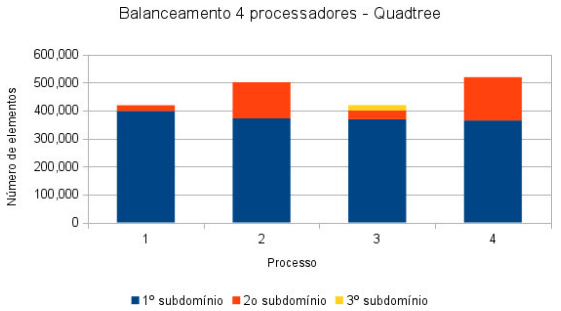


(d) Balanceamento pelo número de elementos do exemplo Chave com 8 processos com particionamento por BSP.

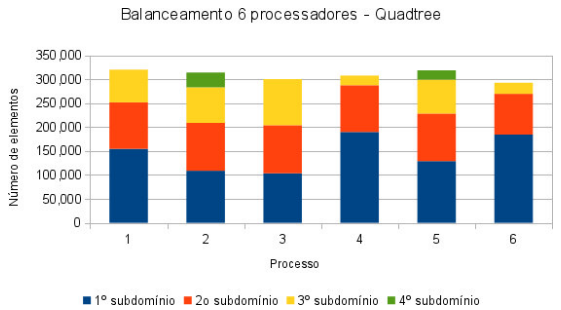
Figura 5.14: Balanceamento pelo número de elementos do exemplo Chave para 2, 4, 6 e 8 processos com particionamento por BSP (continuação).



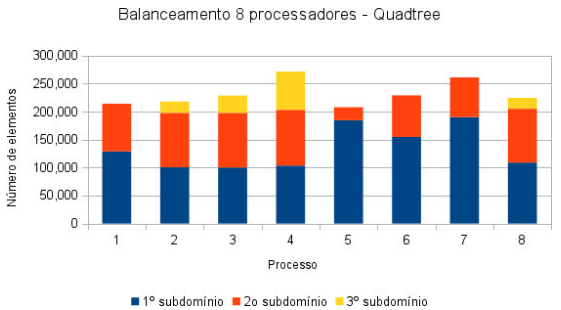
(a) Balanceamento pelo número de elementos do exemplo Placa com 2 processos com particionamento por *quadtree*.



(b) Balanceamento pelo número de elementos do exemplo Placa com 4 processos com particionamento por *quadtree*.

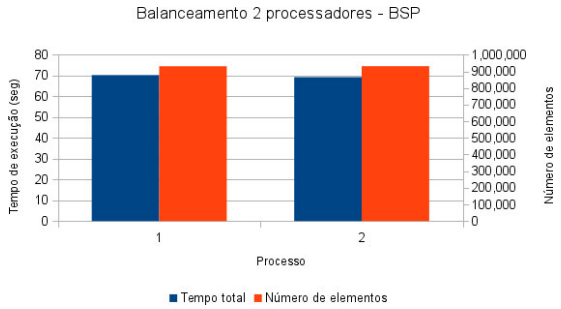


(c) Balanceamento pelo número de elementos do exemplo Placa com 6 processos com particionamento por *quadtree*.

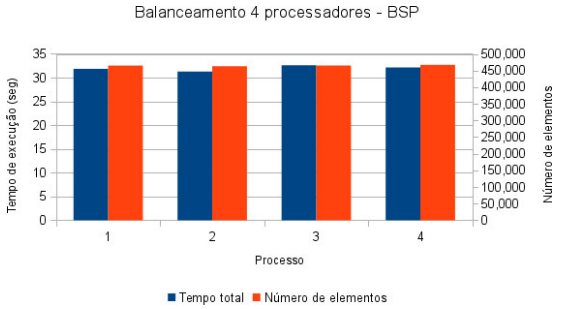


(d) Balanceamento pelo número de elementos do exemplo Placa com 8 processos com particionamento por *quadtree*.

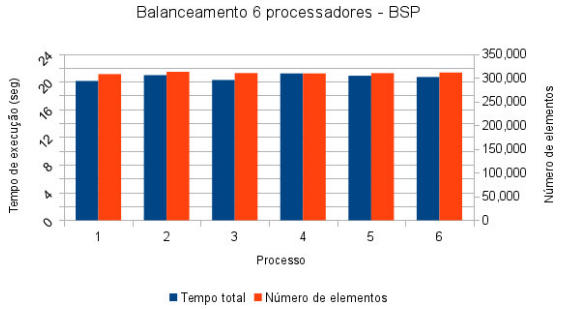
Figura 5.15: Balanceamento pelo número de elementos do exemplo Placa para 2, 4, 6 e 8 processos com particionamento por *quadtree*.



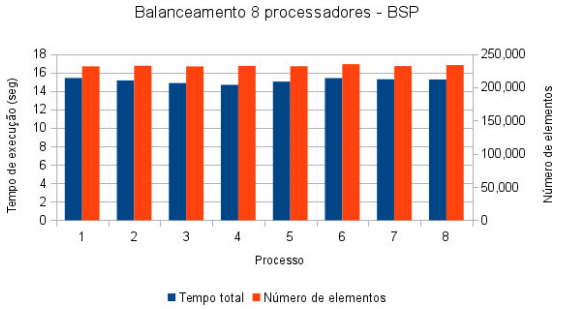
(a) Balanceamento pelo número de elementos do exemplo Placa com 2 processos com particionamento por BSP.



(b) Balanceamento pelo número de elementos do exemplo Placa com 4 processos com particionamento por BSP.



(c) Balanceamento pelo número de elementos do exemplo Placa com 6 processos com particionamento por BSP.



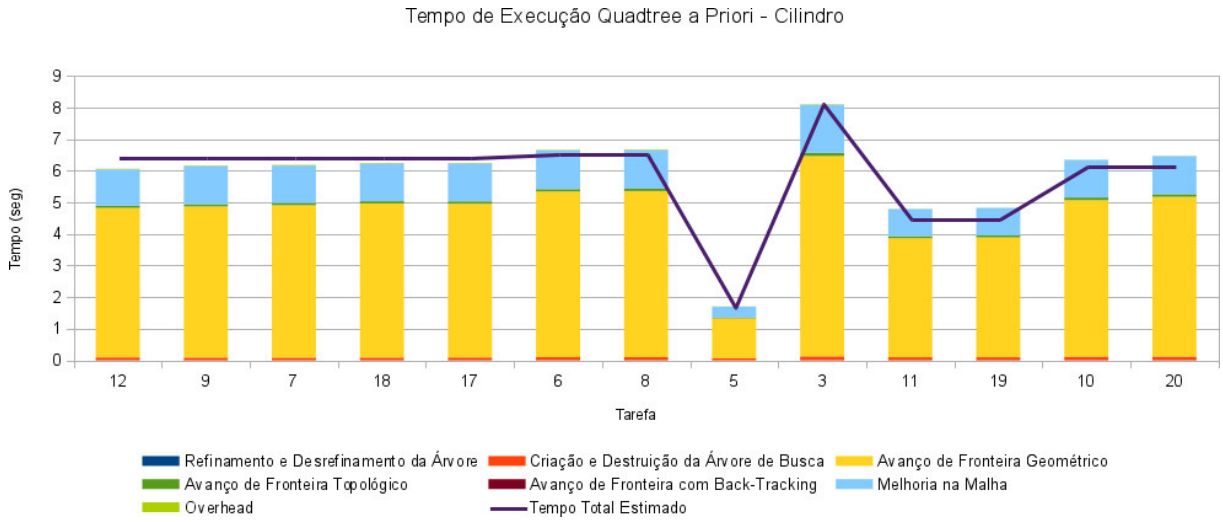
(d) Balanceamento pelo número de elementos do exemplo Placa com 8 processos com particionamento por BSP.

Figura 5.16: Balanceamento pelo número de elementos do exemplo Placa para 2, 4, 6 e 8 processos com particionamento por BSP.

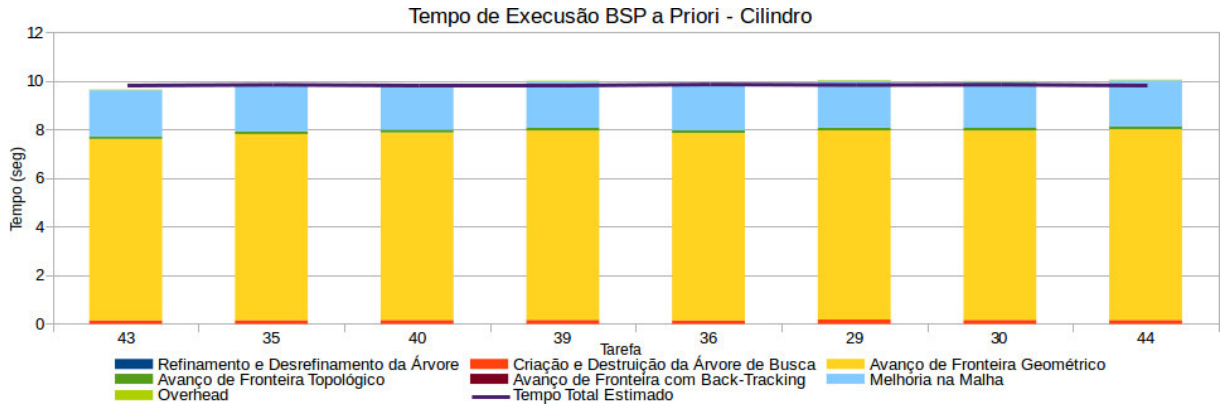
A grande vantagem da BSP se encontra na criação dos subdomínios, onde a melhor posição para se gerar uma fronteira é encontrada. Isso mantém um balanceamento muito bom e evita a criação excessiva de subdomínios, reduzindo assim tempos de comunicação. Com o uso da *quadtree* nem sempre um bom balanceamento é possível como mostrado da figura 5.11d, por exemplo. Um estratégia que pode melhorar o particionamento é subdividir o domínio até que a diferença de carga nos processos seja menor que um dado limite, pois isso melhora o balanceamento mas pode criar subdomínios em excesso e aumentar a comunicação entre processos.

5.6 Estimativa de Carga

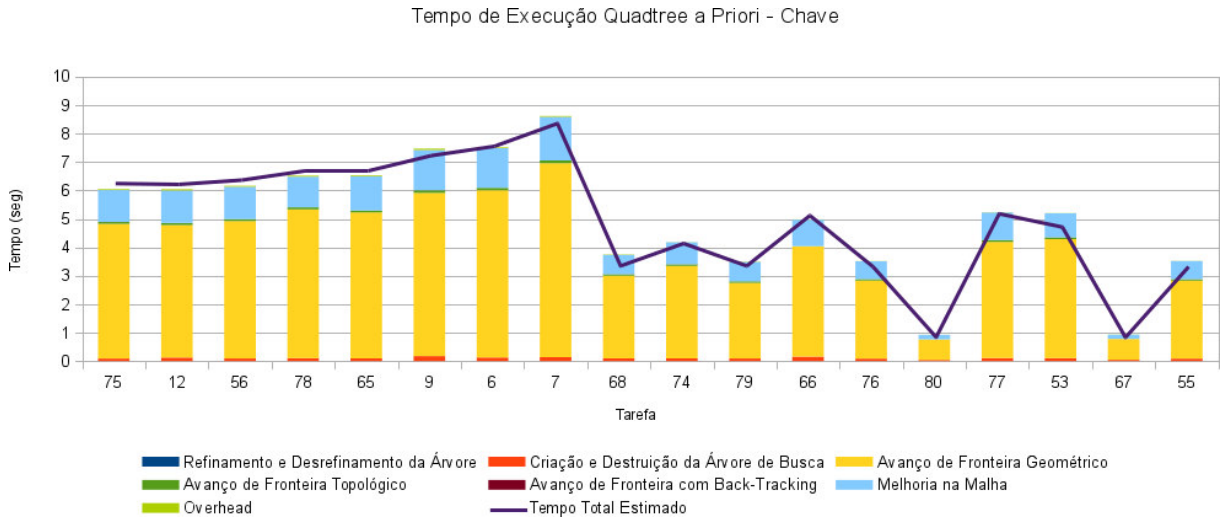
Com o objetivo de provar que a estimativa de carga descrita na Seção 4.2 é acurada, o tempo de execução levado para gerar a malha em cada subdomínio foi calculado e comparado com a carga estimada para aquele subdomínio. A Figura 5.17 compara o tempo de execução de cada passo do procedimento de avanço de fronteira e o tempo de execução estimado para alguns subdomínios. É exibido somente a estimativa para alguns processos pois com essa quantidade já é possível se ter uma ideia do comportamento da técnica. Representar todos os subdomínios não caberia no gráfico, além de não ser necessário para esta comparação.



(a) Estimativa de tempo de execução para o Cilindro com particionamento feito por *quadtree*.

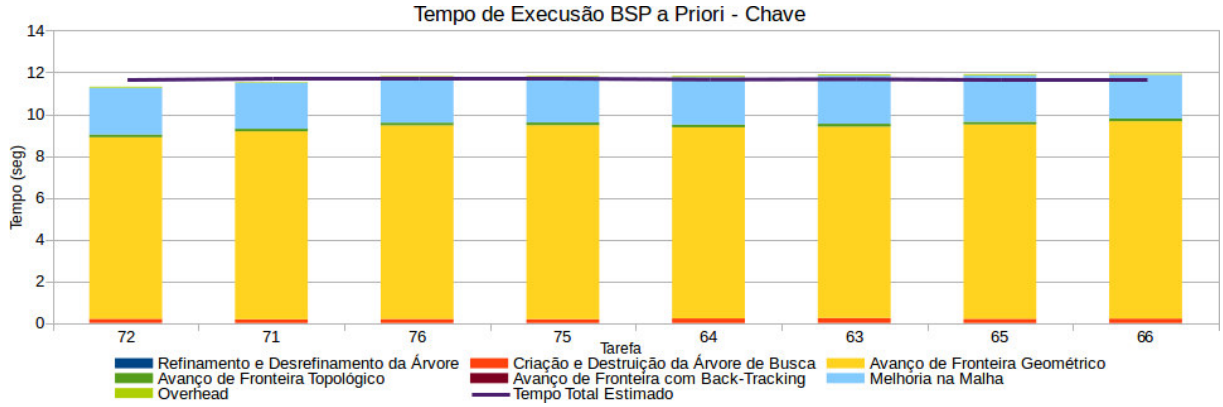


(b) Estimativa de tempo de execução para o Cilindro com particionamento feito por BSP.

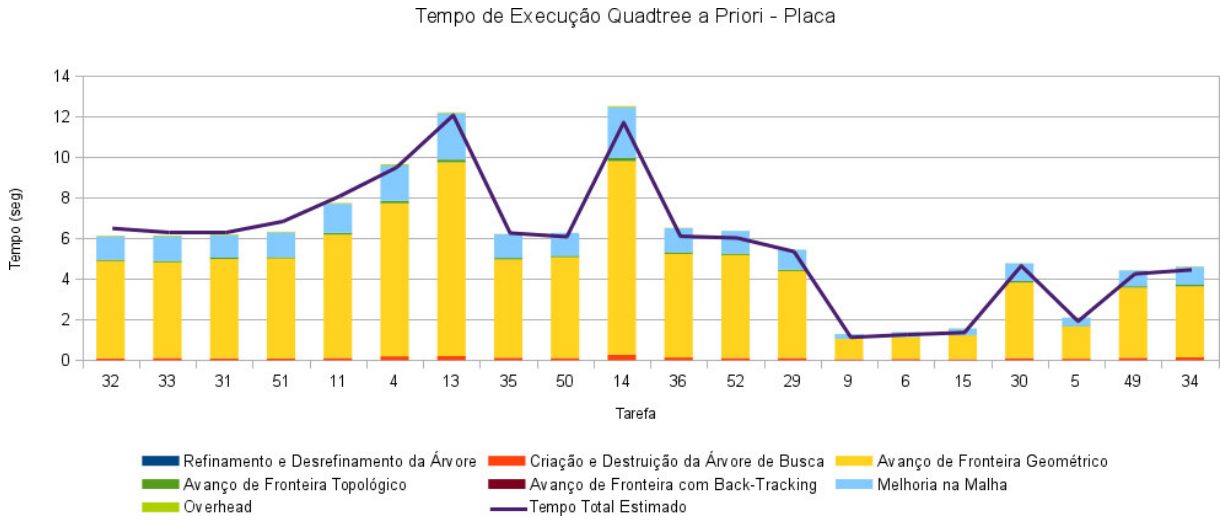


(c) Estimativa de tempo de execução para a Chave com particionamento feito por *quadtree*.

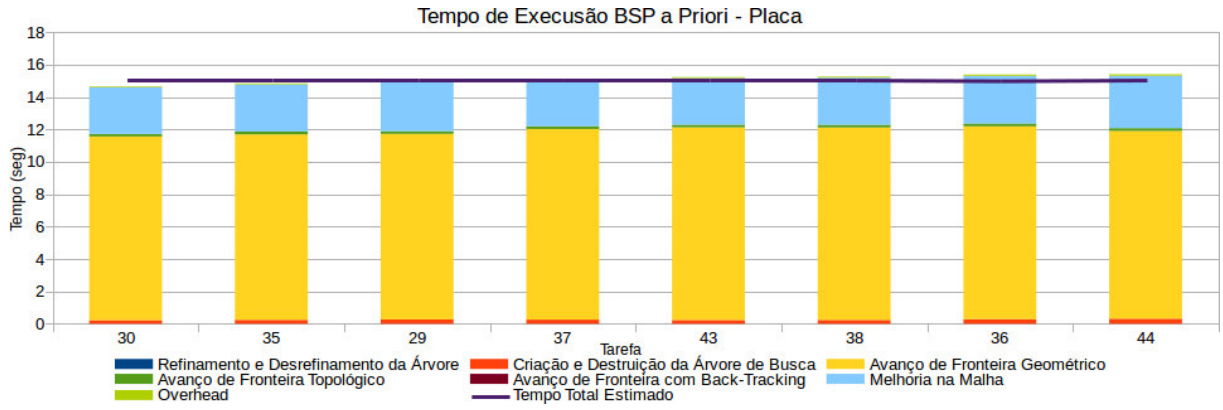
Figura 5.17: Estimativas de tempo de execução dos modelos com particionamento feito por *quadtree* e BSP.



(d) Estimativa de tempo de execução para a Chave com particionamento feito por BSP.



(e) Estimativa de tempo de execução para a Placa com particionamento feito por *quadtree*.

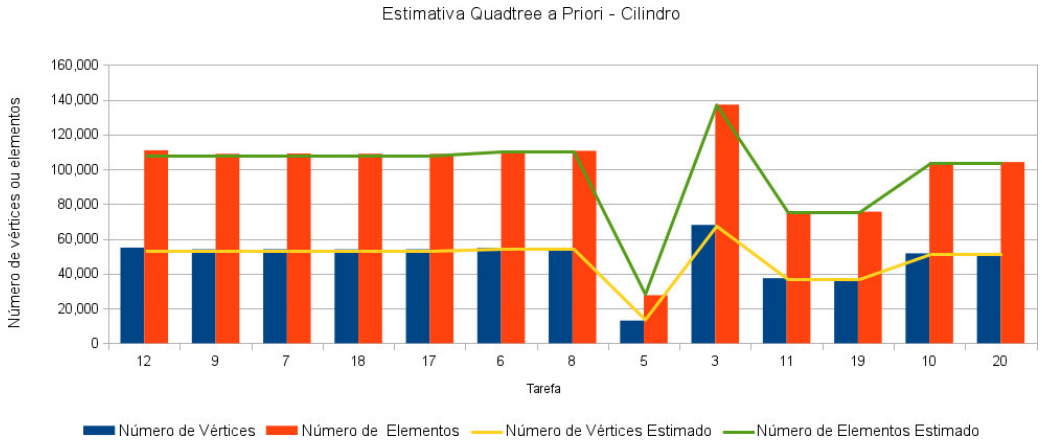


(f) Estimativa de tempo de execução para a Placa com particionamento feito por BSP.

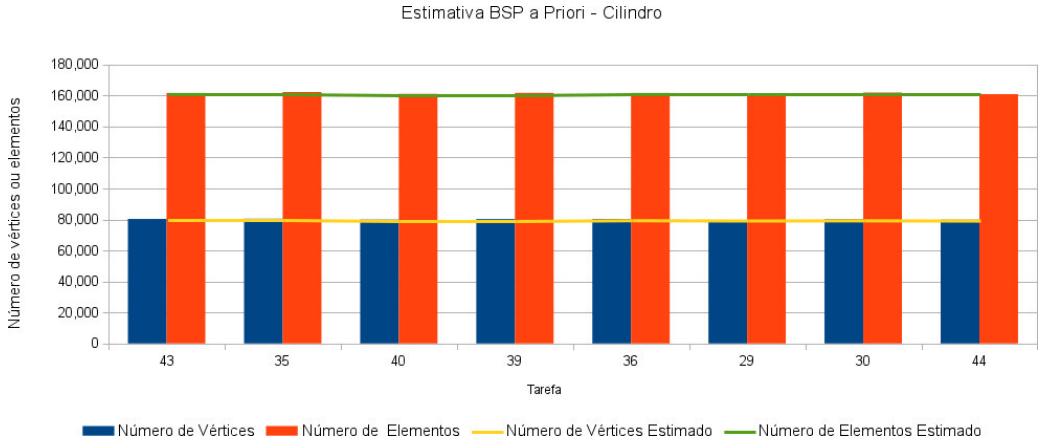
Figura 5.17: Estimativas de tempo de execução dos modelos com particionamento feito por *quadtree* e BSP (continuação).

Os passos de avanço de fronteira e de melhoria da malha pertencem ao procedimento de geração de malha empregado, e o passo de classificação das células que foi descrito na Seção 4.6 está incluso no passo criação e destruição da árvore de busca. O passo de *Overhead* contabiliza o tempo extra gasto fazendo trabalho improdutivo como busca em listas.

Quando a *quadtree* é utilizada na partição de domínio, a quantidade de subdomínios gerados é muito grande. Isso faz com que a carga associada a alguns subdomínios seja muito baixa (vide Figuras 5.17c e 5.17e). No entanto, quando BSP é usada na partição de domínio, a quantidade de tarefas geradas é igual ao número de processos.

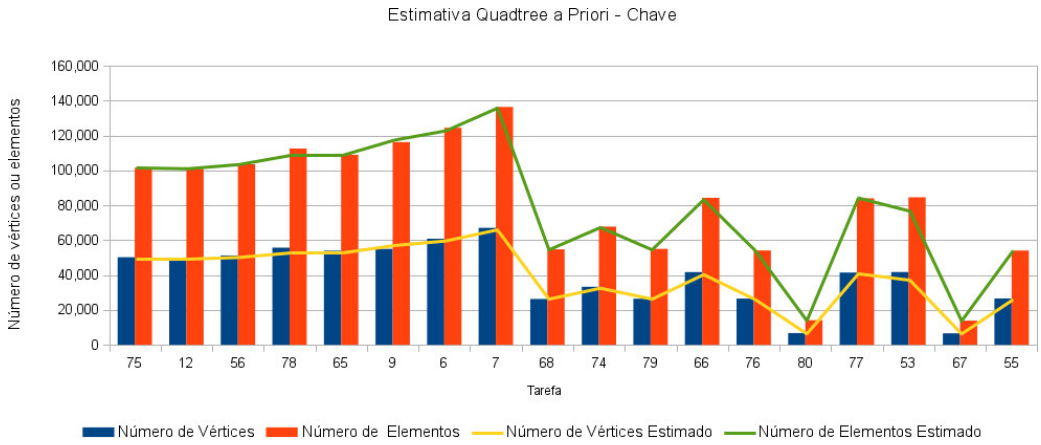


(a) Estimativa de elementos e vértices para o Cilindro com particionamento feito por *quadtree*.

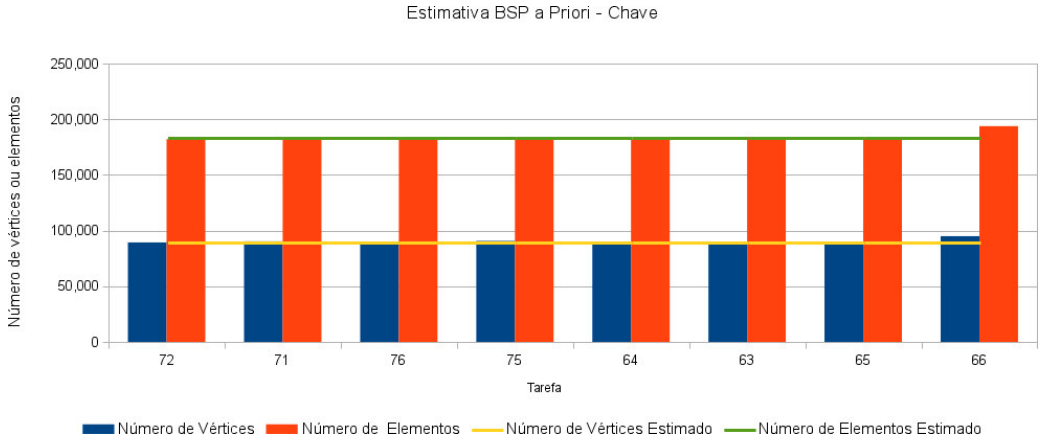


(b) Estimativa de elementos e vértices para o Cilindro com particionamento feito por BSP.

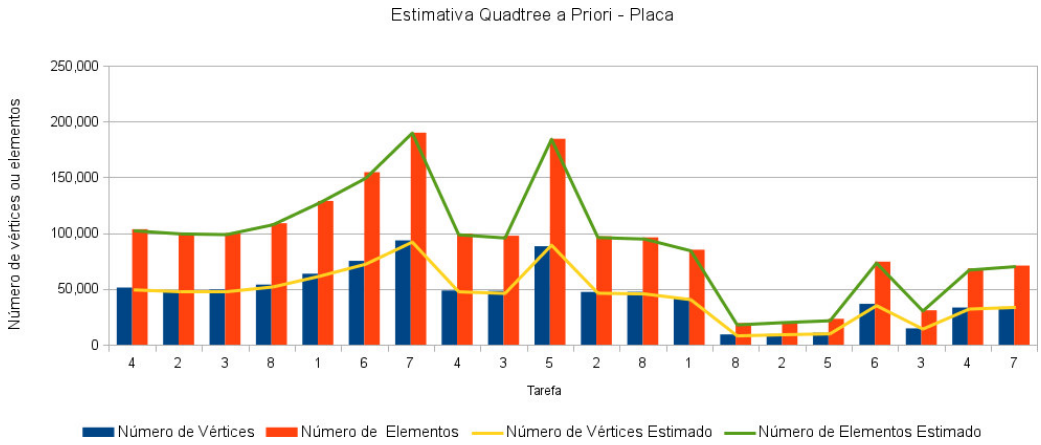
Figura 5.18: Estimativas de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP.



(c) Estimativa de elementos e vértices para o Chave com particionamento feito por *quadtree*.



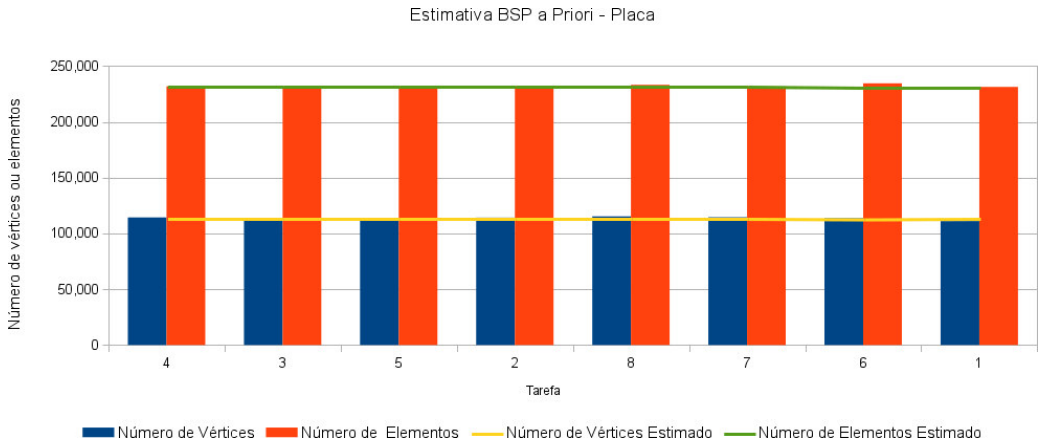
(d) Estimativa de elementos e vértices para a Chave com particionamento feito por BSP.



(e) Estimativa de elementos e vértices para a Placa com particionamento feito por *quadtree*.

Figura 5.18: Estimativas de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP (continuação).

As linhas na Figura 5.17 representam o tempo de execução estimado, que foi calculado como a carga multiplicada pelo fator $\bar{t}/\bar{l}$, onde $\bar{t}$ é o tempo médio de execução e $\bar{l}$ é a carga média estimada para os subdomínios. Já que somente a proporção entre as cargas é necessária para o balanceamento de carga, este fator fixo pode ser aplicado para propósito de avaliação. A Figura 5.18 mostra as mesmas comparações, mas considerando o número de elementos e o



(f) Estimativa de elementos e vértices para a Placa com particionamento feito por BSP.

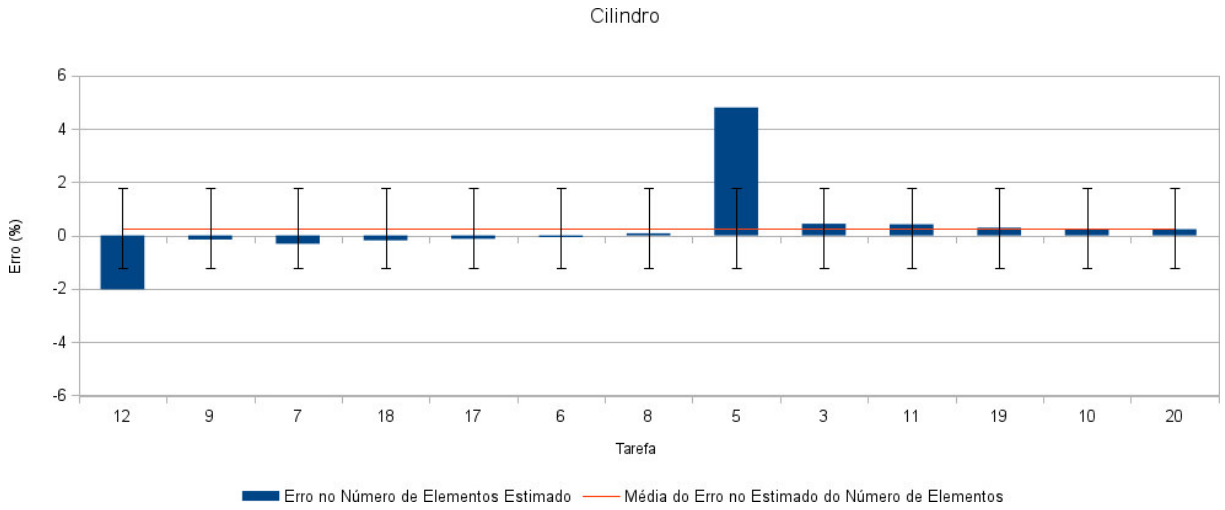
Figura 5.18: Estimativas de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP (continuação).

número de vértices em vez do tempo de execução. As linhas mostram a estimativa, enquanto as barras mostram a quantidade real gerada; como pode ser visto, a estimativa é precisa.

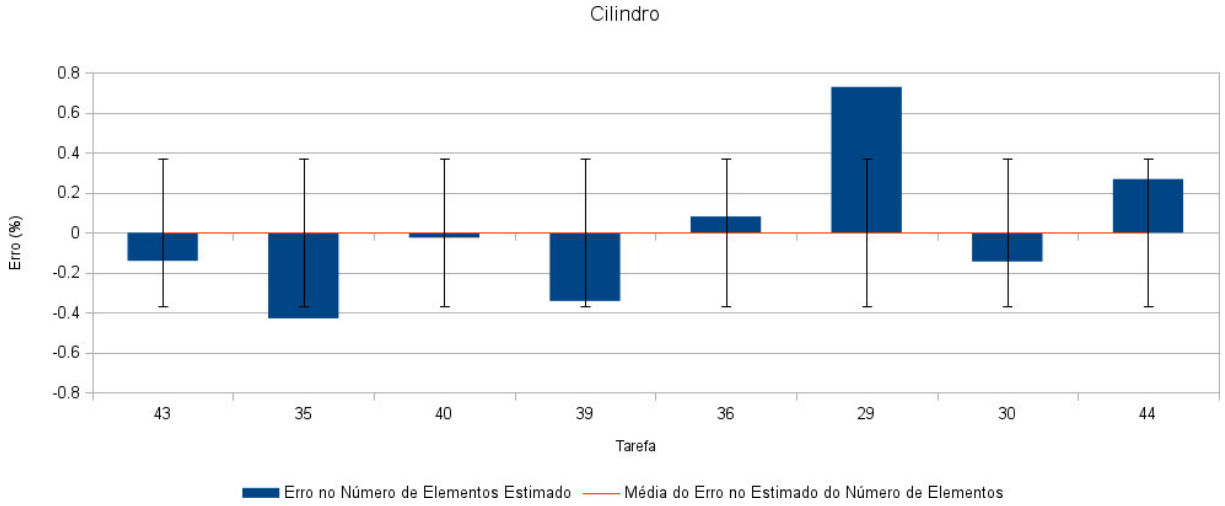
Vale lembrar que a estimativa da carga de um subdomínio é um valor que não tem um significado concreto mas que, quando comparado à estimativa de um outro subdomínio, ganha significado. Se a estimativa de carga para um subdomínio for 100 não necessariamente quer dizer que 100 elementos serão gerados mas, quando comparado à uma carga duas vezes maior de outro subdomínio, o valor de 200 quer dizer que este subdomínio gerará o dobro de elementos do outro subdomínio.

Os gráficos da Figura 5.19 mostram o erro na quantidade de elementos gerados. Quando a barra azul do erro no número de elementos estimados está acima de zero significa que foram gerados mais elementos que o estimado, e quando está abaixo de zero significa que foram gerados menos elementos que o esperado. Nos exemplos do Cilindro e Placa (Figuras 5.19b e 5.19f) com particionamento feito por BSP, o erro médio se manteve bastante baixo; já no caso da Chave o erro é bem maior se comparado com os anteriores e varia em torno de 2%. Isso se deve ao fato da geometria da Chave ser bastante recortada. O mesmo fato acontece quanto o modelo é particionado com uma *quadtree*, mas utilizando-se a BSP o erro médio na estimativa é bem menor.

Os resultados evidenciam que são criados subdomínios carga e tempo de execução sempre nivelados. Até mesmo nos casos em que o modelo tem uma geometria complexa a estimativa se manteve boa, graças à adaptatividade que a BSP possui. Isso não acontece quando uma *quadtree* de particionamento é empregada, como é possível ver nas Figuras 5.18 e 5.17.

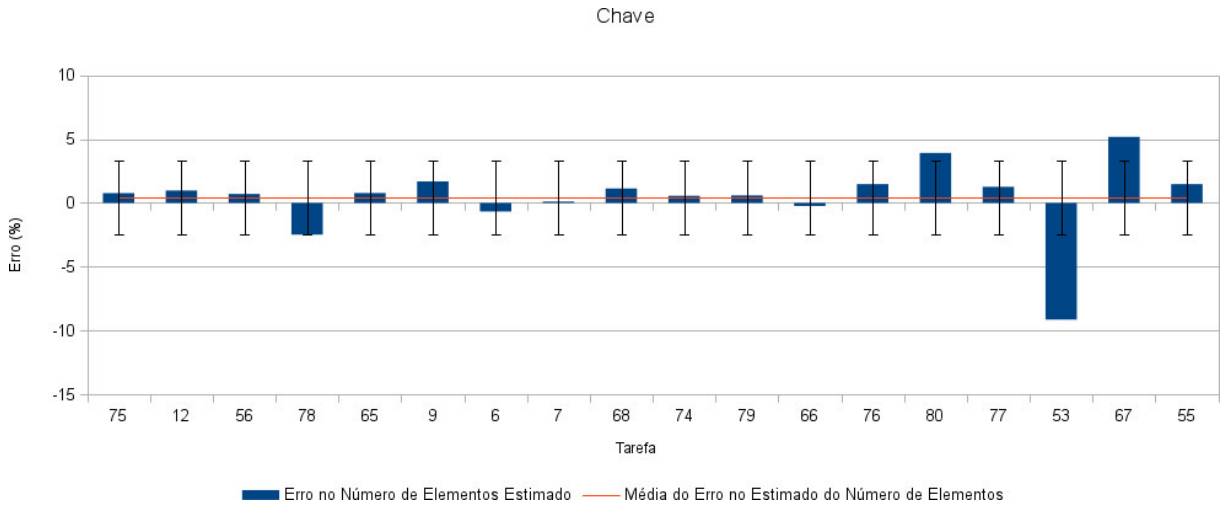


(a) Erro na estimativa de elementos e vértices para o Cilindro com particionamento feito por *quadtree*.

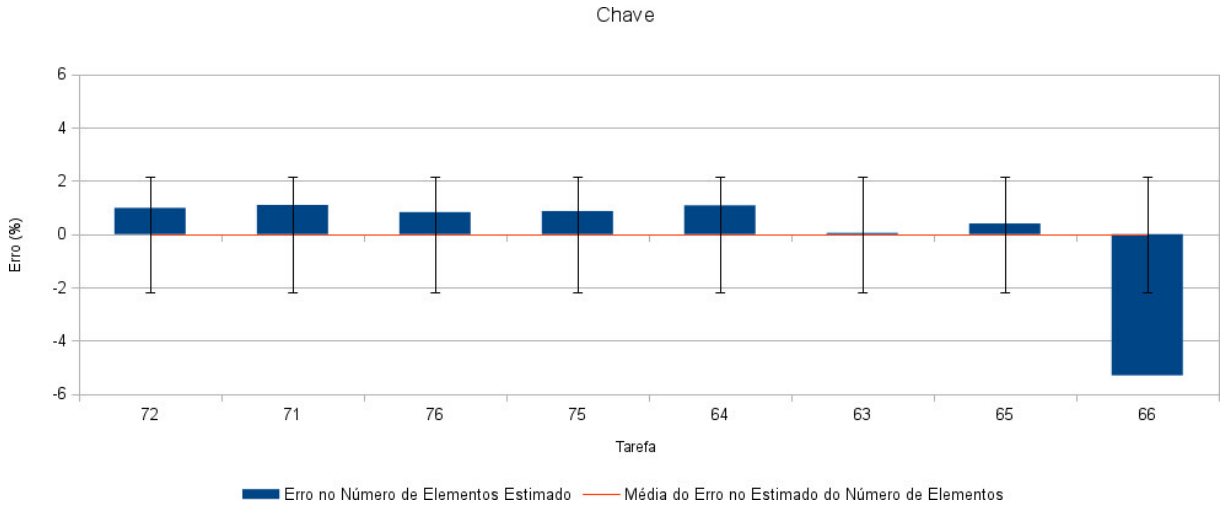


(b) Erro na estimativa de elementos e vértices para o Cilindro com particionamento feito por *quadtree*.

Figura 5.19: Erro na estimativa de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP.

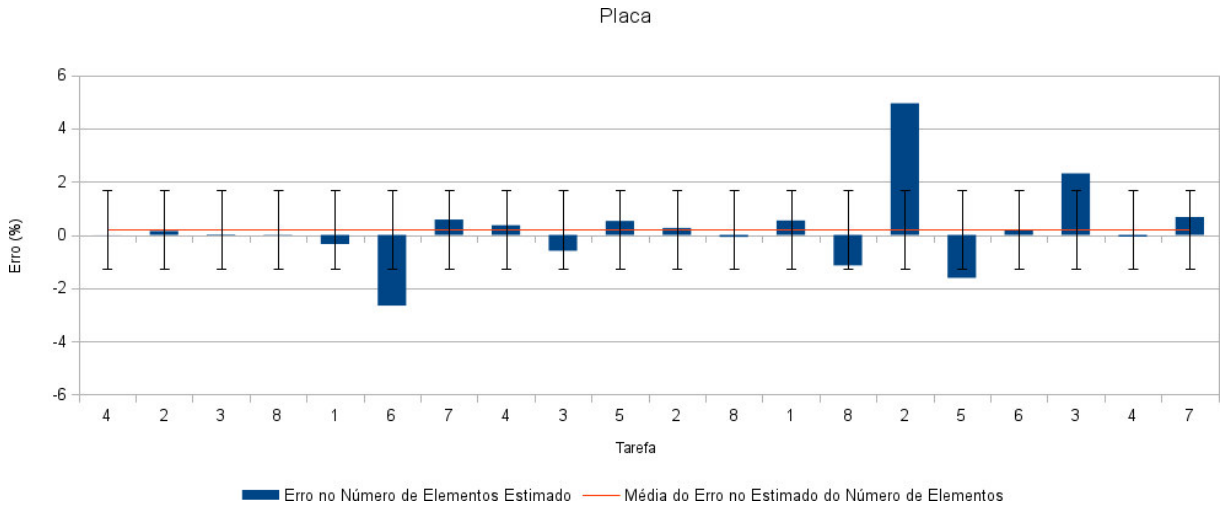


(c) Erro na estimativa de elementos e vértices para a Chave com particionamento feito por *quadtree*.

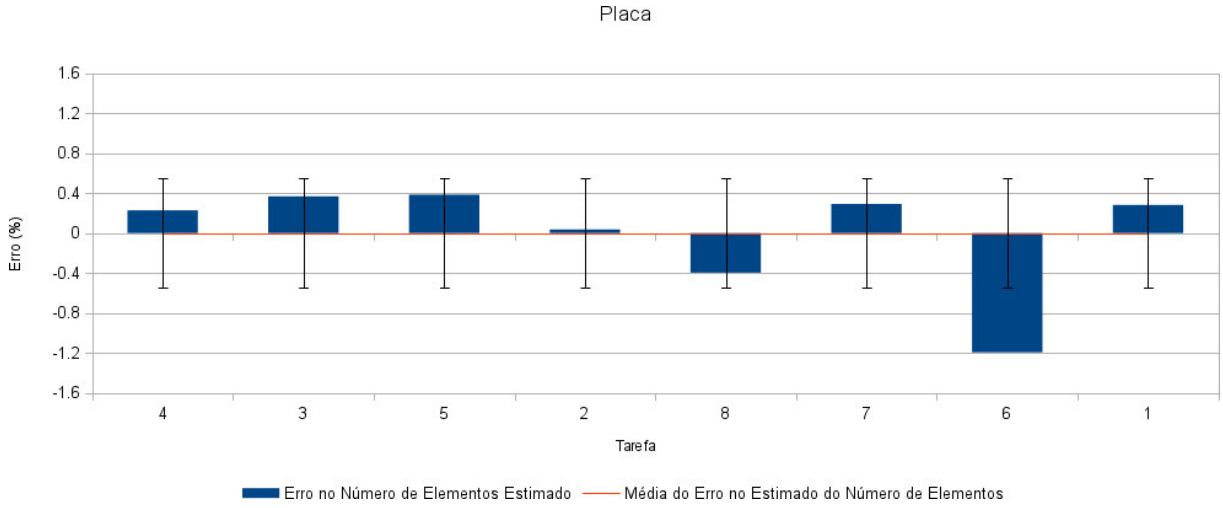


(d) Erro na estimativa de elementos e vértices para a Chave com particionamento feito por *quadtree*.

Figura 5.19: Erro na estimativa de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP (continuação).



(e) Erro na estimativa de elementos e vértices para a Placa com particionamento feito por *quadtree*.



(f) Erro na estimativa de elementos e vértices para a Placa com particionamento feito por *quadtree*.

Figura 5.19: Erro na estimativa de elementos e vértices dos modelos com particionamento feito por *quadtree* e BSP (continuação).

Os dados exibidos nesta seção mostram que o método empregado, baseado em uma *quadtree* para estimar a carga, produz uma estimativa de boa precisão para o número de elementos e de vértices. Além disso, foram apresentados os resultados de particionamentos feitos por *quadtree* e BSP e suas diferenças.

5.7 Considerações Finais

Neste capítulo foram apresentados alguns resultados da técnica proposta. Além disso, foram mostradas comparações da técnica proposta, que usa uma abordagem *a priori*, com uma outra técnica de decomposição (FREITAS et al., 2013) que é baseada em uma decomposição *a posteriori*.

A técnica proposta neste trabalho consegue gerar uma malha de qualidade, com níveis bem próximos da malha gerada de forma sequencial. Com relação à comparação entre as duas técnicas (*a priori* e *a posteriori*), é possível constatar que a técnica *a posteriori* possui uma qualidade mais próxima à da malha gerada sequencialmente, tendo uma diferença menor que a técnica descrita neste trabalho. Esse fato já era esperado, pois abordagens contínuas de particionamento *a posteriori* não criam fronteiras limitando a geração da malha naquelas regiões onde foi particionado, fato que ocorre em abordagens contínuas *a priori*. Isso, entretanto, não é um problema sério pois a qualidade da malha não varia muito como foi mostrado e esse aspecto ocorre, em geral, em regiões internas do modelo, onde isso não é crucial na maioria dos casos.

Com relação à questão de velocidade a técnica apresentada neste trabalho apresenta um *speed-up* próximo do linear e é melhor quando comparada com a do trabalho de (FREITAS et al., 2013). Além de apresentar os resultados do particionamento feito por *quadtree*, também foram mostrados os resultados usando BSP. Comparando os resultados obtidos pelas duas estruturas, concluiu-se que a BSP é melhor que a *quadtree*, pois a qualidade da malha e a velocidade de geração da malha quando uma BSP é usada no particionamento do domínio são bem melhores.

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Principais Contribuições

Esse trabalho apresentou uma técnica *a priori* de subdivisão de domínios bidimensionais para a geração paralela da malha. A técnica proposta é dita *a priori* porque a malha de interface entre os subdomínios é gerada antes das suas malhas internas. Essa técnica gera novos domínios e permite abstrair a técnica de geração de malha aplicada aos subdomínios, podendo-se combinar, por exemplo, as técnicas de Delaunay e Avanço de Fronteira, dentre outras.

Ao final desse trabalho foram apresentados os resultados obtidos pela técnica proposta. Foi também mostrada a diferença nos resultados que as estruturas de dados utilizadas para particionar fazem, comparando a *quadtree* com a BSP. Os resultados foram comparados com os resultados para os mesmos modelos no trabalho de (FREITAS et al., 2013) que usa uma estratégia *a posteriori*, pois essa técnica segue uma abordagem de particionamento diferente.

Uma desvantagem de utilizar uma *quadtree* para particionar domínios é que os domínios criados nem sempre mantém a mesma proporção entre as cargas. Se for preciso subdividir um subdomínio, quatro novos serão criados e isso faz com que o balanceamento da carga fique prejudicado pelo excesso de partições criadas reduzindo assim o *speed-up*. Utilizando BSP para particionamento, a quantidade de subdomínios criados é a mesma quantidade de processos disponíveis, evitando o excesso de tarefas e de comunicação.

A técnica aborda a subdivisão dos domínios e não a geração de malha propriamente dita. Apesar de qualquer algoritmo poder ser utilizado, em particular, um algoritmo de avanço de fronteira é utilizado para gerar malha e está descrito em (MIRANDA; CAVALCANTE-NETO; MARTHA, 1999) e (CAVALCANTE-NETO et al., 2001).

O trabalho proposto pode ser dividido basicamente em duas etapas. A primeira consiste em receber a lista de segmentos que compõem a fronteira de entrada e dividir o domínio definido por ela em partes que possam ser tratadas por novos domínios independentes. Todos essas partes, por sua vez, possuem a quantidade de carga aproximadamente equivalentes. A segunda etapa é a geração da malha em cada um dos novos domínios. Qualquer algoritmo de triangulação dado uma borda como entrada pode ser utilizado, podendo, inclusive, serem empregados algoritmos diferentes nos subdomínios. Além da triangulação, é necessário fazer a junção das diversas malhas geradas em uma só.

As principais contribuições deste trabalho são a apresentação de uma nova técnica de decomposição de domínios que apresentou bons resultados e a comparação da utilização da *quadtree* com a *binary space partitioning* (BSP) para auxiliar o particionamento, mostrando ao final que a BSP possui resultados melhores.

6.2 Trabalhos Futuros

A técnica aqui apresentada permite a utilização de múltiplas técnicas de geração de malha, podendo inclusive combinar-se diferentes algoritmos ao mesmo tempo. Alguns testes preliminares foram feitos usando o algoritmo de Delaunay de (SHEWCHUK, 1996) mas é necessário um estudo mais aprofundado. Viabilizando a perfeita execução do algoritmo de Delaunay será possível gerar malhas mistas e até mesmo comparar com o algoritmo de Avanço de Fronteira.

Este trabalho utiliza o modelo de paralelismo mestre/escravo, que é normalmente utilizado em aplicações que possuem mais tarefas que processos disponíveis. Quando se particiona o domínio por *quadtree*, mais tarefas que processos são criados em geral, mas ao utilizar BSP para particionar, a quantidade de subdomínios será a mesma de processos. A estimativa de carga é bem precisa, fazendo os tempos de execução dos subdomínios criados pela BSP serem praticamente iguais. Isso cria um gargalo no processo mestre, pois ele irá receber praticamente todos os resultados da geração de malha dos subdomínios ao mesmo tempo.

Uma solução para este problema é mudar o modelo de paralelismo fazendo com que cada processo seja responsável por gerar sua fronteira de interface e malha; a junção dos pedaços da malha é feita sempre em pares de processos que possuem subdomínios vizinhos, fazendo estas junções até que toda malha esteja completa. Mudar a forma de distribuição das tarefas vai melhorar bastante os resultados de *speed-up* quando utilizar o particionamento baseado na BSP.

A geração de malhas em domínios com fraturas não foi testada, mas espera-se fazer uma versão deste trabalho que suporte modelos com fraturas existentes na fronteira de entrada. Nessas malhas, a superfície da fratura é representada como uma região de área nula formada por segmentos de borda geometricamente coincidentes, cujas normais têm sentidos opostos. Fraturas são muito importantes em vários tipos de simulações computacionais de problemas de engenharia.

Por fim, uma extensão natural é implementar uma versão tridimensional da técnica. Técnicas de decomposição contínua *a priori* em geral não são facilmente implementadas pois precisam gerar novas fronteiras, criando assim segmentos de interface. Essa criação torna-se complicada pois para se obter uma decomposição totalmente automática é necessário que a técnica seja suficientemente robusta para procurar os melhores pontos de corte, levando em consideração a carga e a qualidade dos elementos que serão gerados ali posteriormente. Na versão tridimensional será necessário criar superfícies que se adaptem à geometria dos modelos de entrada, e isso não é algo trivial, necessitando de estudos mais aprofundados.

APÊNDICE A – GEOMETRIAS DOS EXEMPLOS

A.1 Introdução

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

A.2 Cilindro

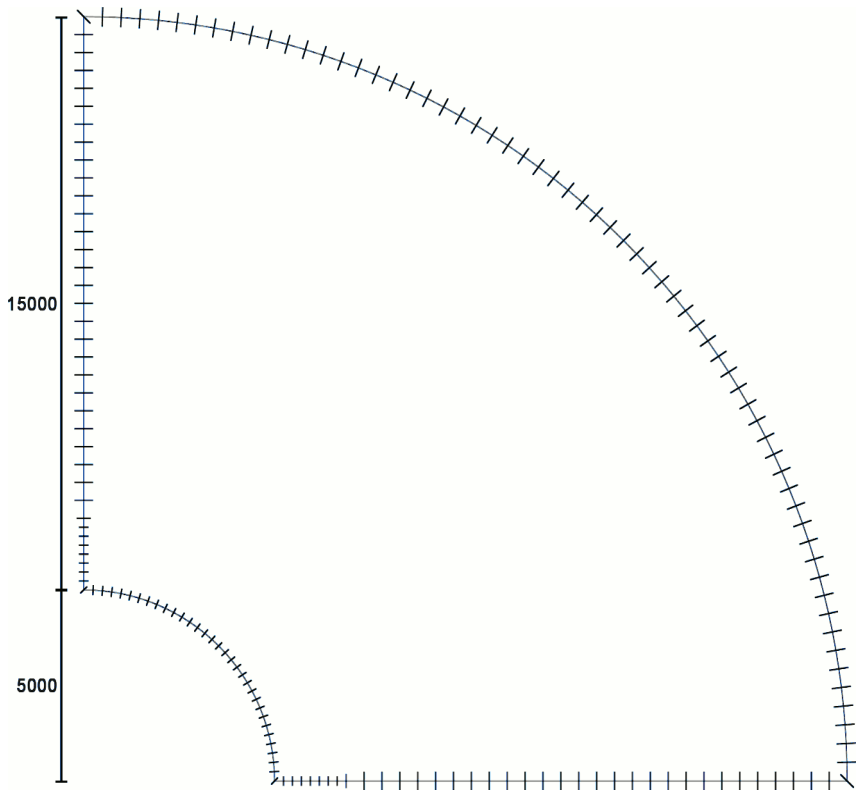


Figura A.1: Geometria do exemplo do Cilindro.

A.3 Chave

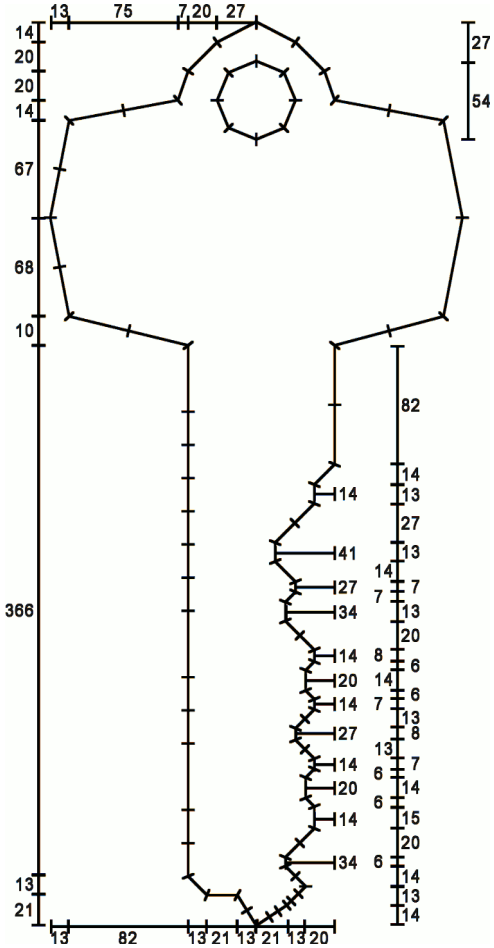


Figura A.2: Geometria do exemplo da Chave.

A.4 Placa

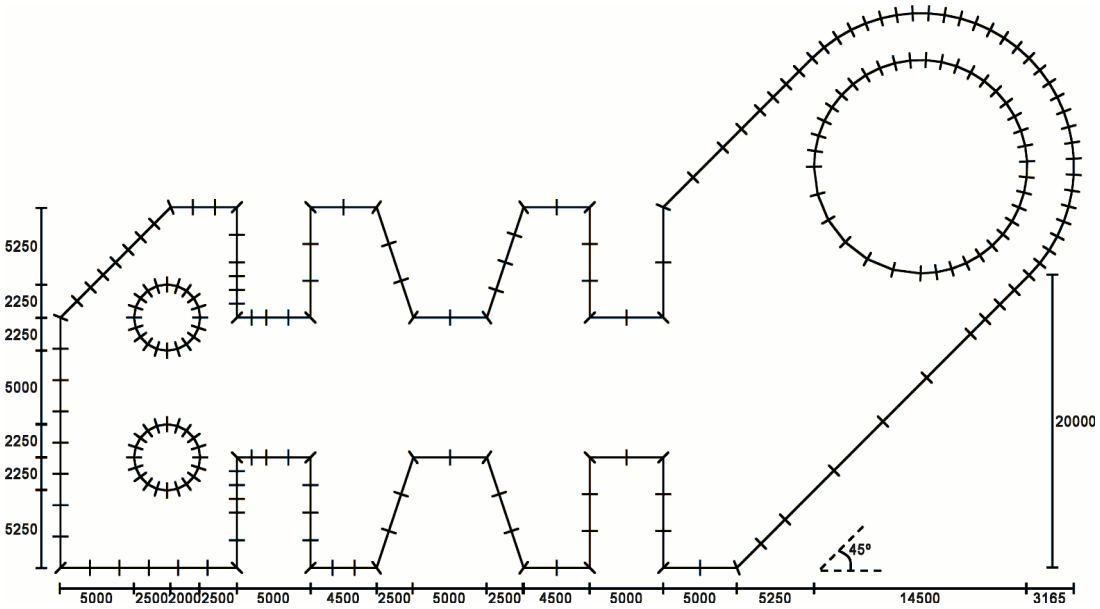


Figura A.3: Geometria do exemplo da Placa.

REFERÊNCIAS BIBLIOGRÁFICAS

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

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

BARKER, K. et al. 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, M. *Principles of Concurrent and Distributed Programming*. 2nd. ed. [S.l.]: Addison-Wesley, 2006.

BERG, M. de et al. *Computational Geometry: Algorithms and Applications*. 2nd. ed. [S.l.]: Springer-Verlag, 2000.

BISWAS, R. et al. Parallel dynamic load balancing strategies for adaptive irregular applications. *Applied Mathematical Modelling*, v. 25, n. 2, p. 109–122, 2000.

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, 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, 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, J. B. et al. A back-tracking procedure for optimization of simplex meshes. *Communications in Numerical Methods in Engineering*, v. 21, n. 12, p. 711–722, 2005. ISSN 1069-8299.

CAVALCANTE-NETO, J. B. et al. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, v. 17, n. 1, p. 75–91, 2001.

CHERNIKOV, A.; CHRISOCHOIDES, N. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In: *Proceedings of the 14th International Meshing Roundtable*. [S.l.]: Sandia National Laboratory, 2005.

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

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, 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, 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.

DæHLEN, M.; HJELLE Øyvind. *Triangulations and Applications*. [S.l.]: Springer, 2006.

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

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

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

FREITAS, M. de O.; CAVALCANTE-NETO, J. B.; VIDAL, C. A. *Parallel generation of meshes with cracks using a binary espacial decomposision*. [S.l.], 2014.

FREITAS, M. O. *Geração em Paralelo de Malhas Triangulares por Avanço de Fronteira com Particionamento por Decomposição Espacial Recursiva*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO CEARÁ, 2010.

FREITAS, M. O. et al. A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. *Advances in Engineering Software*, v. 59, p. 38–52, 2013.

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

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

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

GLUT, B.; JURCZYK, T. Domain decomposition techniques for parallel generation of tetrahedral meshes. *Springer-Verlag Berlin Heidelberg*, p. 641–650, 2008.

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

GRAMA, A. et al. *Introduction to Parallel Computing*. 2nd. ed. [S.l.]: Addison Wesley, 2003.

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

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

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, Y. et al. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, v. 75, n. 5-6, p. 200–209, 2007.

IVANOV, E.; ANDRÄ, H.; KUDRYAVTSEV, A. N. Domain decomposition approach for automatic parallel generation of tetrahedral grids. *Computational Methods in Applied Mathematics*, v. 6, n. 2, p. 178–193, 2006.

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.

KARYPIS, G.; KUMAR, V. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*. [S.l.], September 1998.

KARYPIS, G.; SCHLOEGEL, K.; KUMAR, V. *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.2*. [S.l.], April 2011.

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

LARWOOD, B. G. et al. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering*, v. 58, p. 177–188, 2003.

LAWLOR, O. S. et al. ParFUM: A parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, v. 22, n. 3-4, p. 215–235, 2006.

LINARDAKIS, L.; CHRISOCHOIDES, N. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing*, v. 27, n. 4, p. 1394–1423, 2006.

LO, S. Parallel delaunay triangulation — application to two dimensions. *Finite Elements in Analysis and Design*, p. 37–48, 2012.

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

LÖHNER, R.; PARIKH, P. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, v. 8, p. 1135–1149, 1988.

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, 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, A. C. de O. et al. Surface mesh regeneration considering curvatures. *Engineering with Computers*, Springer-Verlag London Limited, v. 25, n. 2, p. 207–219, 2009. ISSN 0177-0667.

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*. [S.l.]: IEEE Computer Society, 1999. p. 29–38.

MIRANDA, A. C. O. et al. Fatigue life and crack path predictions in generic 2D structural components. *Engineering Fracture Mechanics*, v. 70, p. 1259–1279, 2003.

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

OKUSANYA, T.; PERAIRE, J. Parallel unstructured mesh generation. In: *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*. [S.l.]: Mississippi State University, 1996. p. 719–729.

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. The OpenMP API specification for parallel programming. Disponível em: <http://www.openmp.org>.

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

PÉBAY, P. P. et al. pCAMAL: An embarrassingly parallel hexahedral mesh generator. In: *Proceedings of the 16th International Meshing Roundtable*. Washington, United States: Sandia National Laboratory, 2007.

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

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.

PIRZADEH, S. Z.; ZAGARIS, G. Domain decomposition by the advancing-partition method for parallel unstructured grid generation. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. [S.l.]: AIAA - American Institute of Aeronautics and Astronautics, 2009.

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

RIVARA, M. C. et al. Parallel decoupled terminal-edge bisection method for 3D mesh generation. *Engineering with Computers*, v. 22, p. 111–119, 2006.

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, United States: SIAM - Society for Industrial and Applied Mathematics, 1999. p. 83–62.

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

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

SHEWCHUK, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: LIN, M. C.; MANOCHA, D. (Ed.). *Applied Computational Geometry: Towards Geometric Engineering*. [S.l.]: Springer-Verlag, 1996, (Lecture Notes in Computer Science, v. 1148). p. 203–222. From the First ACM Workshop on Applied Computational Geometry.

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

SHREINER, D. et al. *OpenGL Programming Guide*. 6th. ed. [S.l.]: Addison-Wesley, 2007.

SPEAR, A. D. et al. Structural assessment and prognosis using a multi-scale, fracture mechanics-based approach. In: *Proceedings of the 2010 Aircraft Airworthiness and Sustainment Conference*. Texas, United States: [s.n.], 2010.

TENG, Y. A. et al. A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation. In: *Proceedings of the 1993 Conference on High Performance Network and Computing*. Oregon, United States: Association for Computing Machinery, 1993. p. 112–121.

WAWRZYNEK, P. A.; INGRAFFEA, A. R. *FRANC2D - A Two-Dimensional Crack Propagation Simulator - User's guide - Version 3.1*. [S.l.], 1993.

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

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, 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*. [S.l.]: AIAA - American Institute of Aeronautics and Astronautics, 2009.