Malhas adaptativas em domínios definidos por fronteiras curvas

Luis Gustavo Pinheiro Machado

Malhas adaptativas em domínios definidos por fronteiras curvas

Luis Gustavo Pinheiro Machado

Orientador: ***Prof. Dr. Luis Gustavo Nonato***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional.

USP - São Carlos
Janeiro/2008

Resumo

Dois métodos distintos são descritos e implementados. O primeiro método, proposto por Ruppert [25], possui garantias teóricas de qualidade quando a fronteira do domínio obedece certas restrições.

O segundo método, proposto por Persson [23] possibilita um maior controle na densidade dos elementos que discretizam o domínio. As vantagens, desvantagens e particularidades de cada um dos métodos são descritas e detalhadas.

Palavras-chave: Geração de malhas, Delaunay, refinamento, malhas adaptativas, geometria computacional.

Abstract

Two distinct methods are described and implemented. The first method, proposed by Ruppert [25], has theoretical guarantees on the quality of elements when the domain boundaries respect certain restrictions.

The second method, proposed by Persson [23] makes it possible to have greater control over the density of the elements that make up the domain. The advantages, disadvantages and specific points about each method are described and detailed.

Keywords: Mesh generation, Delaunay, Refinement, adaptive meshes, computational geometry.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

Técnicas de simulações numéricas têm apresentado melhorias significativas em termos de desempenho e qualidade, viabilizando simulações mais complexas como, por exemplo, o escoamento de fluidos, túneis de vento, simulações de resistência em aplicações de engenharia civil e outras simulações envolvendo operações numéricas intensas. Como exemplos práticos de tais tipos de simulação podemos citar os seguintes:

- Injeção de fluidos em moldes para a fabricação de peças variadas.
- Simulação de enchimento de represas em usinas hidroelétricas.
- Simulação de resistência de uma determinada estrutura como uma ponte, prédio ou asa de um avião.

O primeiro exemplo engloba a realidade de várias indústrias que visam a produção de encanamentos, peças plásticas para veículos ou peças para um determinado motor em um processo de fundição.

A simulação de enchimento de represas é de especial interesse visto que interfere gravemente no meio ambiente e a devastação de um local não apropriado para tal fim pode vir a comprometer o ecossistema local e também moradores dos arredores. Esta dissertação possui objetivos relacionados a este fim como será visto adiante.

Simulações de resistência são bastante conhecidas e largamente utilizadas para garantir a solidez e segurança de edifícios, pontes e veículos de uma maneira geral.

Dada a importância das aplicações exemplificadas, consideráveis melhorias nos métodos atuais têm sido feitas, bem como o desenvolvimento de novos métodos. Os métodos de elementos finitos, diferenças finitas, volumes finitos e Lagrangeanos têm sido alguns dos mais amplamente aplicados

com o intuito de obter resultados para tais problemas. Porém, tais métodos necessitam de uma base geométrica que descreve os objetos de interesse, as chamadas malhas. Contextualizamos assim o problema de geração de malhas.

A geração de malhas visa a decomposição de um determinado domínio (uma região geométrica) em vários elementos, de modo a permitir o uso de um método numérico para aproximar a solução das equações governantes do problema em questão. Porém, aplicando-se uma forma inapropriada de decomposição ao domínio resultará em soluções numéricas imprecisas, não confiáveis e pouco úteis. Desta maneira é preciso ter atenção especial para que a decomposição de um determinado domínio seja feita da melhor forma possível, com base no tipo de aplicação desejada, caracterizando a fase de geração de malhas como uma etapa chave no contexto de simulações por métodos numéricos [30].

As malhas podem ser classificadas em duas categorias: estruturadas e não estruturadas. As malhas estruturadas apresentam uma organização topológica uniforme, porém possuem aplicação mais restrita em simulações. As malhas não estruturadas, ao contrário, têm topologia não uniforme e são mais frequentemente utilizadas em simulações numéricas devido à sua maior adaptatividade.

O tipo mais simples de malha estruturada é a malha cartesiana regular, onde os elementos são idênticos em tamanho e forma. A figura 1.1 apresenta o exemplo mais simples de malha cartesiana regular, onde todos os elementos possuem o mesmo espaçamento horizontal e vertical. Classificamos uma determinada malha como topologicamente estruturada se esta apresenta topologia isomorfa à malha cartesiana regular. A figura 1.2 apresenta um exemplo de malha triangular topologicamente estruturada e de uma malha não estruturada.

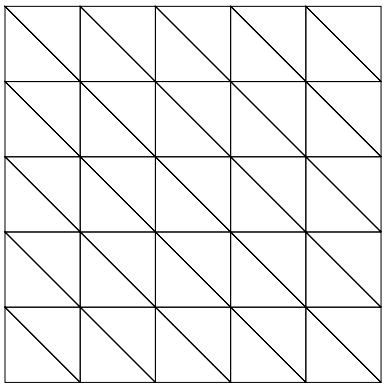


Fig. 1.1: Malha cartesiana regular

A manipulação de malhas estruturadas é mais simples, assim como as estruturas de dados necessárias para sua representação. Porém, o uso de malhas regulares pode não ser apropriado pois limitam a aplicabilidade dos métodos númericos a problemas cujos domínios são geometricamente simples ou parametrizáveis. Em problemas representados por domínios complexos ou com soluções que variam rapidamente sobre o domínio, é mais apropriado o uso de malhas não-estruturadas, visto

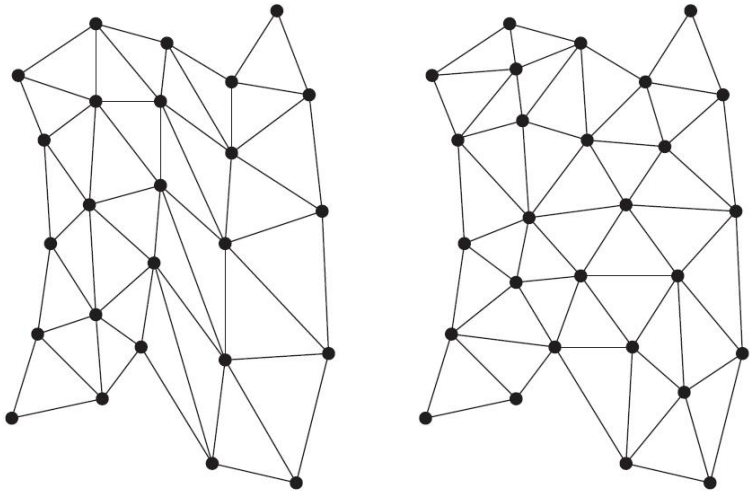


Fig. 1.2: Malha triangular topologicamente estruturada (esquerda) e não estruturada (direita)

que permitem ajustar a topologia e outros parâmetros facilmente em diferentes regiões do domínio, atribuindo adaptatividade à malha.

Como exemplo podemos citar a modelagem de um problema de cálculo de deformação em uma determinada estrutura. Considerando que será aplicada uma força apenas e um dos lados da estrutura, é necessária uma discretização mais fina nesta região, permitindo que a deformação seja calculada com maior precisão. A região central e os outros lados do domínio, onde não há força direta aplicada, serão pouco afetados, podendo ser modelados com poucos elementos.

O número de elementos pode afetar tanto a qualidade quanto o desempenho de uma simulação. Em geral, quanto maior o número de elementos utilizados durante o particionamento de um domínio, melhor a precisão e confiabilidade da solução. Porém, com o aumento do número de elementos, eleva-se o tempo para a obtenção de um resultado devido ao maior número de cálculos envolvidos. Com um número pequeno de elementos temos uma solução mais rápida, porém possivelmente menos confiável. Com a adaptabilidade das malhas não estruturadas, é possível obter bons resultados com um tempo de simulação aceitável, discretizando-se com um maior número de elementos as regiões críticas e usando menos elementos para as regiões de pouco interesse. Um exemplo de um domínio discretizado de modo não uniforme e adaptativo pode ser visto na figura 1.3.

Além da densidade e do tamanho dos elementos em uma malha, uma outra importante característica a se considerar é a forma dos elementos. De um modo geral, os elementos devem ser mais próximos de um triângulo equilátero quanto possível, assegurando uma boa base para a operação dos métodos números. Em contrapartida, triângulos contendo ângulos maiores do que 120^o são considerados menos apropriados para a aplicações dos métodos números, visto que a precisão e confiabilidade dos resultados será comprometida. Ressaltamos que não estão sendo consideradas as possíveis

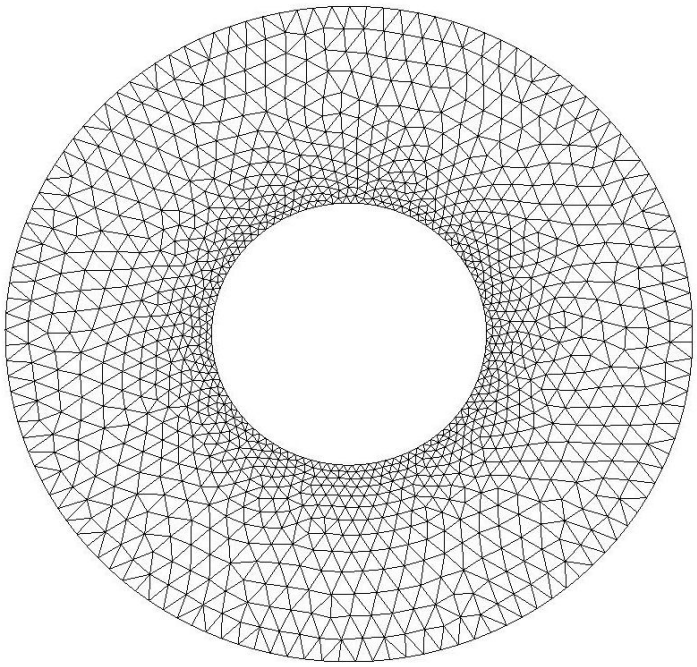


Fig. 1.3: Malha com densidade variável de pontos e elementos de tamanhos distintos

anisotropias do domínio ou da solução.

Sabe-se, portanto, que o tamanho e o formato dos elementos devem ser controlados para garantir uma determinada precisão na solução. Surge assim a necessidade da determinação das regiões do domínio que devem ser mais densas e das que devem ser menos densas. A solução para este problema reside na geometria e no problema físico modelado sobre o domínio onde se dará a simulação.

A análise do gradiente da solução é um bom exemplo. Considerando um problema cuja solução tem uma variação lenta, é possível utilizar uma malha aproximadamente uniforme onde os elementos têm aproximadamente o mesmo tamanho. Para problemas cujas soluções apresentam variações rápidas, pode ser necessária a utilização de malhas adaptativas que apresentam maior densidade de elementos nas regiões mais críticas e menor densidade em regiões de menor interesse.

De forma concreta, definimos o problema de geração de malhas, no contexto deste trabalho, da seguinte maneira:

Entrada: Domínio constituído de uma região poligonal no plano, possivelmente com buracos, vértices e arestas no interior do mesmo.

Saída: Uma triangulação do domínio poligonal cujas arestas da triangulação cobrem todas arestas de entrada e cujos vértices cobrem todos os vértices de entrada.

O grafo dos vértices e arestas de entrada é denotado por G, e a triangulação da saída é denotada por T. A Figura 1.4 ilustra a entrada e a saída para um problema particular de geração de malhas.

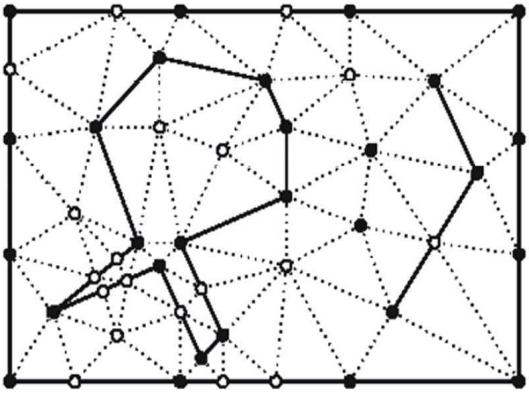


Fig. 1.4: Os vértices e arestas em destaque correspondem ao conjunto de entrada. O conjunto de todos os vértices, arestas e triângulos corresponde à triangulação de saída.

Vários métodos de geração de malhas simpliciais não estruturadas podem ser utilizados. Podemos citar como os mais difundidos os seguintes:

1. **QuadTree (2D) ou Octree (3D)** - Métodos desenvolvidos por Mark Shephard e Rensselaer [22]. Este método baseia-se na decomposição do espaço em quadrados, acompanhando a fronteira do domínio. Quadrados (cubos) são recursivamente subdivididos até que a aproximação do modelo geométrico contido no domínio seja considerada suficiente. Após a decomposição, cada quadrado que intersecta a fronteira do domínio é substituído por um polígono escolhido com base em um conjunto de padrões pré-definidos. A triangulação dos quadrados e dos polígonos produz uma malha que aproxima o modelo geométrico de interesse. A principal vantagem deste método é a adaptatividade da malha, dado que a subdivisão dos quadrados tende a refinar os elementos justamente nos pontos de maior complexidade do domínio [5]. Como uma desvantagem temos a tendência à introdução de elementos direcionais devido ao padrão fixo de triangulação dos quadrados, como visto no trabalho de Vijay e Lee [29].

2. **Avanço de Fronteira** - Rainald Lohner [15] e S. H. Lo [14] foram os principais contribuintes deste método. Seu funcionamento se baseia na subdivisão da fronteira do domínio e na inserção de pontos em camadas ao redor de cada componente da fronteira [14]. Uma fronteira ativa, onde os novos elementos são formados, é mantida, fazendo com que esta avance para as regiões ainda não trianguladas, de modo a preenchê-las com elementos. É importante ressaltar que devido à técnica de inserção de elementos escolhida pelo método, estes tendem a se orientar com base nas linhas de fluxo, aumentando a robustez dos métodos de simulação numérica, tornando este um forte candidato para uso em aplicações de dinâmica de fluidos. Como desvantagem podemos citar a dificuldade em assegurar uma boa subdivisão inicial da fronteira de modo a garantir o correto fechamento de frentes que colidem no interior do domínio [15]. A garantia

de corretude é difícil de se demonstrar, sendo esta uma de suas principais desvantagens.

3. **Delaunay** - A maior parte das técnicas de geração de malhas tem utilizado o critério de Delaunay, também conhecido como o critério do círculo vazio (esfera vazia em 3D). Dado um conjunto de pontos no plano, a triangulação de Delaunay é a triangulação cujo circuncírculo dos triângulos não possui nenhum dos pontos do conjunto em seu interior [12]. As técnicas de geração de malhas baseadas no critério de Delaunay têm tido destaque significativo na comunidade de geometria computacional. A principal razão para este fato é o surgimento de resultados teóricos que permitem assegurar a qualidade e a organização da malha gerada [25] [10]. Podemos citar com uma das principais limitações deste método a dificuldade em satisfazer na prática as restrições impostas pela teoria. Considerando que a fronteira do domínio possui ângulos pequenos, a inserção de pontos Steiner precisa ser realizada com um cuidado especial [28].

Métodos variados abordam o problema de uma forma híbrida, buscando a solução para o problema através da composição de duas ou mais técnicas. Por exemplo, algumas abordagens têm unido as vantagens da triangulação de Delaunay com os métodos de QuadTree [19] e Avanço de Fronteira [17]. Os resultados são bons, porém o custo computacional, em geral, é maior.

Bern e Eppstein [5] e Teng e Wong [30] são bons trabalhos que contemplam a geração de malhas não estruturadas e podem ser utilizados como fonte de revisões de tais métodos.

Dizemos que uma triângulação é ótima se esta é a melhor de acordo com uma métrica baseada no formato, tamanho e número de trinângulos [5].

A qualidade de um triângulo abc tem estreita relação com seu menor ângulo, que definimos como θ . Esta qualidade pode ser classificada pela medida de seu maior ângulo ou pela razão de aspecto do triângulo em questão. Edelsbrunner [10] constatou que um bom limite inferior para o menor ângulo implica em bons limites para as outras duas expressões de qualidade. O maior ângulo deve ser no máximo $\phi - 2\theta$. Desta forma, se o menor ângulo θ é limitado pela expressão anterior, o maior ângulo também é.

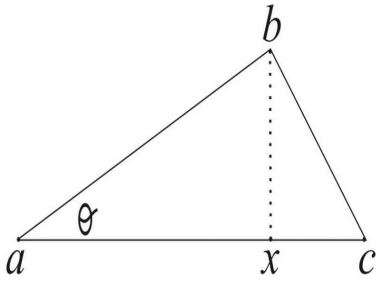


Fig. 1.5: Triângulo com base ac, altura bx e ângulo mínimo θ.

Por fim, apresentamos a segunda métrica de qualidade de um triângulo que é sua razão de aspecto. A razão de aspecto pode ser definida como o comprimento da maior aresta, que assumimos ser ac (figura 1.5), dividido pela distância de b até ac. Supondo-se que o menor ângulo é $\angle cab$, temos que $||b - x|| = ||b - a|| \cdot sin\theta$, onde x é a projeção ortogonal de b em ac. O comprimento de ab é no mínimo igual ao comprimento de cb, e assim $||b - a|| \geq \frac{||c-a||}{2}$. Ou seja,

$$\frac{1}{sen\theta} \leq \frac{||b - a||}{||b - x||} \leq \frac{2}{sen\theta} \tag{1.1}$$

o que assegura a afirmação acima.

No contexto dos geradores de malhas e simulações, introduzimos brevemente o projeto GESAR, que tem estreita relação com este trabalho e que, portanto, poderá fazer uso de seus resultados.

1.1.1 Projeto GESAR

A formação de represas em usinas hidroelétricas tem repercussões severas no meio ambiente, tanto nos cursos d'água quanto nas suas vizinhanças, interferindo nos ecossistemas e nos modos de vida das populações envolvidas. Um dos momentos mais críticos é o enchimento do reservatório quando a água se acumula gradualmente.

Durante este processo a biomassa terrestre é decomposta, lançando substâncias que se concentram nos volumes dos diferentes compartimentos do reservatório. Nestas condições, algumas regiões do reservatório passam por períodos em que os teores de oxigênio dissolvido e a concentração de matéria orgânica comprometem o equilíbrio da flora e da fauna locais.

Não é conhecido, até o momento, nenhum simulador que descreva o processo de enchimento de compartimentos em detalhe, levando em consideração as características dos ecossistemas locais, de modo a fornecer informações confiáveis e geograficamente detalhadas que permitam um planejamento otimizado da limpeza da bacia de acumulação.

O projeto GESAR tem como objetivo criar uma ferramenta capaz de simular em detalhes o comportamento dos futuros compartimentos do reservatório, durante seus respectivos enchimentos, podendo fornecer informações mais precisas de onde, quando, qual e quanto material orgânico deve ser retirado.

Anteriormente à simulação, porém, é necessária a análise dos dados que descrevem a região de interesse, ou seja, o local geográfico onde será, possivelmente, construída uma barragem e constituído um reservatório.

As informações sobre os terrenos são obtidas através de imagens de satélite. Um exemplo pode ser visto na figura 1.6. Os satélites oferecem imagens de alta resolução, possibilitando uma discretização de maneira que não haja perda considerável de precisão.

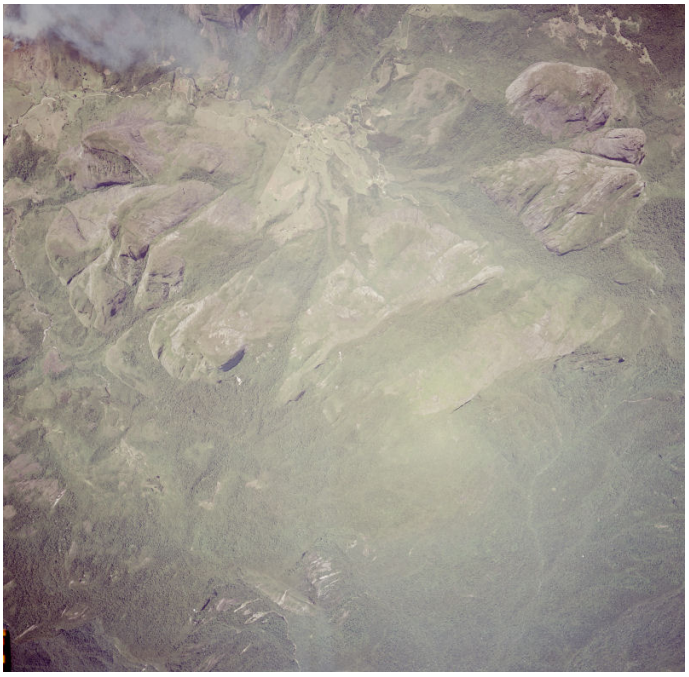


Fig. 1.6: Imagem aérea de um terreno propício à construção de um reservatório.

A discretização dos dados é feita através dos métodos de processamento de imagens, fornecendo como saída um conjunto de pontos com informações de elevação.

Constrói-se, então, uma malha de triângulos que será posteriormente utilizada por um simulador numérico para calcular os dados necessários.

O simulador se encarrega de tomar os dados do terreno, modelados através de uma malha, e utilizar os parâmetros físicos que influenciam o processo de enchimento dos reservatórios a fim de inferir um possível impacto ambiental.

Este trabalho tem estreira relação com a etapa que está situada entre a recepção dos dados e a simulação numérica.

Mais detalhes serão fornecidos adiante.

1.2 Objetivos do Trabalho

Este trabalho tem como objetivo o estudo dos conceitos de Geometria Computacional voltados ao problema de geração de malhas Delaunay bem como a implementação de um método inovador de geração de malhas que pode auxiliar na solução dos problemas relacionados ao projeto GESAR.

Os algoritmos de refinamento também fazem parte do escopo deste trabalho, dado que a melhoria da qualidade de uma malha também é de especial interesse para fins de simulações. Também é apresentado um estudo comparativo entre os métodos de geração e refinamento de malhas, de modo

a ressaltar as qualidades de cada abordagem.

1.3 Estrutura do texto

O texto se encontra segmentado em capítulos, sendo que cada um dos capítulo aborda um tema em particular.

O Capítulo 1 dedica-se a uma breve introdução às simulações numéricas e à area de geração de malhas. Fazemos uma breve contextualização do projeto GESAR, tendo este estreita relação com os temas abordados neste trabalho.

O Capítulo 2 trata dos conceitos gerais de geração de malhas e em especial aos referentes às triangulações ditas Delaunay. Neste capítulo são cobertas as definições e propriedades básicas de uma triangulação de Delaunay. Também são descritos vários métodos para a obtenção de uma triangulação de Delaunay, destacando-se os seguintes:

- Algoritmo de Flipping
- Algoritmo de Flipping incremental
- Algoritmo Incremental
- Algoritmo de Lifting
- Algoritmo Dividir e Conquistar
- Outros algoritmos de triangulação Delaunay

O Capítulo 3 cobre os algoritmos de refinamento Delaunay, sendo estes algoritmos capazes de tomar como entrada uma triangulação de Delaunay com determinado grau de refinamento e torná-la ainda mais densa em determinadas regiões, seguindo algum critério de refinamento. Os algoritmos cobertos neste capítulo são os seguintes:

- Algoritmo de Chew
- Algoritmo de Ruppert
- Algoritmo de Equilíbrio de Forças

O Capítulo 4 apresenta as implementações realizadas, bem como informações sobre a estrutura de dados que serviu como base para tais implementações.

O Capítulo 5 tem o foco nos resultados obtidos pelo trabalho.

Por fim, o Capítulo 6 apresenta as conclusões do trabalho e planos para trabalhos futuros.

Capítulo 2

Triangulação de Delaunay

2.1 Introdução

Pesquisas na área de Geometria Computacional ao longo dos anos contribuíram para o desenvolvimento de vários algoritmos de geração de malhas. Tais métodos geram malhas de triângulos utilizando diferentes abordagens e processando os dados de entrada de maneiras variadas.

No entanto, um tipo de triangulação tem se destacado por suas propriedades bem definidas. Esta triangulação é chamada *Triangulação de Delaunay*.

Além de ser uma triangulação com propriedades teóricas bastante compreendidas, há uma variedade muito grande de algoritmos capazes de gerar triangulações de Delaunay a partir de um conjunto de entrada. Esta triangulação também está intimamente relacionada a outras estruturas de grande importância na área de Geometria Computacional, destacando-se o Diagrama de Voronoi e o Fecho Convexo.

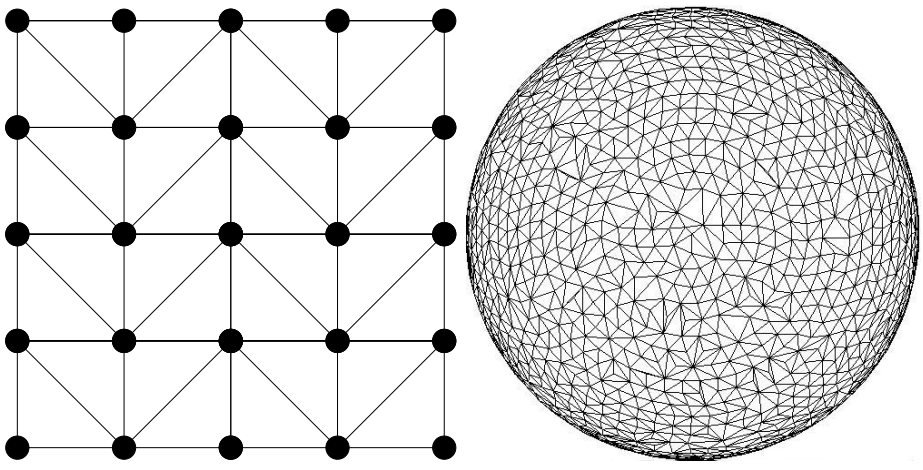


Fig. 2.1: Triangulação simples (esquerda) e triangulação de um objeto circular (direita)

No caso 2D, temos um bom entendimento das propriedades da triangulação de Delaunay por parte da comunidade especializada, comprovado pelo número de artigos publicados nessa área. Este tipo de triangulação, em duas dimensões, será o alvo principal ao longo do desenvolvimento deste trabalho.

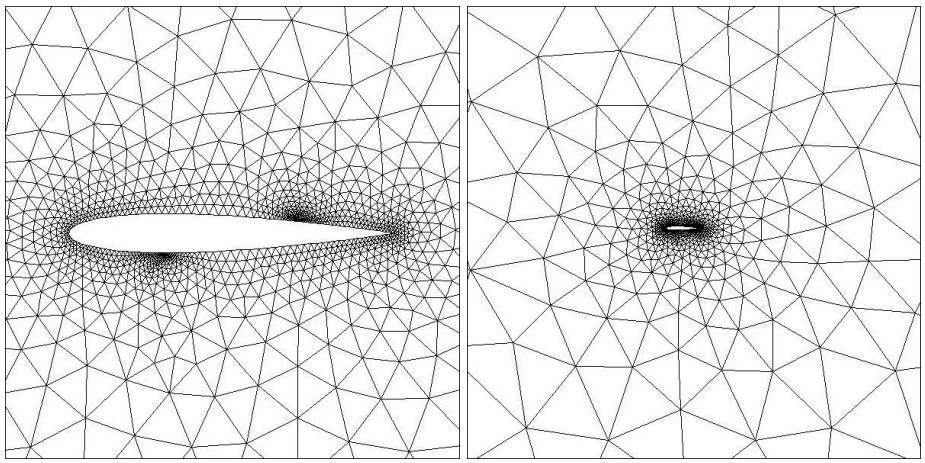


Fig. 2.2: Triangulações de Delaunay (Algoritmo de Refinamento de Ruppert)

Outra abordagem de extrema importância são as gerações de malhas em 3 dimensões. No entanto, o caso tridimensional ainda não é tão bem compreendido quanto o bidimensional. O acréscimo de dimensão traz um aumento de complexidade demasiadamente grande, tornando os estudos mais delicados e a obtenção de resultados teóricos mais difícil. É uma área muito importante que ainda conta com problemas em aberto.

Nas figuras 2.1 e 2.2 temos exemplos de vários tipos de triangulação, porém só as apresentadas na figura 2.2 são Delaunay. A figura 2.3 ilustra um exemplo de uma triangulação Delaunay 3D.

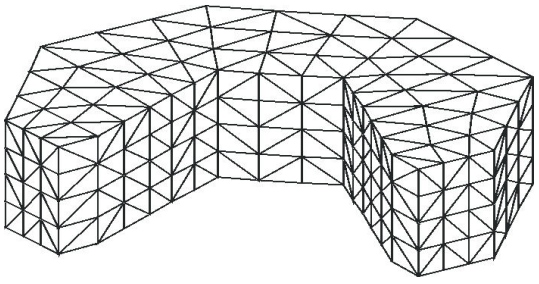


Fig. 2.3: Triangulação de Delaunay superficial em um objeto 3D

Para compreender melhor a triangulação de Delaunay, necessitamos de algumas definições que são enunciadas na seção seguinte.

2.2 Conceitos Básicos

Ao longo desta seção, são apresentados conceitos fundamentais para a compreensão da teoria envolvida na triangulação de Delaunay.

Definição 1 (Espaço Euclidiano) *Chamamos de* Espaço Euclidiano *de dimensão n, o espaço $\mathbf{R}^n$ dotado de métrica euclidiana comum, isto é, dados dois pontos $p = (p_1, \ldots, p_n)$, $q = (q_1, \ldots, q_n) \in \mathbf{R}^n$, a distância entre p e q é calculada da seguinte maneira:*

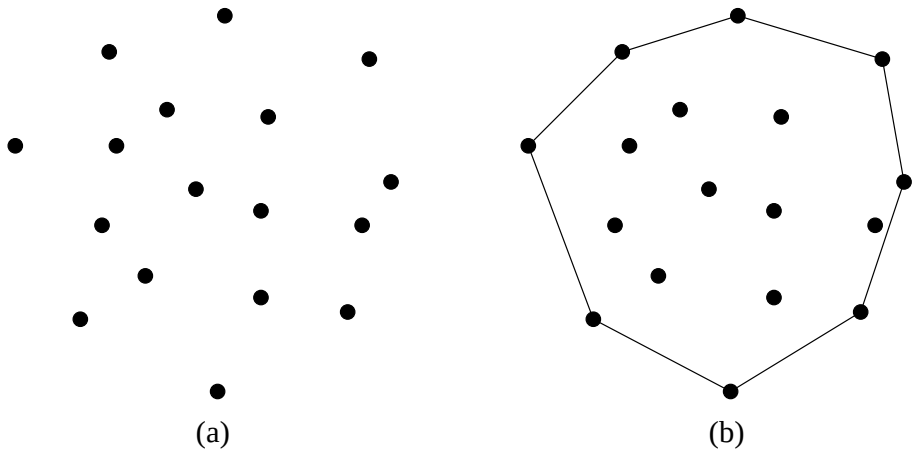
$$||p - q|| = \sqrt{(p_1 - q_1)^2 + \ldots + (p_n - q_n)^2} \tag{2.1}$$

Definição 2 (Fecho Convexo) *Dado um conjunto de pontos $P \subset \mathbf{R}^n$, o conjunto de todas as combinações lineares convexas, dado pela equação 2.2, exprime a definição de* fecho convexo *(figura 2.4).*

$$Conv(P) = \sum_{i=1}^{n} \lambda_i p_i, \sum_{i=1}^{n} \lambda_i = 1, \lambda_i \geq 0, \lambda_i \in \mathbf{R}, p_i \in P \tag{2.2}$$

O fecho convexo $Conv(P)$ pode ser visto como o menor conjunto convexo que contém os pontos de P. Especificamente em duas dimensões, o fecho convexo é o menor polígono convexo que contém todos os pontos. Cada ponto de P é um vértice do bordo do fecho ou está contido em seu interior. Na figura 2.4, temos um conjunto de pontos e também a representação de seu fecho convexo.

O fecho convexo está estreitamente relacionado aos conceitos de triangulação (definição 17) e triangulação de Delaunay (definição 24).



(a) (b)

Fig. 2.4: Conjunto de pontos (esquerda) e seu respectivo fecho convexo (direita)

Definição 3 (Esfera S^n) *A* esfera S^n *centrada em* $c = (c_0, c_1, ..., c_n)$*, com raio* r*, é o conjunto dos pontos* $\{P \in \mathbf{R}^{n+1}/P = (x_0, x_1, ..., x_n), \sqrt{(x_0 - c_0)^2 + (x_1 - c_1)^2 + ... + (x_n - c_n)^2} = r\}$.

Definição 4 (Subespaço Afim) *Um* subespaço afim *é um subespaço vetorial transladado da origem.*

Definição 5 (Célula) *Uma célula convexa afim, gerada por* $v_0, v_1, \ldots, v_n$*, é o fecho convexo de* $v_0, v_1, \ldots, v_n$.

Definição 6 (Dimensão da célula) *A dimensão de uma célula convexa é dada pelo maior número de vetores linearmente independentes do conjunto* $v_1 - v_0, v_2 - v_0, \ldots, v_n - v_0$*. Células de dimensão 1 são chamadas de arestas e células de dimensão 0 são chamadas de vértices.*

Definição 7 (Posição Geral) *Dizemos que um conjunto de pontos* $X \subset \mathbf{R}^n$ *está em* posição geral *se:*

1. *Nenhum subespaço afim de dimensão menor que* n *em* $\mathbf{R}^n$ *contém* X*;*
2. *Nenhuma esfera* S^{n-1} *intercepta mais de* $n + 1$ *pontos de* X.

Definição 8 (Simplexo) *Um* n-simplexo $\sigma \subset \mathbf{R}^n$ *é o fecho convexo de um conjunto de* $n + 1$ *pontos em posição geral (*figura2.5*).*

A Figura 2.5 ilustra quatro simplexos de dimensões diferentes, temos em (a) um simplexo de dimensão 0 (vértice), em (b) um simplexo de dimensão 1 (aresta), em (c) um simplexo de dimensão 2 (triângulo) e em (d) um simplexo de dimensão 3 (tetraedro).

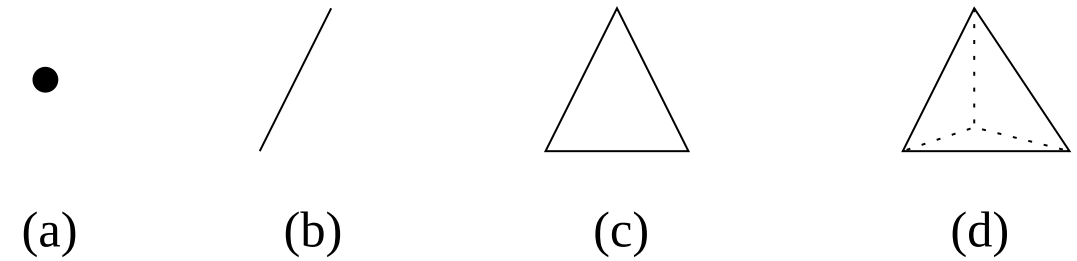


Fig. 2.5: a) 0-simplexo, b) 1-simplexo, c) 2-simplexo, d) 3-simplexo.

Definição 9 (Face) *Se* σ *é um* n*-simplexo formado pelos pontos* $X = \{p_0, ..., p_n\}$*, qualquer* l*-simplexo dado por um subconjunto de* $l + 1$ *elementos de* X*,* $0 \leq l < n$*, é uma* face *de* σ.

Definição 10 (Fronteira de um n-simplexo) *A* fronteira de um n-simplexo σ*, denotada por* $\partial\sigma$*, consiste de todos os* $(n - k)$*-simplexos contidos em* σ*,* $0 < k \leq n$.

Definição 11 (Complexo Simplicial) *Um* complexo simplicial C *é um conjunto finito de simplexos satisfazendo as seguintes propriedades:*

1. *Se* σ *é um simplexo em* C *e* σ' *é uma de suas faces, então* σ' *também pertence a* C.
2. *Se* σ_1 *e* σ_2 *são dois simplexos de* C*, então* $\sigma_1 \cap \sigma_2$ *ou é face de ambos, ou é vazia (*figura 2.6*).*

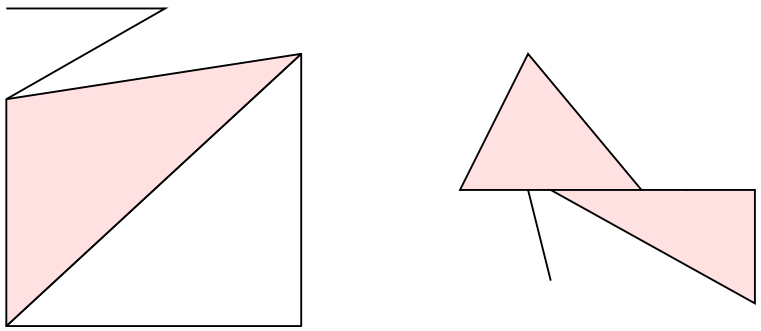


Fig. 2.6: a) Exemplo de um complexo simplicial, b) não define um complexo simplicial, note que existem interseções que não definem uma face compartilhada.

Definição 12 (Dimensão de um Complexo Simplicial) *A* dimensão *de um complexo simplicial* C *é definida como sendo a dimensão máxima de seus simplexos.*

Se σ é um n-simplexo e $n > 0$, então a fronteira $\partial\sigma$ é um complexo simplicial de dimensão $n-1$.

Definição 13 (Decomposição Simplicial) *Dado um complexo simplicial* C*,* C *é uma* decomposição simplicial *para um conjunto* $B \subset \mathbf{R}^n$ *se* $B = \bigcup_{\sigma \in C} \sigma$.

Definição 14 (Estrela) *A* estrela *de um simplexo* σ*, denotada por* $Star(\sigma)$*, é o conjunto de simplexos em* C *que contém* σ *como face* (Figura 2.7).

Definição 15 (Link) *O* link *de um simplexo* σ*, denotado por* $Link(\sigma)$*, são todos os simplexos de* $\partial(Star(\sigma))$ *que não são incidentes a* σ (Figura2.7).

Definição 16 (Grafo) *Um grafo* $G(V, E)$ *é definido como um conjunto de vértices* $V \neq \emptyset$ *e um conjunto finito de arestas* E*, conectando os vértices de* V.

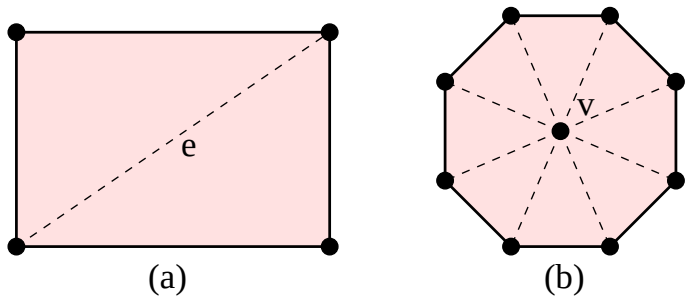


Fig. 2.7: As regiões hachuradas definem a) a estrela de uma aresta e e b) a estrela de um vértice v. As arestas mais largas indicam os respectivos *links* da aresta e e do vértice v.

Definição 17 (Triangulação de um conjunto de Pontos) *Seja $A = \{p_0, ..., p_k\} \subset \mathbf{R}^n$, um conjunto finito de pontos em $\mathbf{R}^n$. Uma* triangulação *de A é uma decomposição simplicial de $Conv(A)$ onde os vértices dos simplexos são os pontos de A.*

Uma triangulação (definição 17) é um exemplo de grafo, considerando-se apenas os vértices e arestas.

Definição 18 (Diagrama de Voronoi) *Seja $P \subset \mathbf{R}^n$ um conjunto de pontos que, no caso do diagrama de Voronoi, serão chamados sites. Para cada par de sites $p, q \in P$, seja $B(p,q) = \{x : ||x-q|| = ||x-p||\}$ o bissetor de p e q. Considere o semi-espaço $D(p,q) = \{x : ||x-p|| \leq ||x-q||\}$ contendo p, e $D(q,p)$ o semi-espaço que contém q. Define-se por $VR(p,P) = \bigcap_{q \in P, q \neq p} D(p,q)$ a* Região de Voronoi de p[1]. *O* Diagrama de Voronoi *de P é definido por:*

$$V(P) = \bigcup_{p \in P} VR(p,P) \tag{2.3}$$

O exemplo de um diagrama de Voronoi de dimensão 2 é ilustrado na figura 2.8.

Definição 19 (Circunsfera) *Define-se circunsfera de um n-simplexo σ como a esfera S^{n-1} contendo todos os vértices de σ. Em $\mathbf{R}^2$, denomina-se* circuncírculo. *O raio da circunsfera é denominado* circunraio .

Definição 20 (Invasão) *Dizemos que um vértice v invade uma aresta ab se este vértice está localizado dentro da circunferência de diâmetro definido por ab.*

[1]Conhecido também como célula de Voronoi.

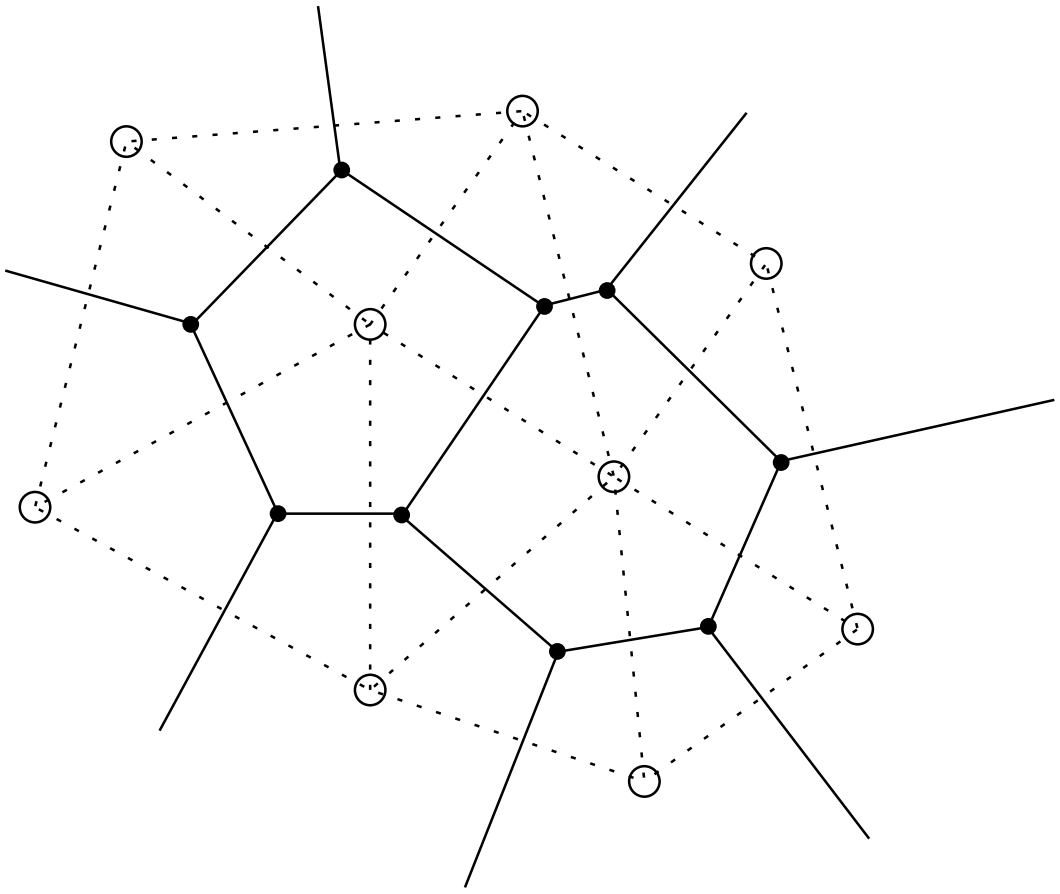


Fig. 2.8: O diagrama de Voronoi dos pontos sólidos é destacado em linhas contínuas. Em linhas pontilhadas temos o dual do Diagrama de Voronoi, a Triangulação de Delaunay.

Definição 21 (Lente Diametral) *A lente diametral de um segmento de reta s é a intersecção de dois discos cujos centros estão no bissetor de s a uma distância $\frac{|s|}{\sqrt{3}}$ de s e possuem raio $\frac{|s|}{\sqrt{3}}$.*

Definição 22 (Círculo Diametral) *O círculo diametral de um segmento de reta s é uma circunferência cujo diâmetro é o comprimento da aresta s.*

Definição 23 (Invasão) *Alternativamente, definimos a invasão de um aresta ab por um vértice v se v está contido na lente diametral definida pela aresta ab.*

A Figura 2.9 ilustra o círculo diametral e a lente diametral.

Definição 24 (Triangulação de Delaunay) *Seja $X = \{p_0, ..., p_k\} \subset \mathbf{R}^n, n \leq k$, um conjunto de pontos em posição geral. Diz-se que uma triangulação de X é* Delaunay *se a circunsfera de todo n-simplexo não contém nenhum outro ponto de X em seu interior (*Figura 2.10 e 2.11*).*

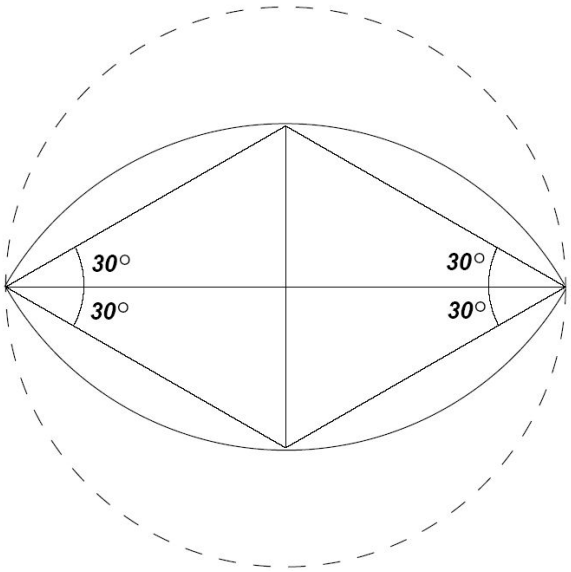


Fig. 2.9: Círculo diametral (pontilhado) e lente diametral (linha sólida). As lentes diametrais permitem que pontos sejam inseridos mais próximos às arestas de bordo.

Pode-se mostrar que o diagrama de Voronoi e a triangulação de Delaunay possuem uma relação de dualidade, como ilustra a Figura 2.8.

Definição 25 (PSLG - Planar Straight Line Graph) *Um* PSLG *é um grafo que satisfaz duas restrições:*

1. *Cada aresta é um segmento de reta.*
2. *Dois segmentos só podem se interceptar nos seus vértices extremos.*

2.3 Triangulações de Delaunay

Nesta seção descrevemos Lemas e Definições relevantes ao estudo das triangulações de Delaunay, de modo a complementar nossa base teórica. As provas dos lemas podem ser encontradas em [28].

Definição 26 (Círculo vazio de pontos) *Dada uma aresta ab de uma triangulação, dizemos que um círculo que passa pelos extremos de ab e não contém nenhum outro vértice da triangulação é vazio de pontos.*

Definição 27 (Aresta globalmente Delaunay) *Uma aresta ab de uma triangulação é dita globalmente Delaunay se existe um círculo vazio de pontos que passa pelos seus extremos.*

Toda aresta de bordo é globalmente Delaunay.

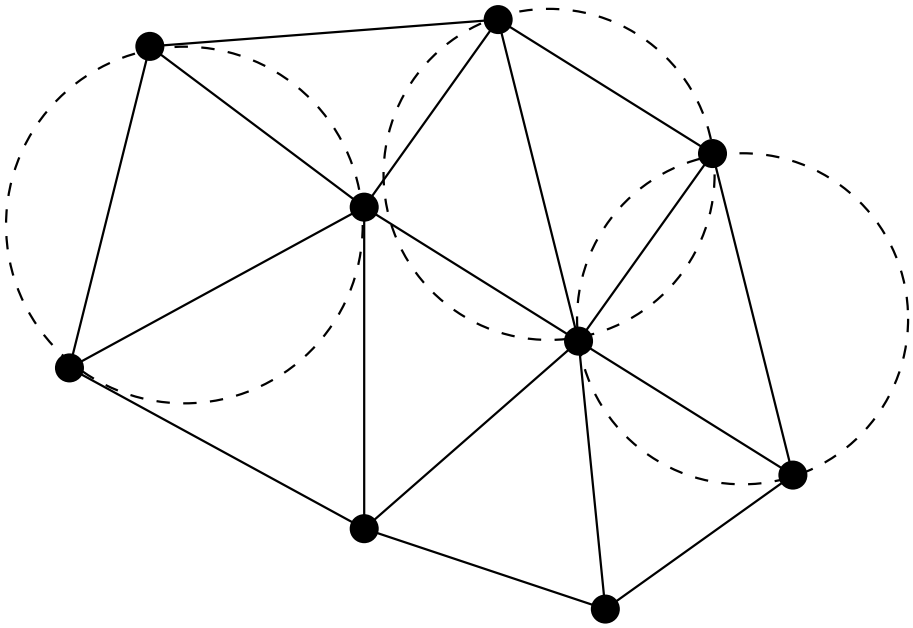


Fig. 2.10: Em uma triangulação de Delaunay os triângulos possuem circuncírculos vazios.

Lema 1 *Seja T uma triangulação no plano. Se todos os triângulos de T são Delaunay, ou seja, possuem circuncírculo vazio de pontos, todas as arestas contidas em T são globalmente Delaunay e vice-versa.*

Definição 28 (Aresta localmente Delaunay) *Seja e uma aresta compartilhada por dois triângulos t_1 e t_2. Tomemos v_1 e v_2 como os vértices opostos a e em t_1 e t_2, respectivamente. A aresta e é dita localmente Delaunay se o circuncírculo de t_1 não contém v_2 e se o circuncírculo de t_2 não contém v_1 (Figura 2.12).*

Lema 2 *Seja T uma triangulação. Consideremos que todas as suas arestas são localmente Delaunay. Então todas as arestas de T são globalmente Delaunay.*

Definição 29 (Flipping) *Consideremos um conjunto de quatro pontos. Tomemos o quadrilátero formado por eles. Se este quadrilátero é convexo, então ele possui duas diagonais, d_1 e d_2. A diagonal d_1 é dita flipping da diagonal d_2 e vice-versa. (Figura 2.13).*

Nem todos os quadriláteros possuem uma forma convexa, isto é, muitos deles podem apresentar concavidade. Esta concavidade ocasiona o aparecimento de diagonais externas, que não são passíveis de flipping. Desta maneira, dizemos que se um quadrilátero não possui diagonais internas, as diagonais não são flipáveis no quadrilátero. Uma ilustração está disponível na Figura 2.14.

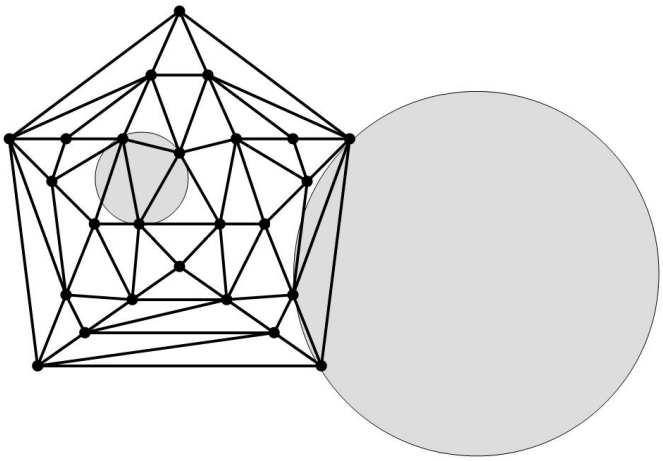


Fig. 2.11: Triangulação de Delaunay de um conjunto de pontos e circuncírculos vazios de dois triângulos.

Lema 3 *Seja e uma aresta de uma triangulação T. Temos 2 possibilidades:*

1. *A aresta e é localmente Delaunay. (Figura 2.12)*

2. *A aresta e não é localmente Delaunay, porém é flipável e o flipping de e é localmente Delaunay. (Figura 2.13)*

Definição 30 (Pontos visíveis) *Dado um conjunto de pontos P e um conjunto de segmentos S, dizemos que dois pontos a e b de P são visíveis se o segmento ab não intercepta qualquer segmento de S.*

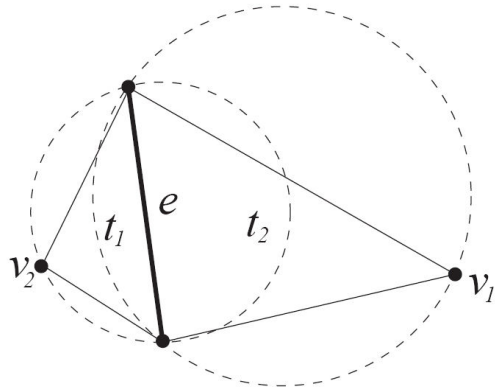
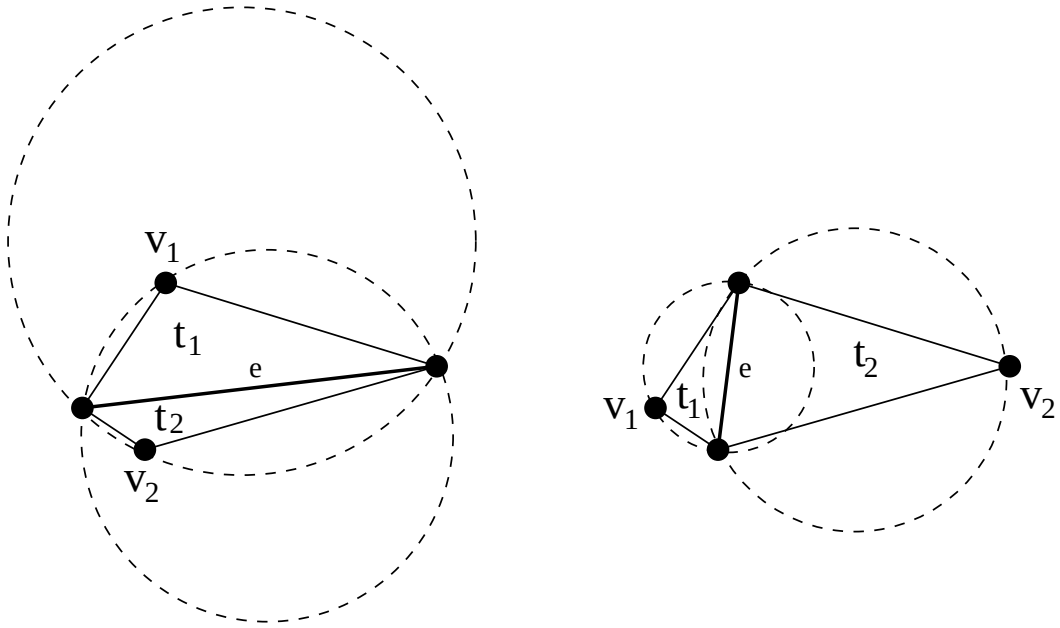


Fig. 2.12: Exemplo de uma aresta localmente Delaunay e



(a) (b)

Fig. 2.13: Processo de flipping de uma diagonal

Definição 31 (Triangulação de Delaunay com restrição) *Seja um PSLG $G = (V, A)$. A Triangulação de Delaunay Restrita (TDR) de G é uma triangulação $T(G) = (V, E')$, $E \subset E'$, onde o circuncírculo de cada triângulo $\Delta(v_i v_j v_k)$ não contém em seu interior qualquer outro vértice de V que seja visível a partir dos vértices v_i, v_j e v_k do triângulo. As arestas do conjunto $A' - A$ são ditas arestas Delaunay.*

Com base nas definições e nos lemas vistos anteriormente neste capítulo, enunciamos algumas propriedades da triangulação de Delaunay [11, 27]:

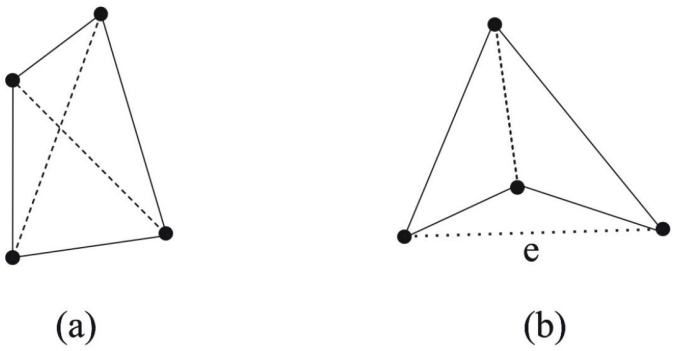


Fig. 2.14: Quadrilátero com diagonal flipável (a) e quadrilátero com diagonal não flipável (b)

- A Triangulação de Delaunay é única para pontos em posição geral.
- Por definição, o circuncírculo de cada triângulo não contém nenhum outro vértice da triangulação.
- Em $\mathbf{R}^2$, a Triangulação de Delaunay maximiza o ângulo mínimo (menor ângulo dentre os ângulos dos triângulos) dos triângulos da triangulação.
- A Triangulação de Delaunay maximiza o maior círculo mínimo (círculo que contém o triângulo e não necessariamente é o circuncírculo), em $\mathbf{R}^2$.
- A Triangulação de Delaunay minimiza o maior circuncírculo da triangulação.
- É o dual do diagrama de Voronoi [2];
- Em $\mathbf{R}^2$ é possível obter a triangulação em $O(k \log k)$, onde k é o número de pontos;
- Para pontos $(x, y) \in \mathbf{R}^2$, a triangulação é a projeção ortogonal da parte inferior do fecho convexo dos pontos $(x, y, x^2 + y^2) \in \mathbf{R}^3$ [11].

2.4 Algoritmos para triangulação

Nesta seção apresentaremos algumas das abordagens mais utilizadas para a geração de uma Triangulação de Delaunay. Como o foco deste trabalho não é exatamente a Triangulação de Delaunay, apenas uma breve descrição dos algoritmos é fornecida. Informações mais específicas podem ser encontradas nas referências.

2.4.1 Algoritmo de Flipping

O algoritmo de Flipping para geração de uma Triangulação de Delaunay possui um funcionamento bastante simples e direto. Como o próprio nome diz, o algoritmo se baseia no Flipping de arestas.

A entrada para este algoritmo é uma triangulação qualquer em $\mathbf{R}^2$, não necessariamente Delaunay. Com a entrada definida, as arestas são verificadas uma a uma pelo critério de localmente Delaunay. As arestas que não satisfizerem o critério, ou seja, não são localmente Delaunay, são flipadas.

Quando todas as arestas forem localmente Delaunay, teremos uma Triangulação de Delaunay. Sendo assim, o algoritmo de Flipping termina.

Em resumo, o algoritmo de Flipping em $\mathbf{R}^2$ consiste em, a partir de uma triangulação qualquer, flipar todas as arestas não localmente Delaunay.

Teorema 1 *Dada uma triangulação 2D, o algoritmo de flipping termina em $O(n^2)$, onde n é o número de vértices, produzindo uma triangulação de Delaunay.*

Mais detalhes sobre a desmontração do Teorema 1 podem ser encontrados em [4].

2.4.2 Algoritmo de Flipping Incremental

O algoritmo de flipping incremental utiliza os mesmos princípios básicos do algoritmo de Flipping, ou seja, utiliza as definições vistas anteriormente para a garantia das propriedades de Delaunay. A diferença é que não há uma triangulação definida a priori, sendo esta construída desde o início.

Desta maneira, suponha que uma Triangulação de Delaunay já tenha sido calculada para os pontos $p_1, p_2, p_3, ..., p_{i-1}$ e que p_i será inserido como um novo vértice. O algoritmo de *Flipping Incremental* encontra um triângulo abc que contém o ponto p_i, cria os triângulos p_iab, p_ibc e p_ica, e verifica se as arestas ab, bc e ca são localmente Delaunay.

Caso algumas das arestas não sejam localmente Delaunay, utiliza-se o flipping para que tais arestas passem a respeitar o critério. As outras arestas que fazem parte do link de p_i (Definição 15) também são testadas contra o mesmo critério.

O algoritmo começa com três vértices no infinito e termina quando não existem mais pontos a serem inseridos. A Figura 2.15 ilustra os passos do algoritmo de Flipping Incremental para a inserção do ponto p.

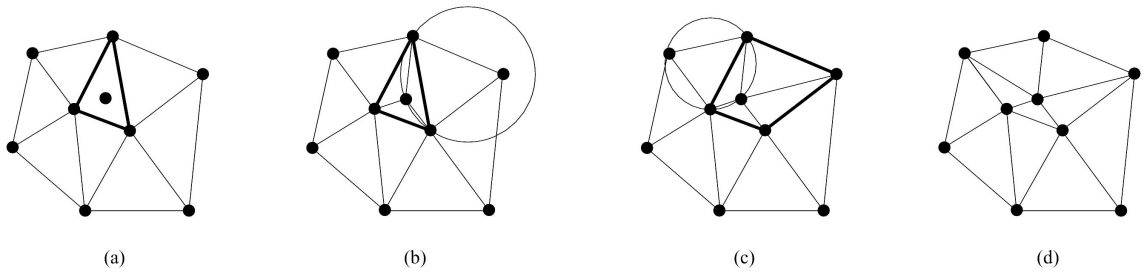


Fig. 2.15: Algoritmo de Flipping Incremental; os segmentos em negrito indicam as arestas que precisam ser testadas se são localmente Delaunay; (a) o triângulo que contém o ponto é encontrado e suas arestas devem ser testadas; (b) uma das arestas não é localmente Delaunay, ocorre o flipping, adicionando duas arestas às que devem ser testadas; (c) flipping de outra aresta e (d) final da inserção, onde todas as arestas são localmente Delaunay.

Alguns lemas que validam o algoritmo:

Lema 4 *p_ia, p_ib e p_ic são arestas Delaunay*

Prova: Pela propriedade de circuncírculo vazio da triangulação de Delaunay, considerando o triângulo abc pertencente à triangulação, seu circuncírculo é vazio.

Os círculos que contêm p_ia, p_ib e p_ic e estão no interior do circuncírculo do triângulo abc são vazios.

Assim as arestas p_ia, p_ib e p_ic são arestas (globalmente) Delaunay.

Lema 5 *Qualquer aresta incidente a p_i, pela propriedade de flipping, é localmente Delaunay.*

Prova: Este lema é provado pelo lema 3, onde qualquer aresta incidente a p_i é uma aresta da triangulação oriunda de um flipping, logo localmente Delaunay.

Lema 6 *Nenhuma aresta é testada mais de uma vez para cada novo ponto.*

Como cada aresta é testada apenas uma vez, temos $O(n)$ para cada novo ponto inserido. Este algoritmo também pode ser implementado em 3D.

2.4.3 Algoritmo Incremental

Supondo que a triangulação de Delaunay tenha sido previamente calculada para os pontos $p_1, p_2, p_3, \ldots, p_{i-1}$. O algoritmo Incremental encontra os triângulos cujos circuncírculos contém o ponto p_i e em seguida os remove. Esta operação gera um polígono estrelado com relação a p_i, ou seja, os vértices do polígono gerado anteriormente são visíveis a partir de p_i.

Após a remoção dos triângulos, o ponto p_i é unido aos vértices do polígono criado através da remoção dos triângulos antigos, como pode ser visto na Figura 2.16 [12].

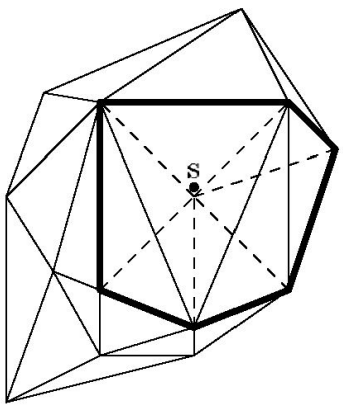


Fig. 2.16: Funcionamento do algoritmo Incremental durante a inserção de um ponto p

Se p_i não está no interior da triangulação, é necessário conectá-lo aos vértices de bordo visíveis a partir de p_i.

Lema 7 *Cada aresta com um extremo em p_i e outro no bordo do polígono gerado pela remoção dos triângulos é Delaunay.*

Prova: Esta prova segue o raciocínio idêntico ao do lema 4. Como as demais arestas já eram localmente Delaunay, então a triangulação é Delaunay.

2.4.4 Algoritmo de Lifting

O algoritmo de Lifting tem como princípio básico projetar os pontos a serem triangulados sobre a superfície de um parabolóide de coordenadas $(x, y, x^2 + y^2)$, onde (x, y) são as coordenadas cartesianas dos pontos, e calcular o fecho convexo deste parabolóide. Após o término do cálculo do fecho convexo, removem-se os triângulos da "tampa"do parabolóide.

As faces restantes do parabolóide são projetadas sobre o plano xy. Pode ser mostrado que esta projeção é a triangulação de Delaunay dos pontos.

Um exemplo do mecanismo de funcionamento deste algoritmo pode ser visto na Figura 2.17.

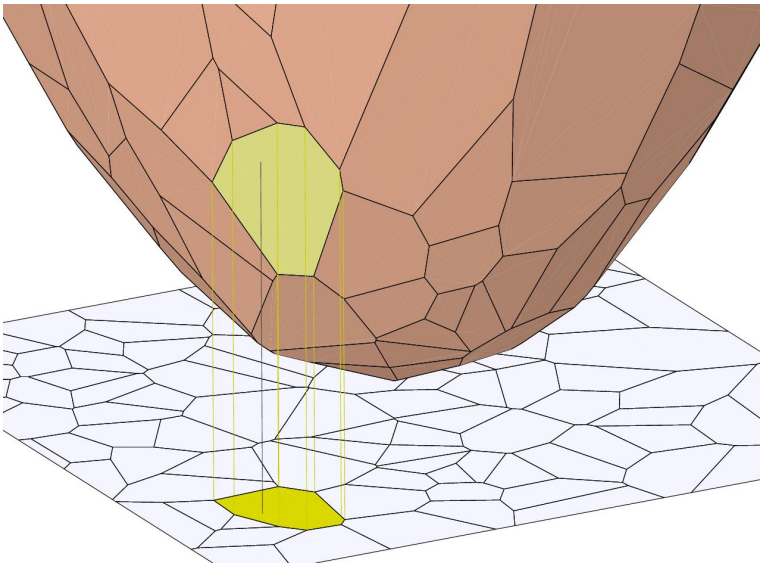


Fig. 2.17: Etapa final do algoritmo de Lifting com a projeção do parabolóide sobre o plano, resultando em um diagrama de Voronoi, do qual é possível obter diretamente uma triangulação de Delaunay.

2.4.5 Algoritmo Dividir e Conquistar

Este algoritmo possui uma abordagem semelhante a todos os algoritmos do tipo *Divide and Conquer*. Como um exemplo deste tipo de algoritmo podemos citar o *Quick Hull* para a geração de um fecho convexo em 2 ou 3 dimensões.

O funcionamento padrão deste tipo de algoritmo é a divisão inicial dos dados em conjuntos menores. Inicialmente os dados de entrada, no caso os pontos, são divididos em dois conjuntos distintos através de uma linha vertical. No próximo passo estes dois conjuntos são divididos novamente em outros dois conjuntos de dados, cada qual com metade dos pontos do conjunto inicial.

Quando algum conjunto apresentar um número mínimo de pontos para formar um triângulo ou uma aresta, a divisão é encerrada e a construção da triangulação pode ter início. A triangulação é gerada a partir da união dos vários subconjuntos de pontos, recursivamente, até a obtenção da triangulação completa.

Podemos observar na Figura 2.18 uma etapa do funcionamento do algoritmo, onde dois conjuntos distintos estão sendo concatenados.

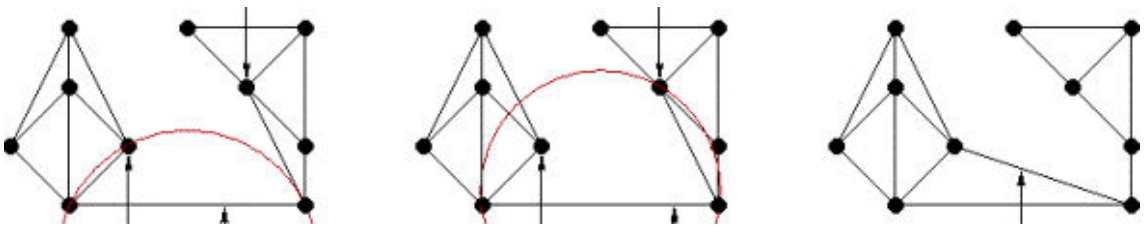


Fig. 2.18: Etapa de concatenação de triangulações no algoritmo Dividir para Conquistar

A concatenação leva um tempo linear no pior caso, com isso podemos observar uma ordem de complexidade algorítmica de $O(n \ln n)$. A implementação deste método é um pouco trabalhosa, porém foi a primeira abordagem a obter uma complexidade $O(n \ln n)$. Detalhes adicionais sobre este método podem ser vistos em [13].

2.4.6 Outras abordagens

Ainda podemos destacar uma outra classe de algoritmos que utiliza abordagens diferentes, com o algoritmo de Embrulho para Presente ou *Gift Wrapping*, que constrói o fecho convexo através de uma operação de *Liftting*, extraindo o diagrama de Voronoi em seguida. A partir do Diagrama de Voronoi, a Triangulação de Delaunay pode ser obtida pela propriedade dual. Este algoritmo é apenas um dos exemplos de abordagens que geram a Triangulação a partir de um Diagrama de Voronoi. Outros exemplos podem ser vistos em [1].

Outro algoritmo bastante conhecido é o *Plane Sweep*, desenvolvido por Fortune [12]. Este algoritmo constrói a Triangulação de Delaunay interativamente através da abordagem da *Sweep Line*. A *Sweep Line* varre os pontos verticalmente em ordem decrescente de coordenada y. Utilizando algumas propriedades dos diagramas de Voronoi, as fronteiras da triangulação vão sendo estendidas. Ao atingir o último ponto, que tem a menor coordenada y, o algoritmo termina.

Existem também algoritmos para a construção de uma triangulação de Delaunay com restrição, não estando limitado a uma entrada contendo somente um conjunto de vértices, podendo também receber como entrada um *Planar Straight Line Graph* (PSLG), que consiste de um conjunto de vértices e segmentos, onde os segmentos pertencentes ao PSLG devem pertencer à malha de saída. Dentre os autores que implementaram tais algoritmos, podemos citar Shewchuck [**?**], com a ferramenta Triangle, capaz de gerar triangulações rapidamente e possibilitando ajustes de vários parâmetros envolvidos no método. Chew também apresenta uma implementação da triangulação restrita em [**?**].

Capítulo 3

Geração de Malha Delaunay

Com a obtenção de uma Triangulação de Delaunay através de algum dos métodos citados anteriormente, é possível analisar se esta se encontra em padrões adequados de qualidade para os objetivos aos quais se aplica. Em geral, temos simulações numéricas como principal foco. Tais simulações necessitam de uma boa qualidade dos elementos da triangulação. Através dos algoritmos de refinamento, podemos adicionar seletivamente, através de critérios que serão vistos em breve, triângulos à malha, tornando sua qualidade superior à malha anterior.

Em métodos de elementos finitos a qualidade dos elementos que decompõem o domínio influencia na qualidade da solução das equações. Este fato tem gerado uma grande pesquisa na área de geração de malhas com razão de aspecto garantida, chamada malha de qualidade [7]. Uma vasta literatura sobre este assunto pode ser encontrada em [4].

No entanto os algoritmos devem ter um controle sobre a adição de pontos para que a malha não apresente um refinamento excessivo onde alguns dos pontos adicionados não acarretem mudanças significativas nos resultados das simulações, ou seja, são elementos desnecessários que só aumentam o tamanho da malha.

Podemos citar como um ponto positivo dos algoritmos de refinamento de Delaunay a exploração das características deste tipo de triangulação. Uma das características de muita importância é a maximização do ângulo mínimo dentre todas as triangulações de um mesmo conjunto de pontos. Outra característica é a localidade das operações da inserção de pontos, fazendo com que somente as regiões vizinhas sejam afetadas pela inserção dos pontos, preservando, assim, as outras regiões da malha. Também é importante ressaltar que o local de inserção dos novos pontos é fundamental para uma boa triangulação, sendo que pontos inseridos muito próximos aos outros tornam as arestas muito pequenas, resultando em triangulação de baixa qualidade. Considerando a propriedade de Delaunay que diz que o circuncírculo de um triângulo é vazio de pontos, esta é uma boa localização para a inserção de um novo ponto, como veremos a seguir.

Um ponto importante nos algoritmos de refinamento de Delaunay é descobrir o local ideal no qual o novo vértice deve ser inserido. Neste capítulo veremos que o local ideal para inserir um vértice é o mais longe possível dos outros vértices, caso contrário resultará em arestas pequenas, o que por sua vez resultará em triângulos muito finos.

3.1 Medida de Qualidade

Segundo Shewchuk [27] uma medida de qualidade natural e elegante para analizar os algoritmos de refinamento de Delaunay é a razão *circunraio-menor aresta* de um simplexo, ou seja, o raio da circunsfera de um simplexo dividido pelo tamanho da menor aresta do simplexo. É desejável que esta razão seja a menor possível. No caso bidimensional a razão *circunraio-menor aresta* de um triângulo é uma função do seu menor ângulo. Seja Δijk com circuncentro c e circunraio r, como ilustrado na figura 3.1(a). Tome d como o tamanho da menor aresta ij, e o ângulo oposto a esta aresta como α.

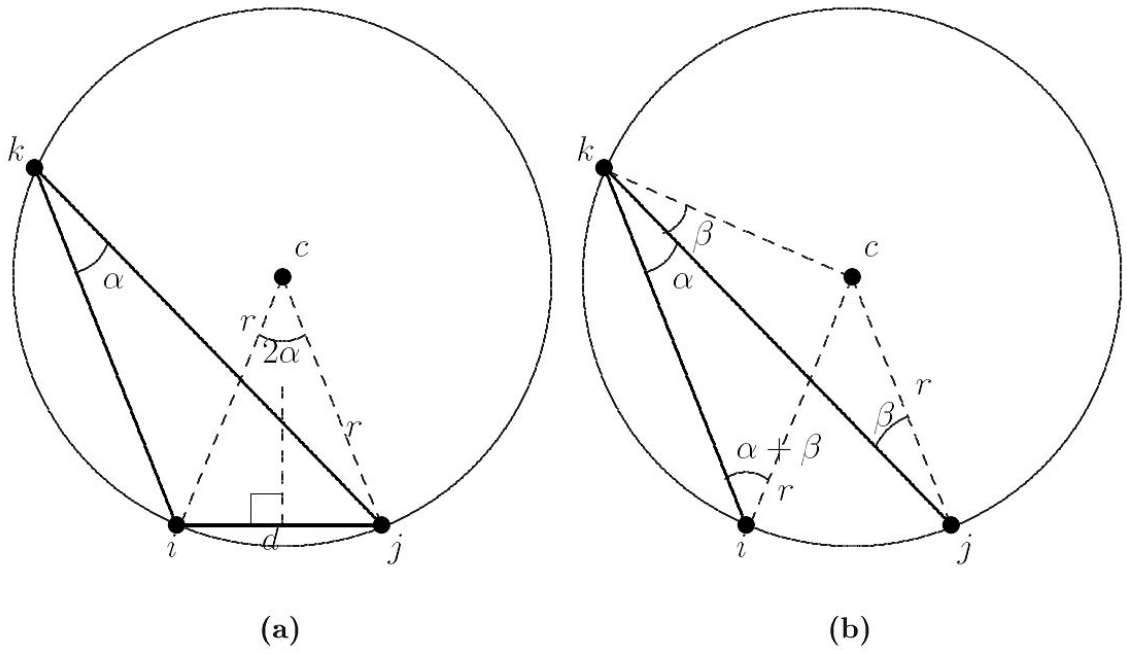


Fig. 3.1: Caracterização de qualidade

.

Um fato geométrico bem conhecido é que $\angle icj = 2\alpha$, como mostra a figura 3.1(a). Seja $\beta = \angle jkc$. Como Δkci e Δkcj são isósceles, $\angle kci = 180^o - 2(\alpha + \beta)$ e $\angle kcj = 180^o - 2\beta$. Subtraindo $\angle kci$ de $\angle kcj$ temos $\angle icj = 2\alpha$. Observando a Figura 3.1(a) pode-se notar que $\sin\alpha = \frac{d}{2r}$. Então, se a menor aresta do triângulo tem tamanho d, α é o menor ângulo. Se B é um limitante superior sobre a razão *circunraio-menor aresta* entre todos os triângulos da malha, então não há ângulo menor

que $\arcsin \frac{1}{2B}$. A seguir utilizamos B como medida de qualidade, uma vez que B está diretamente relacionado com o menor ângulo de um triângulo.

A seguir, apresentamos dois algoritmos de refinamento de Delaunay bidimensional, desenvolvidos por Paul Chew [8] e por Jim Ruppert[25]. Ambos utilizam um limitante B superior para a razão *circunraio-menor aresta* e recebem como entrada um domínio em formato PSLG.

Apresentamos ainda uma terceira abordagem para geração de malha Delaunay bidimensional, a qual é baseada em uma analogia com um sistema físico e equilíbrio de forças. Este método também é capaz de gerar malhas Delaunay com refinamento adaptativo, embora não de forma incremental pela adição de pontos.

3.2 Refinamento por Chew

O algoritmo desenvolvido por Chew [8] é capaz de gerar malhas de qualidade a partir de uma triangulação de Delaunay. O principal passo deste algoritmo é a inserção de novos pontos, estrategicamente posicionados para evitar a criação de elementos ruins.

A cada ponto inserido na triangulação, esta é refeita localmente para respeitar a nova organização. Este passo é repetido até que a malha apresente uma qualidade mínima especificada, a ser calculada a partir dos ângulos da triangulação. Atingida a qualidade necessária, o algoritmo termina.

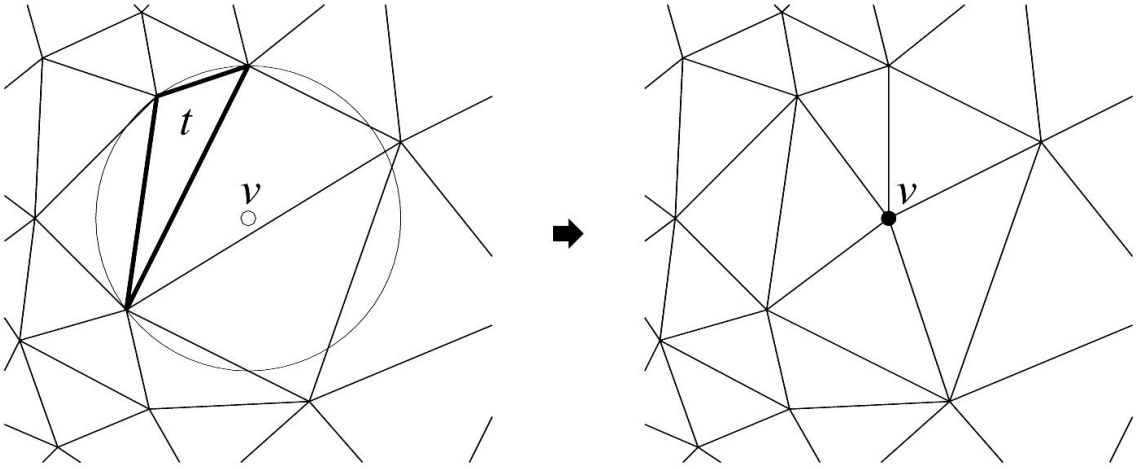


Fig. 3.2: Refinamento de Delaunay

.

O algoritmo de Chew possui algumas características que são descritas a seguir.

- As fronteiras internas e externas e os vértices previamente fornecidos pelo usuário são respeitados e preservados. Os triângulos gerados pelo algoritmo de Chew nunca cruzam as fronteiras e os vértices nunca são eliminados da triangulação.

- Todos os triângulos da malha possuem ângulos pertencentes ao intervalo entre 30^o e 120^o, excetuando-se os triângulos respeitando as fronteiras, que podem ser constituídas por polígonos que apresentam cantos com um ângulo menor do que 30 graus.

- Há um controle do tamanho dos triângulos que depende diretamente das características do polígono de entrada fornecido pelo usuário.

Como se trata de uma triangulação de Delaunay, sabemos que não há vértices no interior do circuncírculo de um triângulo t com qualidade inferior a B. Sendo v o circuncentro do triângulo t, nenhuma aresta criada com a inserção de v pode ser menor que o circunraio de t. Como a razão *circunraio-menor aresta* de t é maior do que uma constante B, as novas arestas terão tamanho mínimo B vezes a menor aresta de t.

Considerando h_{min} o comprimento da menor aresta de um dada triangulação T, o algoritmo de Chew refina a malha introduzindo vértices nos circuncentros de todos os triângulos cujos circunraios são maiores do que h_{min}. Sendo assim, o algoritmo não gera arestas com comprimento inferior a h_{min}. A restrição do tamanho mínimo da aresta faz com que o algoritmo tenha também uma condição de parada bem definida.

Como o algoritmo preserva as arestas da fronteira, estas podem impedir que algum triângulo ruim seja eliminado, pois o circuncentro que deveria ser inserido se posiciona fora do domínio especificado. A figura 3.3 ilustra este problema, onde o triângulo em negrito não pode ser eliminado pela inserção de um vértice no circuncentro. A região sombreada na figura 3.3 representa a região onde não é permitida a inserção de um novo vértice para o refinamento, pois este estaria a uma distância menor do que h_{min} de outro vértice. Nota-se que a região compreende todo o triângulo que deveria ser refinado.

A solução para o problema, proposta por Chew, é a seguinte. Seja G um polígono de entrada onde as arestas definem a região a ser triangulada e seja h um parâmetro do algoritmo tal que os segmentos que compreendem G sejam subdivididos em segmentos de comprimento l, tal que $h \leq l \leq \sqrt{3h}$. A restrição é que h deve ser menor que a menor distância entre dois elementos quaisquer de G (vértice e arestas), sendo assim as arestas adjacentes de G devem apresentar um ângulo maior que 30^o.

Com a técnica de subdivisão do polígono G, garantimos que as arestas do polígono de entrada estejam presentes na triangulação final e que os circuncentros sempre estejam contidos no interior do domínio.

O algoritmo de Chew para refinamento de malhas pode ser estendido para malhas de superfície. As malhas, neste caso, apresentarão as mesmas características de qualidade que apresentariam no caso 2D [9].

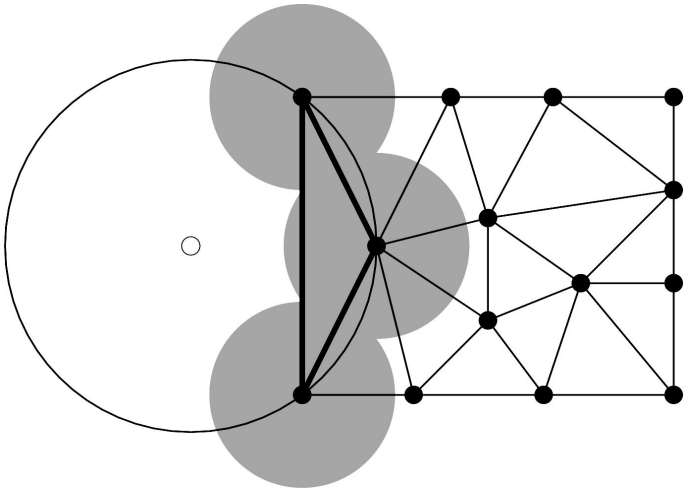


Fig. 3.3: Problema característico do refinamento de chew com triângulos residentes na fronteira da malha [27]

.

3.2.1 Algoritmo

Através das informações fornecidas anteriormente, é possível construir um algoritmo para fazer refinamento de Delaunay. O procedimento pode ser visto no Algoritmo 1

Algorithm 1 Algoritmo para refinamento de Chew

1: $G \leftarrow PSLG$
2: h_{min} : comprimento mínimo da aresta ou entre vértices
3: Dividir as arestas em segmentos de comprimento l tal que $h_{min} \leq l \leq \sqrt{3h_{min}}$, produzindo $G^{'}$
4: Criar a triangulação de Delaunay T com base nos vértices de $G^{'}$
5: **Enquanto** Há algum triângulos $t \in T$ que possui circunraio maior que h_{min} **Faça**
6: Inserir circuncentro de t em T
7: Atualizar a triangulação T
8: **Fim Enquanto**

3.3 Refinamento por Ruppert

O algoritmo de refinamento de Jim Ruppert [25] para geração de malhas bidimensionais com um nível de qualidade satisfatório é um método iterativo que tem como base os conceitos da Triangulação de Delaunay. A cada iteração, vértices adicionais são inseridos na malha, mantendo válida a propriedade de Delaunay, até que seja obtida uma triangulação que satisfaça as condições desejadas de qualidade.

O algoritmo pode ser aplicado a uma Triangulação de Delaunay com ou sem restrição, embora a apresentação que Ruppert faz de seu algoritmo se baseie em uma versão da triangulação que não contém restrições. A entrada do algoritmo é um PSLG. Assumindo a triangulação de Delaunay apenas dos vértices, ignorando a entrada de segmentos, alguns segmentos que fazem parte do PSLG podem não aparecer na triangulação, estes segmentos são ditos os segmentos ausentes. Um aspecto interessante do algoritmo de Ruppert é que os segmentos ausentes acabam sendo inseridos como conseqüência natural do algoritmo [28] restrito, visto que à medida que pontos são inseridos no ponto médio destes segmento ausente, temos uma aproximação cada vez melhor para a aresta ausente.

O algoritmo de Ruppert é uma extensão do primeiro algoritmo de Chew, e sua principal operação também é a inserção de pontos no circuncentro de triângulos de menor qualidade. A principal diferença é a variação na densidade dos triângulos. Enquanto o algoritmo de Chew gera uma malha mais homogênea, com densidade constante de triângulos, o algoritmo de Ruppert, por sua vez, utiliza uma densidade variável, o que acaba gerando uma quantidade menor de triângulos.

Segundo Shewchuk o primeiro algoritmo de geração de malha que fornece a prova teórica da qualidade da malha procurada na prática é, possivelmente, o algoritmo de Ruppert. Maiores detalhes sobre as demonstrações teóricas podem ser vistos em [25, 28].

3.3.1 Definições básicas

A seguir são fornecidas algumas definições básicas que auxiliam a compreensão do algoritmo.

Definição 32 (Segmento invadido) *Um segmento é dito segmento invadido se um vértice estiver estritamente dentro do seu círculo diametral.*

Um segmento do PSLG que não aparece como uma aresta da triangulação de Delaunay é sempre considerado um segmento invadido.

Quando um segmento é invadido, um vértice é imediatamente inserido no ponto médio deste segmento (Figura 3.4). Os dois segmentos resultantes têm círculo diametral menor, e podem ou não estar invadidos.

Como no algoritmo de Chew, cada triângulo ruim é dividido normalmente pela inserção de um vértice no seu circuncentro. A propriedade de Delaunay garante que este triângulo é eliminado. No entanto, se o novo vértice invadir algum segmento, então ele não será inserido, e, em vez disso, todos os segmentos que estariam invadidos serão divididos. Os segmentos invadidos têm prioridade sobre os triângulos ruins.

3.3.2 Algoritmo

O algoritmo de Ruppert refina a triangulação utilizando os seguintes passos:

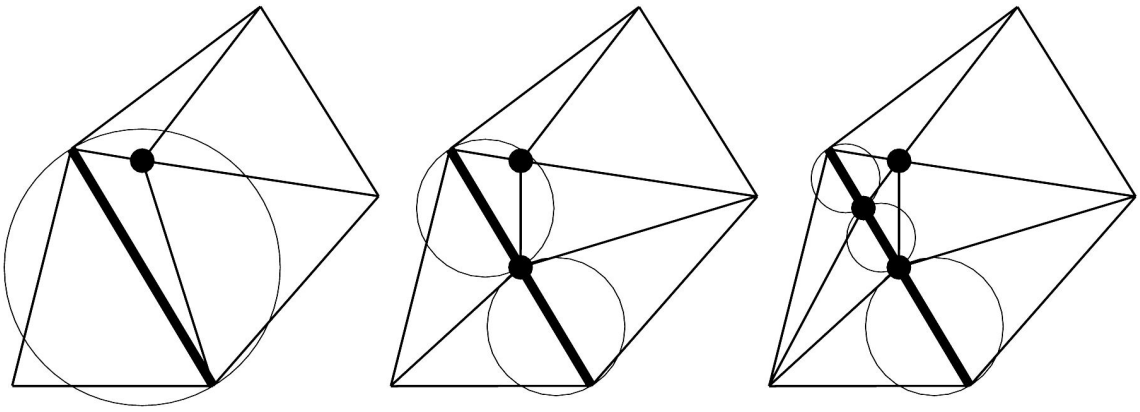


Fig. 3.4: O segmento do PSLG (em negrito) é dividido recursivamente até que seus círculos diametrais sejam vazios de pontos

.

- Recuperação de todas as arestas do PSLG que não aparecem na triangulação de Delaunay inicial, gerada a partir dos vértices do PSLG.

- Remover triângulos ruins através do refinamento.

Supondo-se que $\overline{ab}$ é um segmento que não aparece na triangulação de Delaunay, temos que o círculo diametral de $\overline{ab}$ contém algum vértice do polígono em seu interior. Nesse caso, o ponto médio de $\overline{ab}$ é inserido na triangulação, como ilustrado na Figura 3.4.

Suponha que t_{abc} seja um triângulo ruim. O circuncentro p de t_{abc} será incluído na triangulação se não invadir qualquer sub-segmento do PSLG. Se o circuncentro p for um invasor de um segmento s, então p não é inserido e o ponto médio de s é adicionado à triangulação.

De modo a manter a triangulação de Delaunay a cada inserção de vértice, a estratégia utilizada pelo algoritmo de Ruppert é a mesma estratégia utilizada pelo algoritmo de Flipping Incremental. Um algoritmo em alto nível é apresentado em seguida.

Pode-se observar que o número de triângulos produzidos pelo algoritmo de Chew (Figura 3.5) é tipicamente maior que o número de triângulos produzidos pelo algoritmo de Ruppert (Figura 3.6).

Com as definições básicas em mãos, é possível construir um algoritmo para Refinamento de Delaunay que respeite os padrões propostos por Ruppert. O procedimento pode ser visto no Algoritmo 2.

É possível mostrar que o algoritmo de Ruppert não insere pontos fora do *PSLG* e termina corretamente quando o PSLG não possui ângulos menores que 30^o.

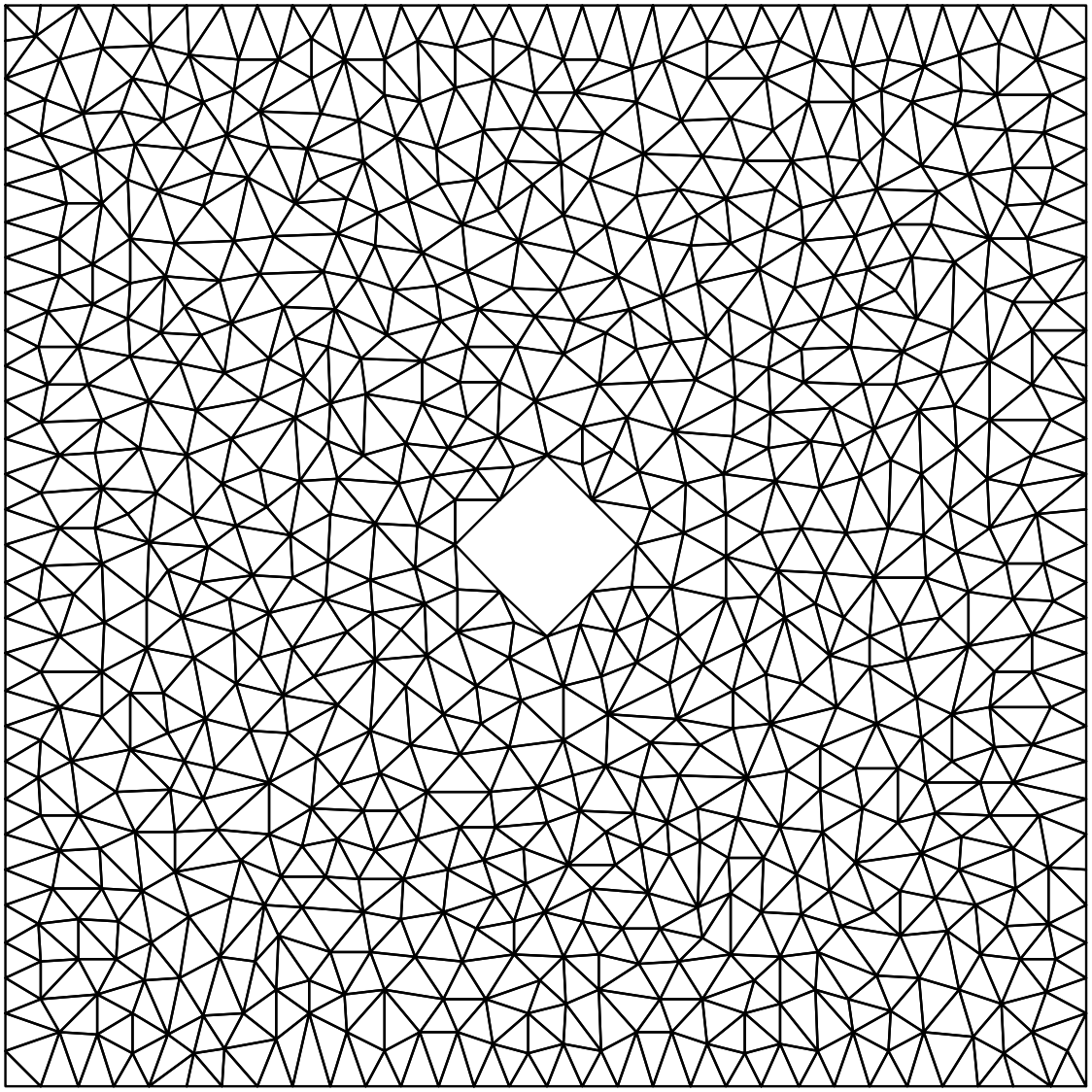


Fig. 3.5: Malha gerada pelo algoritmo de Chew

.

3.3.3 Abordagem para ângulos pequenos

A presença de um ângulo pequeno na entrada do algoritmo, neste caso num PSLG, requer um procedimento especial de modo a evitar que a malha resultante possua elementos de baixa qualidade, além dos restritos por ângulos pequenos de entrada. Sabendo que ângulos pequenos de entrada não podem ser removidos, o principal propósito é triangular um PSLG sem criar nenhum ângulo pequeno que já não esteja presente na entrada. Nenhum algoritmo fornece esta garantia para todos PSLG's. Descrevemos posteriormente a abordagem de Jim Ruppert para este problema. Outras propostas também foram elaboradas, como a descrita por Miller [18].

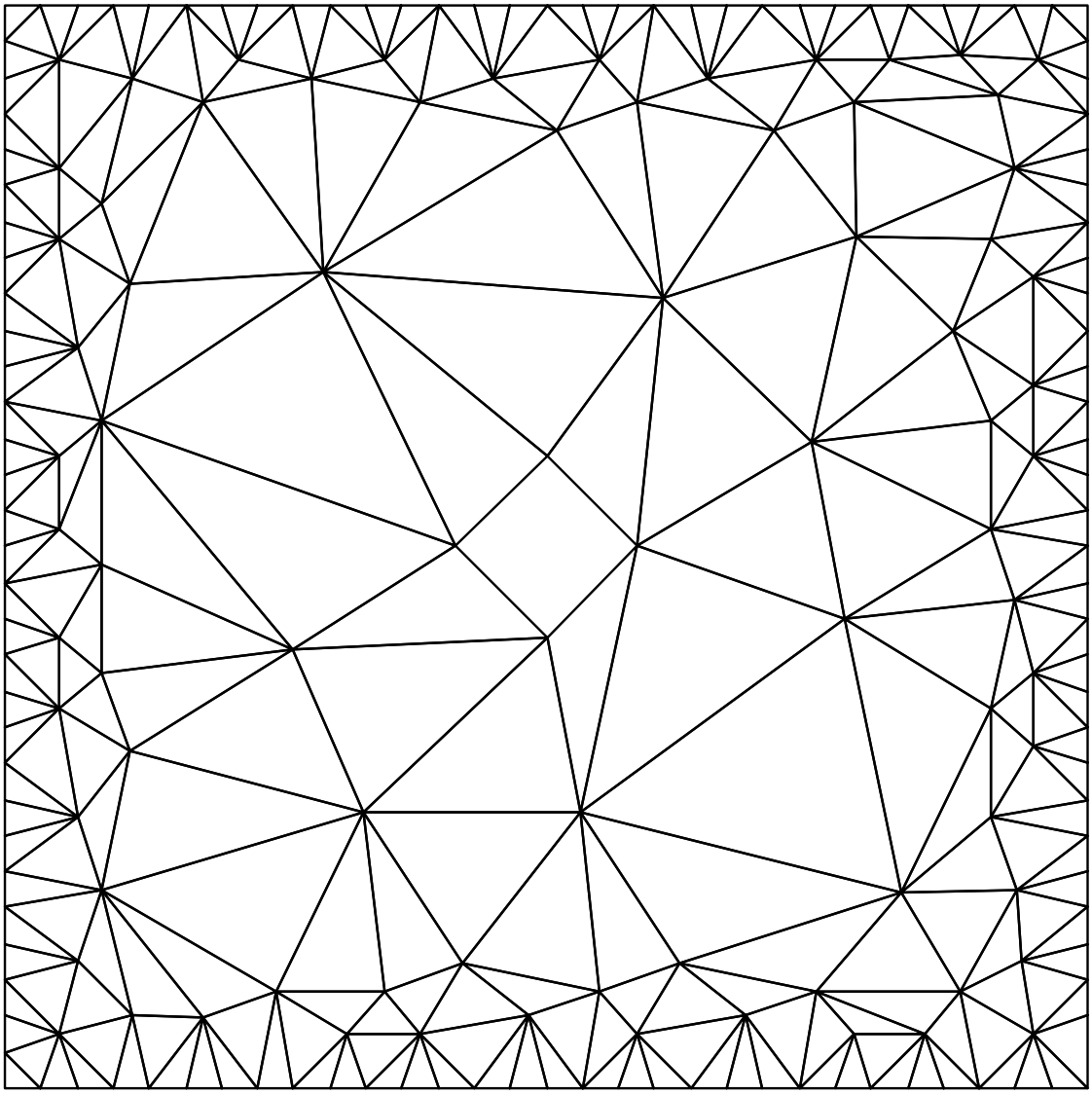


Fig. 3.6: Malha refinada pelo algoritmo de Ruppert

.

Para melhor compreensão deste problema necessitamos de duas importantes definições, vistas a seguir.

Definição 33 (Incidentes) *Dado um PSLG X, dois segmentos ou um vértice e um segmento de X são ditos incidentes se eles se intersectam em suas extremidades.*

Definição 34 (*Local Feature Size*) *Dado um PSLG X, o local feature size de um ponto p, $lfs(p)$, é o raio do menor disco centrado em p que intersecta dois vértices, dois segmentos ou um vértice e um segmento não incidentes de X.*

Algorithm 2 Algoritmo para refinamento de Ruppert

1: $G \leftarrow PSLG$
2: α : qualidade mínima
3: Gerar a triangulação de Delaunay T para os vértices de G
4: **Enquanto** Existir um segmento invadido $s \in G$ **Faça**
5: Dividir s inserindo um ponto médio
6: Atualizar G e T
7: **Fim Enquanto**
8: **Enquanto** existir algum triângulo $t \in T$ com qualidade menor que α **Faça**
9: Seja p o circuncentro de t, considerando que p invade os segmentos $s_1, s_2, \ldots, s_k$
10: **Se** $k \geq 1$ **Então**
11: Dividir em dois os segmentos $s_1, s_2, \ldots, s_k$ inserindo os k pontos médios.
12: Atualizar G e T.
13: **Senão**
14: Insere p
15: **Fim Se**
16: **Fim Enquanto**

A Figura 3.7 ilustra o *local feature size* de diferentes pontos, onde o raio do disco D_i é $lfs(p_i)$.

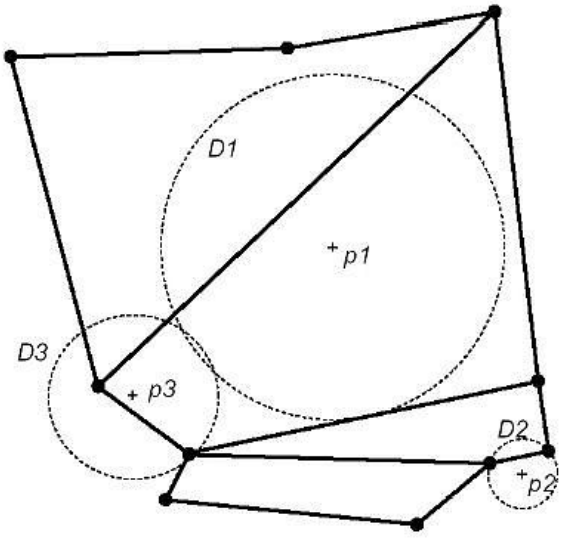


Fig. 3.7: *Local Feature Size* de alguns pontos

Lema 8 *Para algum PSLG X, e quaisquer dois pontos u e v no plano, $lfs(v) \leq lfs(u) + |uv|$.*

Abordagem de Ruppert para ângulos pequenos

Para triangular algum PSLG sem introdução adicional de ângulos pequenos, Ruppert sugere que um vértice no canto de um ângulo pequeno pode ser coberto com uma faixa fina de triângulos bem formados, ou seja, triângulos que possuem todos os ângulos maiores que 30^o, exceto o triângulo com vértice no ângulo pequeno. Esta técnica utiliza *shield edge* [25] como explicado a seguir.

Primeiramente realiza-se o cálculo do *local feature size* para todo vértice de entrada v. Todo vértice v com ângulo pequeno é cercado com um círculo que intersecta as arestas de entrada, como mostra a Figura 3.8. O raio do círculo é $\frac{lfs(v)}{3}$, assim os círculos ao redor de vértices diferentes não se intersectam e não estão muito próximos.

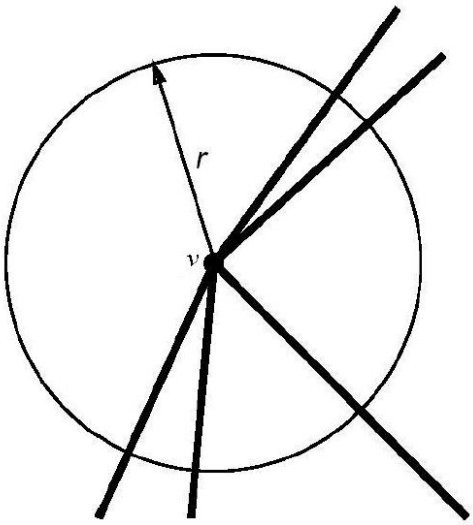


Fig. 3.8: Círculo ao redor do vértice de entrada.

Sendo α o menor ângulo de entrada do PSLG, os ângulos em v maiores que 2α são divididos para que fiquem entre α e 2α, introduzindo *shield edges* ao redor do círculo, e *spoke edges* de v para o círculo, como mostra a Figura 3.9. Estas arestas e os novos vértices se tornam parte do PSLG de entrada, aparecendo na malha de saída e reduzindo o *local feature size* no PSLG.

Cada *shield edge* é dividida, havendo para isto duas opções:

- Colocam-se arestas entre os vértices que dividiram a *shield edge* e v, como mostra a Figura 3.10.

- Na segunda opção, a idéia é que a *shield edge* seja dividida um número constante de vezes, e pode-se usar o número constante de camadas para preencher com triângulos, Figura 3.11. Esta construção é mais complicada, mas pode evitar que o menor de todos os ângulos de entrada seja dividido.

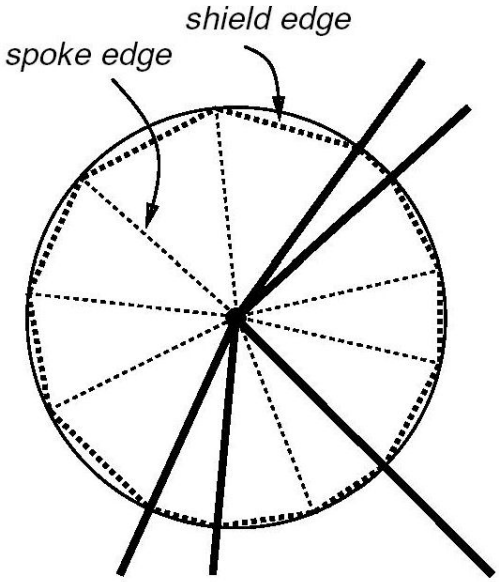


Fig. 3.9: Ilustração das *shield edges* e *spoke edges*.

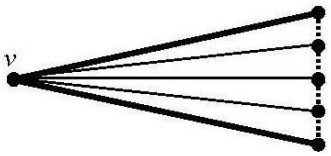


Fig. 3.10: Primeiro método de divisão da shield edge.

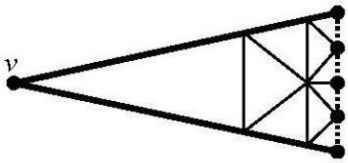


Fig. 3.11: Segundo método de divisão da shield edge.

Assim, a região dentro das *shield edge* é triangulada por uma seqüência finita de tiras similares, com cada tira sucessivamente menor que a tira anterior por um fator constante perto de 1. Essas tiras são finas e compostas por triângulos bem formados, Figura 3.12 [28].

Segundo Shewchuk [28], Ruppert afirma de forma errônea que a região escondida atrás da *shield edge* sempre tem uma triangulação finita de boa qualidade.

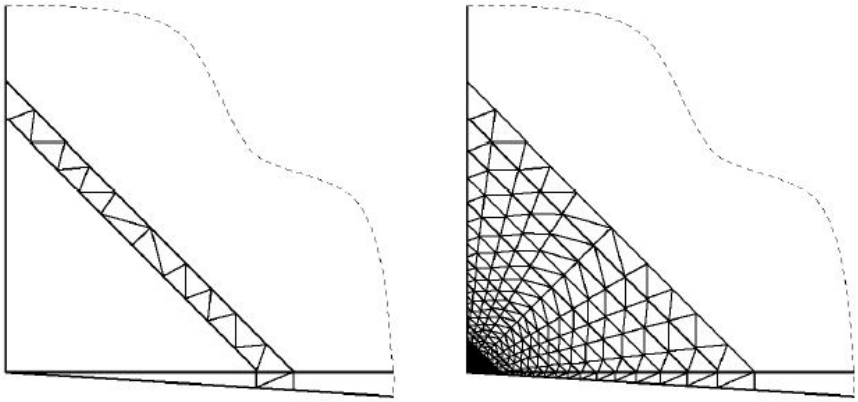


Fig. 3.12: Triangulação da região dentro de shield, com uma seqüência finita de tiras similares sucessivamente menores de triângulos bem formados.

Outras Abordagens

Uma alternativa sugerida por Shewchuk [28] baseia-se no *Quitter*, um algoritmo para refinamento de Delaunay que fornece flexibilidade e que possui término garantido.

O *Quitter* baseia-se no refinamento de Delaunay com círculos concêntricos. Quando um segmento s é invadido pelo circuncentro de um triângulo fino, toma-se uma decisão entre dividir ou não o segmento s com um vértice v.

Miller [18], propõe uma alteração do algoritmo de Ruppert que gera uma malha Delaunay com todos os ângulos maiores que 26.45^o, exceto os ângulos dos triângulos cuja menor aresta é oposta a um ângulo de entrada $\theta \leq 36.53^o$. Neste caso, o ângulo de saída será maior do que $\arctan \frac{\sin\theta}{2-\cos\theta}$, onde θ é o ângulo mínimo de entrada. Todos os triângulos da malha gerada pelo algoritmo proposto por Miller possuem ângulos menores que 137.1^o.

3.4 Método do Equilíbrio de Forças

Um método alternativo de geração de malhas adaptativas de Delaunay foi proposto por Per-Olof Persson e Gilbert Strang [23]. O objetivo principal desta nova abordagem é a simplificação do código do gerador, tornando seu uso mais simples tanto para uma simples geração de malhas quanto para a integração do gerador com outros projetos.

Originalmente o método foi construído utilizando-se como base o software *Matlab*, permitindo que o gerador fosse implementado com poucas linhas de código.

O domínio a ser triangulado é definido através de uma função implícita contínua $f(x, y)$. Tais

funções também podem ser combinadas para formar domínios mais complexos através de operações de união, intersecção e subtração. Consideremos como exemplo a função definida por $f(x, y) = \sqrt{x^2 + y^2} - 1$. Esta função define um domínio circular de raio unitário, ilustrado na figura 3.13.

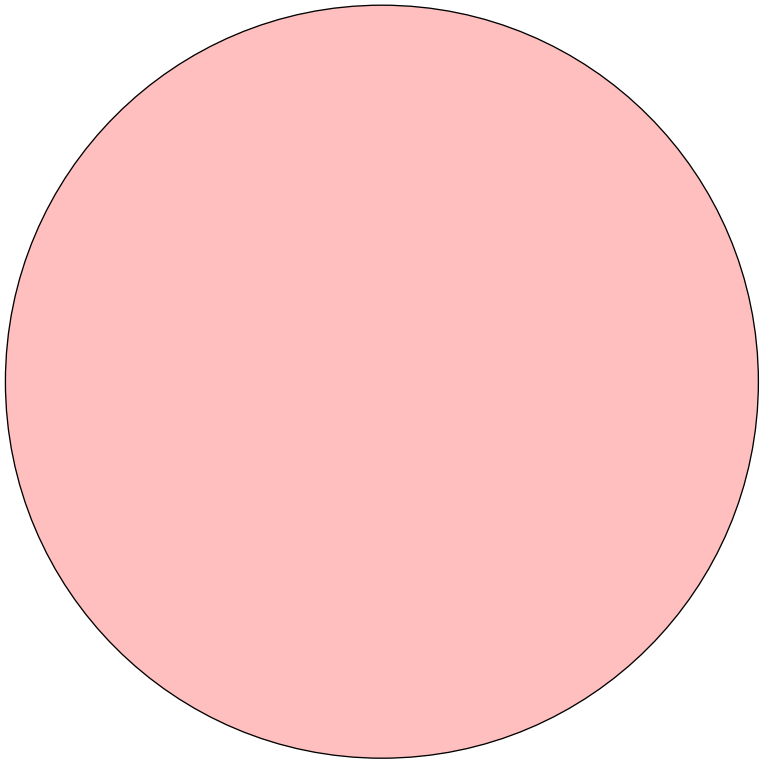


Fig. 3.13: Função distância definindo um domínio circular

Utilizando as operações booleanas, é possível formar um domínio através da combinação de duas funções. Tomemos a primeira função como $f_1(x, y) = \sqrt{x^2 + y^2} - 1$, o mesmo círculo unitário da figura 3.13. Tomemos a segunda função como $f_2(x, y) = \sqrt{x^2 + y^2} - \sqrt{0.4}$, um círculo de raio 0.4. Combinaremos as funções para obter uma nova função $f_3(x, y) = f_1(x, y) - f_2(x, y)$. Ou seja, temos uma circunferência de raio unitário com um buraco de raio 0.4, como mostra a figura 3.14.

O conceito base do método é a modelagem da malha como uma estrutura de molas. Os vértices da malha são os elos de ligação enquanto as arestas são as molas que fazem a conexão dos elos. Cada mola apresenta uma força associada que propicia tanto o crescimento como o encurtamento de seu comprimento. Considerando um conjunto de molas, as forças atrativas e repulsivas somam-se para uma força resultante na estrutura e, a cada iteração, resolve-se o sistema para o equilíbrio. As forças das molas movem os elos de ligação, distribuindo os pontos através do domínio.

A combinação da função distância com o movimento dos elos apresenta grande vantagem pois através desta função é possível determinar rapidamente se um vértice está dentro ou fora do domínio. Se o ponto se encontra fora do domínio, um cálculo imediato utilizando-se $f(x, y)$ pode determinar

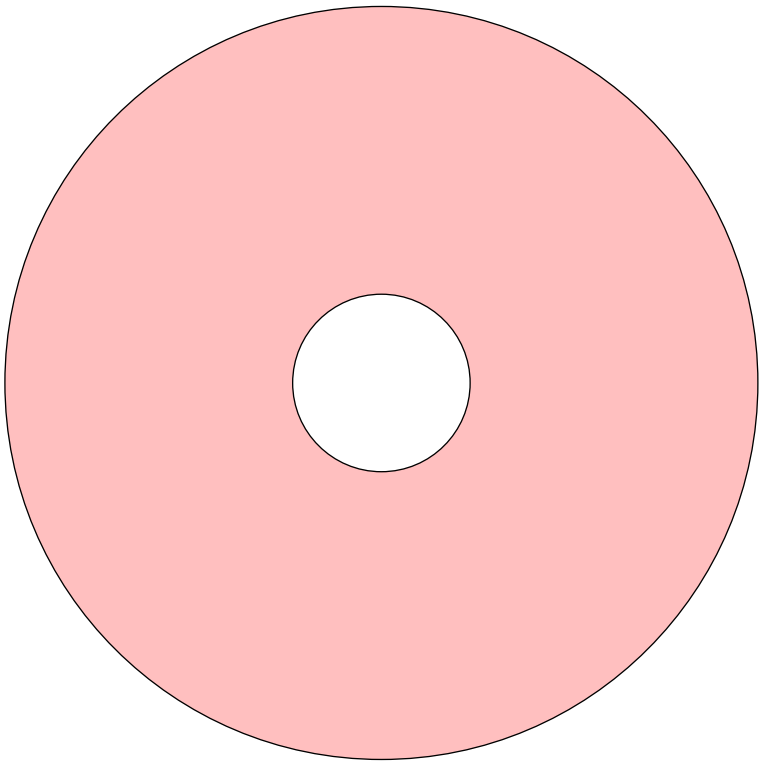


Fig. 3.14: Combinando funções implícitas para formar domínios complexos

rapidamente, através da projeção do gradiente, o ponto mais próximo do bordo do domínio.

Apesar da simplicidade, tal abordagem tem como resultado malhas de ótima qualidade, mesmo se comparadas às técnicas tradicionais de refinamento de Delaunay.

Considerando a analogia da malha com uma estrutura de molas, cada uma das molas tem uma força interna associada baseada em seu comprimento atual e seu comprimento não estendido. Temos como exemplo as malhas ilustradas nas figuras 3.15 e 3.16.

As forças externas vêm através dos bordos. Em cada ponto do bordo há uma reação contrária agindo de forma normal nos pontos. Consideramos a magnitude desta reação grande o suficiente para que mantenha todos os pontos dentro do domínio.

O objetivo do método é o cálculo das posições dos pontos nodais, que são a principal incógnita do problema. Para resolver este sistema, organizam-se as coordenadas cartesianas de todos os vértices em um vetor $N \times 2$, onde as coordenadas horizontais ficam na primeira coluna e as coordenadas verticais na segunda coluna.

O vetor força tem componentes horizontais e verticais para cada vértice da malha, sendo constituído de forças internas provenientes das barras (molas) e as forças externas provenientes da reação dos bordos, como pode ser visto na equação 3.1.

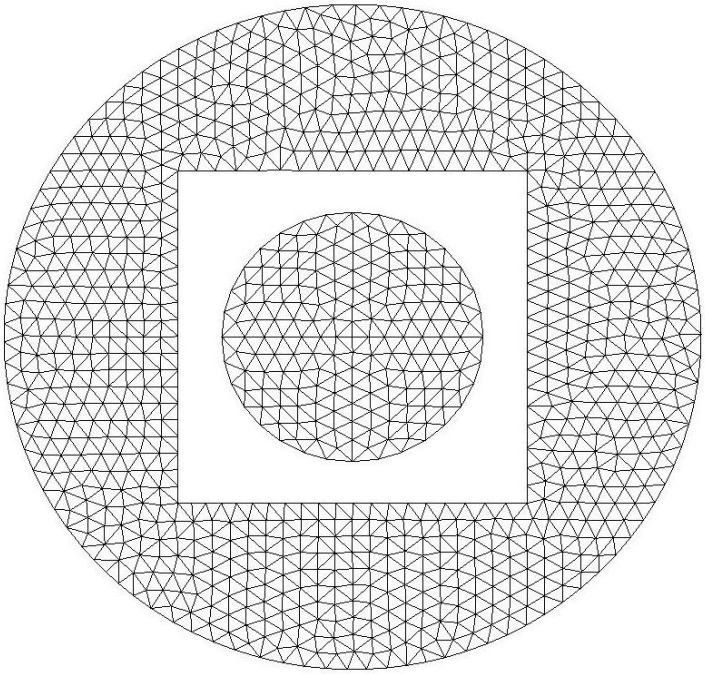


Fig. 3.15: Qualidade da malha gerada pelo método

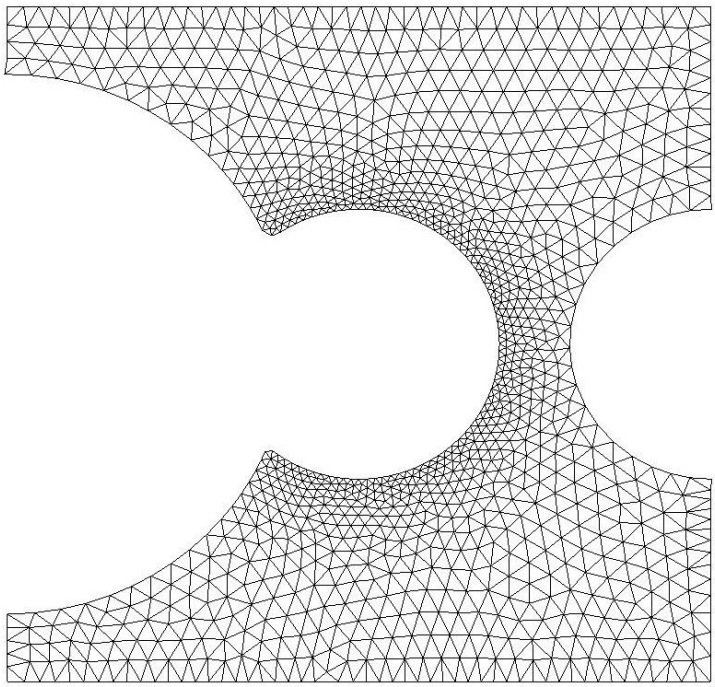


Fig. 3.16: Malha sobre um domínio gerado por combinações de funções distância

$$F(p) = [F_{int,x}(p), F_{int,y}(p)] + [F_{ext,x}(p), F_{ext,y}(p)] \tag{3.1}$$

As forças dependem diretamente da topologia da estrutura de molas. Esta topologia é obtida através de uma triangulação de Delaunay, que é gerada com base nos pontos de entrada. Nota-se também que o vetor de forças $F(p)$ não é uma função contínua sobre p visto que a topologia pode ser alterada pela triangulação de Delaunay de acordo com as posições dos pontos nodais.

O sistema $F(p) = 0$ deve ser resolvido para um conjunto de posições de equilíbrio p. Este problema é relativamente complexo em parte devido à descontinuidade na função $F(p)$ e parte devido às reações externas dos bordos.

Uma abordagem simples para a solução deste problema é considerar um elemento temporal artificial. Para algum $p(0) = p_0$ consideramos um sistema de equações diferenciais:

$$\frac{dp}{dt} = F(p), t \geq 0 \tag{3.2}$$

Se uma situação de equilíbrio for encontrada, a solução para a equação diferencial (3.2) será uma solução para o sistema $F(p) = 0$. O sistema 3.2 pode ser aproximado através do método de Euler. Temos que no instante de tempo $t_n = n\Delta t$ a solução aproximada $p_n \approx p(t_n)$ é atualizada através da equação 3.3.

$$p_{n+1} = p_n + \Delta t F(p_n) \tag{3.3}$$

Durante o cálculo da função força $F(p)$, as posições dos pontos p_n são conhecidas assim como a topologia da estrutura, devido à triangulação de Delaunay do conjunto de pontos inicial.

As forças de reação externas são tratadas da seguinte maneira: Todos os pontos que venham a ser movidos para fora do domínio de triangulação durante uma atualização de posições de p_n para p_{n+1} são trazidos de volta ao ponto do bordo mais próximo. Isto está em conformidade com a ação das forças na direção normal ao bordo. Os pontos podem ser movidos ao longo do bordo, porém nunca para fora dele.

A função força é modelada de maneira a permitir somente forças repulsivas, não permitindo porém as forças atrativas, onde a constante k têm um papel de correção de unidade. Apresentamos a função força em cada barra (aresta) na equação 3.4.

$$f(l, l_0) = \begin{cases} 0, & l \geq l_0 \\ k(l_0 - l), & l < l_0 \end{cases} \tag{3.4}$$

O tratamento dos bordos apresentado neste método visa espalhar os pontos ao longo de todo o domínio. Para que isto aconteça é importante que as barras tenham forças repulsivas $f > 0$. Isto

implica que a função $f(l, l_0)$ deve ser positiva quando l é próximo ao comprimento desejado das arestas, o que pode ser feito com a escolha de um comprimento l_0 um pouco maior de que o realmente desejado. Resultados empíricos mostram que um comprimento 20% maior traz bons resultados.

Para malhas uniformes temos l_0 constante, porém há casos onde um comportamento não uniforme é preferido, em especial nos casos onde as regiões possuam áreas que requerem maior precisão nos cálculos.

O comprimento das arestas é fornecido ao método através de uma função comprimento $h(x, y)$. É importante ressaltar que a função $h(x, y)$ não representa o comprimento absoluto de uma aresta e sim seu comprimento relativo. Desta maneira evitamos uma relação implícita entre o número de nós, que não é pedido ao usuário como entrada, e o comprimento das arestas.

Considerando a função $h(x, y) = 1 + x$ para um domínio de um quadrado unitário, temos que os comprimentos das arestas próximas à região esquerda do bordo (x = 0) serão a metade dos comprimentos das arestas próximas à região direita (x = 1). Este comportamento é consistente e vale para qualquer número de pontos.

Devido à natureza relativa dos comprimentos das arestas através da função $h(x, y)$, trabalha-se com um valor de escala, que é calculado com a fórmula 3.5, onde l_i faz parte da coleção de todas as arestas.

$$S = \sqrt{\frac{\sum l_i^2}{\sum h(x_i, y_i)^2}}, \tag{3.5}$$

Tendo em mente a facilidade de operação do método, a função $h(x, y)$ é definida pelo usuário como um dado de entrada para o gerador.

A distribuição inicial de pontos p_0 pode ser feita de várias maneiras. Por exemplo, é possível fazer uma distribuição aleatória de pontos ao longo do domínio, ou então distribuir os pontos uniformemente, isto é, com pontos igualmente espaçados.

Dependendo do tipo de domínio, um tipo de distribuição pode ser mais vantanjoso do que outro. Recomenda-se a distribuição uniforme para malhas que terão elementos de tamanhos semelhantes ou que apresentem geometria mais simples. A geração de malhas com elementos de tamanhos variados tem convergência mais rápida quando se utiliza uma probabilidade para descartar pontos com base na função $h(x, y)$. Esta probabilidade é dada por $\frac{1}{h(x,y)^2}$.

3.4.1 Algoritmo

O algoritmo em alto nível para o método do equilíbrio de forças é descrito a seguir, juntamente com explicações para cada passo importante.

Algorithm 3 Método do Equilíbrio de Forças

```
1: Criar distribuição inicial de pontos no bounding box do domínio
2: Remover pontos fora do domínio
3: Triangulação de Delaunay inicial
4: Enquanto Movimentação dos pontos || Qualidade ruim Faça
5:     Se houve movimento considerável entre os pontos Então
6:         Salvar posições dos pontos antigos
7:         Triangulação de Delaunay dos pontos
8:         Calcular os centróides dos triângulos
9:         Remover os triângulos fora do domínio
10:    Fim Se
11:    Calcular os comprimento atuais das arestas
12:    Calcular os comprimentos desejados
13:    Calcular o vetor de forças
14:    Aplicar força e atualizar as posições dos pontos
15:    Trazer pontos externos para dentro do domínio
16: Fim Enquanto
```

A primeita etapa do método é a distribuição inicial de pontos ao longo do domínio. Neste passo os pontos podem ser distribuídos de duas maneiras: Uniforme e Aleatória.

A distribuição uniforme utiliza como base um valor de espaçamento h_0 entre os pontos, tanto na vertical como na horizontal. Dado este espaçamento, os pontos são posicionados ao longo de um quadrilátero que contém o domínio especificado.

A distribuição aleatória gera coordenadas randômicas dentro do quadrilátero que contém o domínio, e utiliza como base um número n de pontos que é fornecido como entrada para o método. Neste caso, o número de pontos é pedido pois não é possível, de uma maneira trivial, determinar o espaçamento entre todos os pontos para garantir que atingimos uma certa distância mínima entre todos os pontos, ao contrário da distribuição uniforme, onde temos uma única distância para toda a distribuição.

A segunda etapa do algoritmo remove os pontos que se posicionam entre o domínio e o quadrilátero que contém o domínio. Desta forma garantimos que todos os pontos estarão dentro dos bordos do domínio especificado.

A seguir o método gera uma triangulação inicial no domínio, de modo a dar início ao método.

A partir deste ponto, o algoritmo entra em um ciclo iterativo de melhoramento da malha, buscando o equilíbrio dos pontos. Inicialmente, o comprimento de cada aresta é calculado, assim como os comprimentos desejados. Com base nos comprimentos atual e final, calcula-se a força resultante em cada aresta.

Com base nos dados de força resultante de cada aresta, aplicamos estas forças ao sistema e movimentamos assim os pontos, de modo a distruibuí-los ao longo do domínio.

Eventualmente, os pontos podem vir a sair do domínio, devido às forças aplicadas. Devido a esta possibilidade, aplicamos uma força contrária com base no gradiente da função implícita de maneira a trazer tais pontos de volta aos bordos do domínio.

Com as posições dos pontos atualizadas, verificamos o deslocamento interno devido à movimentação geral do sistema. Se este valor estiver dentro de um intervalo considerado, repetimos a iteração para que haja um novo ajuste da posição dos pontos. Quando este deslocamento for menor que o intervalo considerado, o método chega ao fim.

Da etapa 5 até a etapa 9 do algoritmo 3, temos a reconstrução da malha para refletir a mudança da posição dos pontos e possivelmente da topologia, juntamente com a remoção de triângulos que possuem centróides fora do domínio especificado. Desta maneira temos uma malha restrita apenas aos bordos do domínio. Os centróides são utilizados como uma boa aproximação para sabermos quando um determinado triângulo saiu do domínio.

Capítulo 4

Implementações

O aumento do volume de dados exige que as estruturas de dados dedicadas à representação de malhas não estruturadas sejam capazes de manipular informações de maneira eficiente e compacta, ou seja, com o mínimo de memória possível. Neste quesito, as estruturas de dados topológicas, que foram originalmente concebidas por Baumgart [3], possuem um comportamento muito bom, demonstrando robustez no tratamento de malhas simpliciais [21, 20]. Outras estruturas utilizadas na manipulação de malhas incluem a *winged-edge* [3], *quad-edge* [13] e a *half-edge* [16].

Neste capítulo apresentamos uma nova estrutura de dados, desenvolvida no LCAD (Laboratório de Computação de Alto Desempenho), denominada *OF* (*Opposite Face*), bem como uma breve explicação de suas principais características. Esta estrutura de dados substitui a estrutura anterior denominada SHE. A SHE tinha sua base na *half-edge*, concebida por Mäntyla [16].

A estrutura de dados *OF* tem como base a estrutura denominada *Corner Table* e a biblioteca de Geometria Computacional CGAL [6]. A *Corner Table* foi originalmente concebida por Jarek Rossignac [24] para uso conjunto com um algoritmo de compressão de malhas, desenvolvido pelo próprio autor. As principais diferenças da estrutura OF em relação à *Corner Table* estão nas representações de bordo, isentas na estrutura original, e um suporte aprimorado a malhas dinâmicas, ou seja, malhas onde operações de inserção e remoção são constantemente requisitadas, demandando capacidade de reordenação em um tempo baixo.

4.1 Estrutura de dados *OF*

A estrutura *OF* trabalha com representações explícitas das primitivas como vértices e células. As células podem ser triângulos, no caso bidimensional, ou tetraedros, no caso tridimensional. As demais primitivas são representadas implicitamente, podendo ser obtidas através do relacionamento entre os vértices e as células.

Esta abordagem permite maior simplicidade na manipulação da malha de triângulos e também reduz o espaço de armazenamento, já que as primitivas não existem fisicamente e sim logicamente. Os elementos são definidos por classes e o acesso às primitivas é feito através de um identificador numérico único, associado aos vértices e células.

As operações básicas da estrutura são inserções e remoções de elementos (tanto vértices quanto células) e consulta de informações referentes à topologia da malha.

A maioria das classes da estrutura *OF* precisa ser instanciada com um ou mais parâmetros passados por template. O principal parâmetro é um conjunto de tipos básicos (perfil) da estrutura, cuja função é definir as características da malha alocada. Cada classe necessita de apenas alguns dos tipos disponíveis no perfil, mas por praticidade, podemos criar um único perfil, com todos os tipos que serão utilizados para o tipo de malha que desejamos. Para tanto, basta atribuí-lo a qualquer objeto específico desta malha.

Nos vértices temos as únicas informações geométricas da malha. Cada vértice está associado a uma posição do espaço que é a base das operações que envolvem cálculos geométricos. Nos vértices também é armazenada uma referência para uma célula que o contém. No caso de vértices singulares, uma referência é armazenada para cada célula de cada componente que constitui uma singularidade. Com esta informação, todas as componentes da singularidade são acessíveis através de um vértice, conseqüentemente tornando a malha totalmente acessível.

As células representam os triângulos ou tetraedros (ou simplexos de dimensões superiores) da malha, dependendo apenas da dimensão atribuída. Em cada célula são armazenadas referências para todos os vértices que a formam (três, quatro ou mais) e também para todas as células vizinhas por arestas ou faces. Os parâmetros templates necessários para a instanciação da classe *ofCell* são o perfil de tipos e a dimensão da célula, ou seja, 2 para os triângulos e 3 para os tetraedros.

A estrutura *OF* concentra todos os cálculos geométricos mais utilizados na classe *ofGeometric*. O propósito desta classe é tornar flexível a alteração dos cálculos realizados pela estrutura e pelos algoritmos implementados sobre a *OF*. Esta classe precisa ser especificada no perfil de tipos, podendo ser substituída por outra classe com a mesma interface. Tal recurso é importante para a utilização de primitivas geométricas calculadas através de aritmética exata. Vale ressaltar que a estrutura *OF* só realiza cálculos geométricos nas malhas que exigem células orientadas.

Alguns padrões de acesso a malhas são freqüentemente utilizados, como a obtenção da estrela de um vértice, ou então caminhar por todas as células ou vértices da malha. Visando a simplificação e padronização do acesso aos elementos que possuem alguma relação entre si, a estrutura possui uma classe de iteradores (*ofIterator*).

É possível carregar malhas diretamente de arquivos para a estrutura *OF* por meio das classes herdadas de *ofReader*. Atualmente, existem leitores de arquivo no formato VTK [26] (unstructured

grid e polydata), OFF e WRL.

Na estrutura *OF* encontramos um conjunto de estruturas simples utilizadas em implementações gerais (Listas, Vetores, Árvores e Pilhas). Seguindo o mesmo padrão de programação da *OF*, estas estruturas precisam ser instanciadas com o tipo de objeto que será armazenado.

4.2 Algoritmos de triangulação

A estrutura OF implementa dois métodos de geração de malhas, sendo elas a triangulação de Delaunay e o método de Refinamento de Ruppert. Ambas implementações são para casos em duas dimensões, sendo que o método de Ruppert aceita restrições.

A seguir descrevemos melhor estes métodos.

4.2.1 Triangulação de Delaunay

O algoritmo implementado com base na estrutura *OF* é aplicado a uma malha instanciada com um perfil bidimensional utilizando vértices orientados. A abordagem utilizada foi a mesma do método de Flipping Incremental.

Há uma opção para possível aumento de desempenho durante a construção da malha. Este aumento de desempenho é conseguido através da indexação de vértices em uma estrutura de árvore chamada *kd-tree* (*ofKdTree*).

Os vértices podem ser fornecidos ao algoritmo através do construtor, passando uma malha contendo apenas os vértices, ou então é possível adicionar os vértices um a um, utilizando o método *add vertex*. Desta maneira os vértices podem ser adicionados à malha quando necessário.

4.2.2 Refinamento de Ruppert

O algoritmo de refinamento de Ruppert bidimensional também está disponível para a estrutura de dados *OF*.

A entrada do algoritmo é uma lista de polígonos representando o bordo do domínio (PSLG). Estes polígonos possuem os bordos orientados no sentido anti-horário quando são externos e bordos orientados no sentido horário quando são internos.

A leitura de arquivos CGM foi implementada para que fosse possível receber arquivos criados pelo aplicativo XFIG, presente em várias distribuições do Linux.

O valor que define a qualidade da malha no algoritmo de refinamento de Ruppert deve ser fornecido no início da execução do método.

Na Figura 4.1 podemos observar o contorno inicial que servirá como base para a geração da malha. A Figura 4.2 ilustra o contorno já com a malha construída e refinada pelo algoritmo de Ruppert.

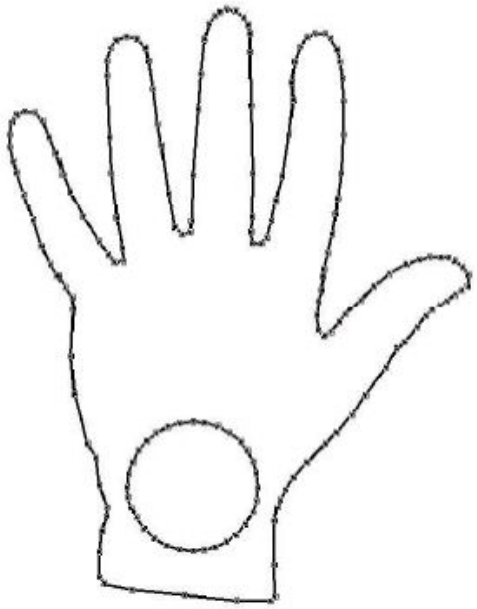


Fig. 4.1: Contorno descrito em PSLG

.

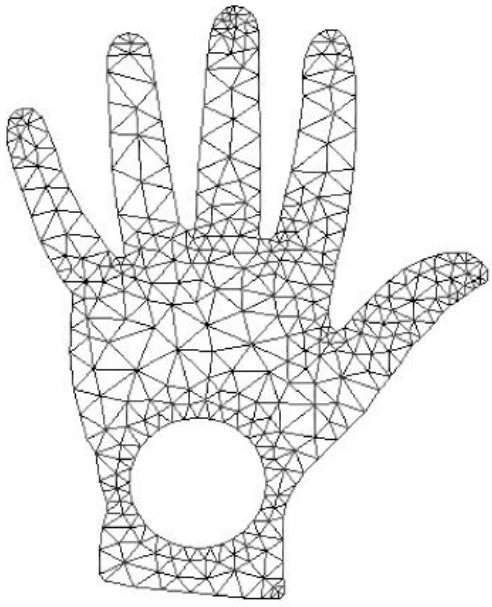


Fig. 4.2: Malha construída através do método de refinamento de Ruppert

.

4.2.3 Método do equilíbrio de forças

Utilizando como base a estrutura de dados OF, apresentada anteriormente, implementou-se o método do equilíbrio de forças em linguagem C/C++. Apresentamos em detalhes as estruturas que compoem o gerador de malhas bem como exemplos de funcionamento e dados de desempenho.

O método baseia-se em uma analogia com um sistema físico em equilíbrio de forças, sendo que a resultante das forças faz com que os vários pontos dentro de um domínio sejam reposicionados até o equilíbrio. Descrevemos primeiramente as funções distância.

Funções distância

Para a obtenção das funções distância implícitas que definem um domínio foram implementadas primitivas. Consideramos as primitivas como funções distância simples que podem ser utilizadas como base para a construção de domínios mais complexos.

O código implementado considerou as seguintes primitivas (código 4):

- **Retângulo**: Primitiva que retorna a distância mínima de um determinado ponto $p = (x, y)$ às arestas de um retângulo definido pelos pontos $p_1 = (x_0, y_0)$, $p_2 = (x_1, y_0)$, $p_3 = (x_1, y_1)$ e $p_4 = (x_0, y_1)$.

- **Círculo**: Primitiva que retorna a distância mínima de um determinado ponto $p = (x, y)$ a uma circunferência de raio r e centro $p_c = (x_c, y_c)$.

A combinação das primitivas pode ser feita através de funções auxiliares ditas booleanas. Entre estas funções temos a adição, subtração e interseção. A translação e rotação também podem ser utilizadas para alterar o resultado final, gerando uma função distância de complexidade maior.

Considerando duas funções distância, d_1 e d_2, quaisquer, podendo estas serem uma das primitivas anteriormente expostas ou funções complexas, aplicamos uma função booleana de adição entre d_1 e d_2. A função distância resultante é $d_{add} = min(d_1, d_2)$. De maneira similar, aplicando as funções booleanas de subtração e interseção, respectivamente, temos as funções resultantes $d_{sub} = max(d_1, -d_2)$ e $d_{inter} = max(d_1, d_2)$. As funções complexas podem ser novamente recombinadas através das funções booleanas para obter o domínio desejado.

As funções de translação e rotação também podem ser aplicadas para os ajustes necessários.

Primitivas discretas como triângulos, hexágonos e outros polígonos definidos por segmentos de reta também podem ser utilizadas durante a geração da função distância, sendo que estas primitivas necessitam de funções capazes de calcular a distância mínima de um ponto $p = (x, y)$ até um determinado polígono $P = (p_0, p_1, \ldots, p_n)$.

Algorithm 4 Funções distância primitivas

```
1: double dcircle(double x, double y, double xc, double yc, double r)
2: {
3: return (sqrt(pow(x - xc,2) + pow(y - yc,2)) - r);
4: }
5:
6: double drectangle(double x, double y, double x1, double y1, double x2, double y2)
7: {
8: return( - min (min( min(-y1 + y, y2 - y), -x1 + x), x2 - x) );
9: }
```

Função razão de comprimento das arestas

A função razão de comprimento estipula valores relativos para o comprimento das arestas dos triângulos da malha. O controle desta razão interfere na aparência da malha, alterando tanto a concentração de pontos ao longo do domínio quanto o equilíbrio de forças no sistema. A falta de equilíbrio no sistema acaba provocando deslocamentos de pontos até que estes venham a atingir o equilíbrio novamente.

O usuário pode alterar a função razão de acordo com suas necessidades por meio da alteração do código. Duas funções são ilustradas no código 5, uma função razão variável que propicia a criação de elementos de tamanho variável e uma função razão constante que gera elementos de mesmo tamanho.

Algorithm 5 Funções razão de comprimento

```
1: double variable-length(double x, double y)
2: {
3: return fabs(4 * sqrt(x*x + y*y)) + 1;
4: }
5:
6: double uniform-length(double x, double y)
7: {
8: return 1;
9: }
```

Classe BarObject

A classe BarObject representa uma única aresta (segmento de reta) na malha e é uma das principais em termos de armazenamento dos dados. Todas as informações referentes a coordenadas, comprimento e forças estão disponíveis nesta classe. Tais informações são utilizadas frequentemente durante a iteração do método, visto que as forças resultantes estão sempre variando e, portanto, os comprimentos das arestas também, requerendo atualizações em todas as arestas da malha.

A definição desta classe pode ser vista no código 6 e uma breve explicação de seus atributos e membros é fornecida a seguir.

- **Construtores/Destrutores**: Encarregam-se, respectivamente, de alocar a memória necessária para os atributos da classe quando esta é instanciada e de desalocar a memória utilizada quando um objeto é liberado.
- **p1, p2**: Vértices $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ que representam os pontos extremos de um segmento de reta/aresta.
- **idp1, idp2**: Índices referentes, respectivamente, aos vértices p1 e p2. Estes índices são utilizados para acessar os respectivos vértices diretamente através da estrutura OF.
- **barvector**: Vetor descrevendo o segmento de reta. É calculado pela diferença entre p1 e p2.
- **forcevector**: Vetor força resultante na respectiva aresta, obtido através da multiplicação do valor escalar de força pelo vetor *barvector*.
- **l**: Comprimento atual da aresta.
- l_0: Comprimento estipulado da aresta. Este valor deve ser atingido, aproximadamente, quando o método chega ao fim e é estipulado pelo usuário.
- **h**: Valor da função razão de comprimento no ponto médio da respectiva aresta.
- **force**: Módulo da força resultante na respectiva aresta.
- **Funções SET**: Métodos utilizados para armazenamento de dados no objeto.
- **Funções GET**: Métodos utilizados para acessar informações dos objetos.

Classe FEDelaunay

A classe FEDelaunay é a principal do gerador de malhas por equilíbrio de forças. Esta classe contém os métodos responsáveis por gerar a malha através de um conjunto de funções distância bem como métodos auxiliares de distribuição de pontos, cálculo de forças entre outros. O processamento foi centralizado na classe FEDelaunay, consequentemente facilitando o uso do algoritmo em outros projetos, bastando uma simples inicialização da estrutura.

A estrutura da classe pode ser vista no código 7. Em seguida estabelecemos os papéis de cada um dos atributos e métodos da classe.

Algorithm 6 Classe Bar Object

```
1: typedef class BarObject
2: {
3: public:
4:
5: Vertex p1, p2;
6: int idp1, idp2;
7: Vector barvector;
8: Vector forcevector;
9: double l;
10: double l0;
11: double h;
12: double force;
13:
14: BarObject(Vertex *pointa, Vertex *pointb);
15: B̃arObject();
16: void setLength(double length);
17: void setDesiredLength(double length);
18: void setHValue(double value);
19: void setForce(double value);
20: void setForceVector(double value);
21:
22: double getVectorX();
23: double getVectorY();
24: double getLength();
25: double getDesiredLength();
26: double getHValue();
27: double getForce();
28: }Bar;
```

Algorithm 7 Classe FEDelaunay

```
1: typedef class FEDelaunay
2: {
3: double dptol;
4: double ttol;
5: double Fscale;
6: double geps;
7: double deps;
8: double h0;
9: double deltat;
10: bbox limits;
11: int nvertex;
12: BarList bars;
13: IMap barlist;
14: VertexList *new-points;
15: VertexList *old-points;
16: VertexList *fixed-points;
17: distance-function distance;
18: length-function edge-length;
19: Mesh *mesh;
20:
21: Mesh *Delaunay(VertexList *points);
22: double Maximum-Movement(VertexList *oldpoints, VertexList *points);
23: VertexList *UDistribution(bbox *box, double length);
24: VertexList *RDistribution(bbox *box, int npoints);
25: void RejectPoints(length-function edge-length, VertexList *points);
26: void RemoveOuterPoints(distance-function distance, VertexList *points);
27: void RemoveOuterTriangles(Mesh *delaunay-mesh, distance-function distance);
28: void ApplyForces(VertexList *points);
29: void BackupPoints(VertexList *points, VertexList *oldpoints);
30: void BringPointsBack(VertexList *points);
31: void ExtractBars(Mesh *delaunay-mesh);
32: void CalculateForces(Mesh *delaunay-mesh);
33: void AddFixedPoints();
34: void AddOrderedBars(IMap *bar-map, int id1, int id2);
35: double MinimumQuality(Mesh *delaunay-mesh);
36:
37: public:
38: ForceEquilibriumDelaunay(distance-function fd, length-function fh,
39: double length, bbox *box, VertexList *pfix,
40: int npfix, double delta-t);
41: ~ForceEquilibriumDelaunay();
42: Mesh *distmesh2d(double minimum-quality, unsigned int points, unsigned int dist-type);
43: }FEDelaunay;
```

- **Construtores/Destrutores**: Encarregam-se, respectivamente, de alocar a memória necessária para os atributos da classe quando esta é instanciada e de desalocar a memória utilizada quando um objeto é liberado.

- **dptol**: Deslocamento máximo entre duas iterações. Quando o deslocamento for menor do que dptol, considera-se que o sistema entrou em equilíbrio.

- **ttol**: Deslocamento mínimo entre duas iterações. Quando o deslocamento relativo dos pontos entre duas iterações for maior do que ttol, a triangulação de Delaunay é construída novamente com base nas novas posições dos pontos

- **Fscale**: Defina a pressão interna do sistema. Na prática este valor aumenta o valor das forças internas em uma certa porcentagem para garantir que os pontos entrem em equilíbrio de maneira mais eficiente.

- **geps**: Tolerância de erro nos cálculos geométricos. Esta variável é escalável de acordo com o valor de h0, possibilitando a geração de malhas em domínios muito pequenos ou muito grandes.

- **deps**: Raiz quadrada da tolerância de erro do hardware. Esta variável é escalável de acordo com o valor de h0, possibilitando a geração de malhas em domínios muito pequenos ou muito grandes.

- **h0**: Comprimento inicial das arestas da malha.

- **deltat**: Intervalo de tempo utilizado na solução da equação de Euler.

- **limits**: É um quadrilátero que contém o domínio a ser triangulado, dentro do qual os pontos serão gerados.

- **nvertex**: Número de pontos utilizados na triangulação do domínio, no caso de uma distribuição aleatória de pontos.

- **bars**: Lista de todas as arestas da triangulação, sem repetição. Ou seja, a aresta $\overline{AB}$ de um triângulo t_1 é considerada igual à aresta $\overline{BA}$. São as arestas efetivamente utilizadas para o cálculo e alteração das forças internas do sistema.

- **barlist**: Lista de todas as arestas da triangulação, com repetição. Ou seja, a aresta $\overline{AB}$ de um triângulo t_1 é considerada diferente da aresta $\overline{BA}$.

- **new-points**: Pontos que fazem da parte da iteração atual do método.

- **old-points**: Pontos da iteração anterior, utilizados para cálculo de deslocamento em relação aos pontos da iteração atual.

- **fixed-points**: Pontos fixos. Pontos considerados fixos não sofrem influência das forças resultantes das arestas.

- **distance**: Callback para função distância.

- **edge-length**: Callback para função razão de comprimento.

- **Delaunay(VertexList *points)**: Constrói uma triangulação de Delaunay a partir de uma lista de pontos fornecida como parâmetro.

- **Maximum-Movement(VertexList *oldpoints, VertexList *points)**: Calcula o deslocamento máximo entre os pontos da iteração corrente e da iteração anterior.

- **UDistribution(bbox *box, double length)**: Faz uma distribuição uniforme de pontos em um quadrilátero box com base em um determinado comprimento $length$.

- **RDistribution(bbox *box, int npoints)**: Faz uma distribuição aleatória de pontos em um quadrilátero box. O número de pontos é fornecido pelo parâmetro $npoints$.

- **RejectPoints(length-function edge-length, VertexList *points)**: Descarta alguns pontos com base na probabilidade $\frac{1}{(h(x,y))^2}$.

- **RemoveOuterPoints(distance-function distance, VertexList *points)**: Descarta os pontos que se encontram fora do domínio definido pela função distância $distance$.

- **RemoveOuterTriangles(Mesh *delaunay-mesh, distance-function distance)**: Remove os triângulos que possuem baricentro fora do domínio definido pela função distância $distance$.

- **ApplyForces(VertexList *points)**: Aplica as forças armazenadas nas arestas aos pontos e calcula seus respectivos deslocamentos.

- **BackupPoints(VertexList *points, VertexList *oldpoints)**: Salva os pontos da iteração atual para serem utilizados na próxima iteração como base para o cálculo de deslocamento.

- **BringPointsBack(VertexList *points)**: Reposiciona os pontos que saíram do domínio válido definido pela função distância através de gradientes. Os pontos são movidos até a fronteira mais próxima do domínio.

- **ExtractBars(Mesh *delaunay-mesh)**: Extrai todas as arestas existentes na malha passada como parâmetro.
- **CalculateForces(Mesh *delaunay-mesh)**: Calcula as forças resultantes em cada aresta da malha com base no comprimento atual e no comprimento pretendido.
- **AddFixedPoints()**: Adiciona os pontos fixos à lista de pontos principal.
- **AddOrderedBars(IMap *bar-map, int id1, int id2)**: Constrói uma lista de arestas sem repetição com base em uma lista de arestas da malha. Arestas com vértices $id1$ e $id2$ são consideradas iguais às arestas com vértices $id2$ e $id1$ respectivamente.
- **MinimumQuality(Mesh *delaunay-mesh)**: Calcula o menor índice de qualidade dentre todos os triângulos contidos na malha.
- **distmesh2d(double minimum-quality, unsigned int points, unsigned int dist-type)**: Gera uma malha Delaunay através do método de equilíbrio de forças. Recebe como entrada a qualidade mínima para finalizar o algoritmo, o número de pontos caso seja uma distribuição aleatória e o tipo de distribuição.

Exemplos de funcionamento

A seguir vemos alguns exemplos de malhas produzidas pela implementação descrita anteriormente. O primeiro exemplo considera um domínio composto pelas seguintes primitivas:

- C_1: Círculo de raio 0.4 centrado na origem.
- C_2: Círculo de raio 0.21 centrado na origem.
- Q_1: Quadrilátero de coordenadas $p_1 = (-0.8, -0.2)$, $p_2 = (-0.8, 0.2)$, $p_3 = (-0.2, -0.2)$ e $p_4 = (-0.2, 0.2)$.

Definimos a função distância como $fd = C_1 - C_2 - Q_1$, e consideramos uma função razão de comprimento tal que os elementos sejam menores nas extremidades de C_1. O domínio é representado na figura 4.3.

Os parâmetro utilizado para a geração das malhas foi uma distribuição aleatória de pontos dentro do domínio com 1500 pontos no total dentro de um bounding box com coordenadas $p_1 = (-1, -1)$, $p_2 = (-1, 1)$, $p_3 = (-1, -1)$ e $p_4 = (-1, 1)$. Optou-se por 250 iterações, sem limite de qualidade, comprimento mínimo de 0.01 e um intervalo de tempo para a solução da equação diferencial de 0.2. O resultado pode ser visto na figura 4.4.

O segundo exemplo ilustra uma malha gerada em um domínio composto pelas 5 primitivas seguintes:

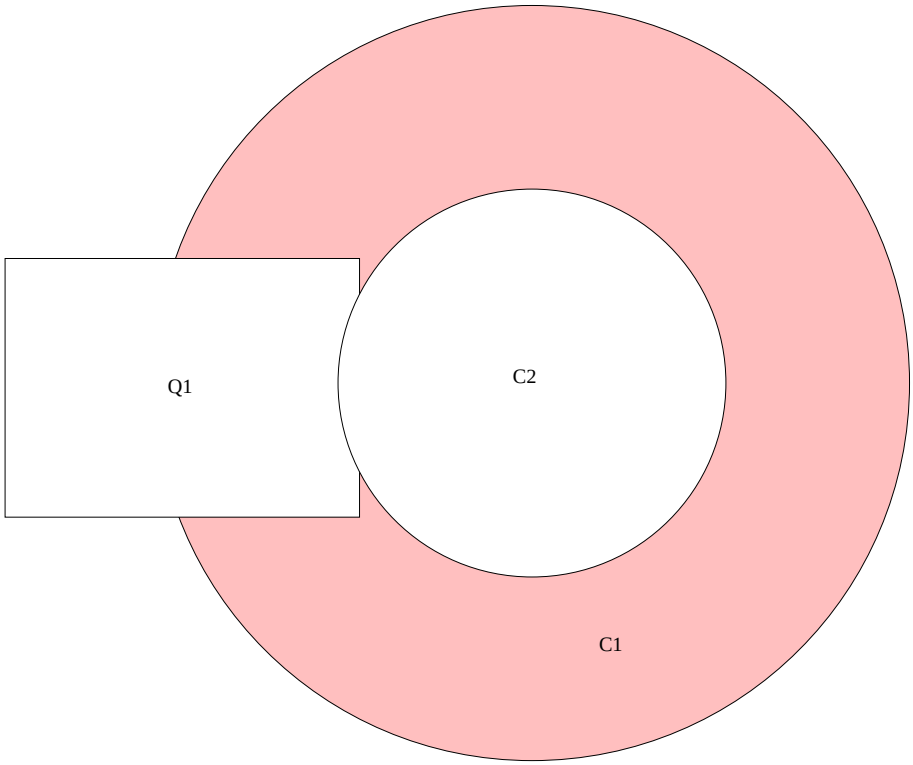


Fig. 4.3: Domínio para a geração da malha do exemplo 1

- C_1: Círculo de raio 0.4 centrado na origem.
- C_2: Círculo de raio 0.4 centrado em $c = (-0.6, 0.0)$.
- C_3: Círculo de raio 0.4 centrado em $c = (0.6, 0.0)$.
- C_4: Círculo de raio 0.4 centrado em $c = (0.0, 0.6)$.
- C_5: Círculo de raio 0.4 centrado em $c = (0.0, -0.6)$.

Neste exemplo a função distância é definida por $fd = C_1 - C_2 - C_3 - C_4 - C_5$, considerando uma função razão de comprimento que estabelece uma concentração maior de pontos próximo ao centro de C_1. O domínio é representado na figura 4.5.

Optou-se por uma distribuição aleatória de 1000 pontos dentro de um domínio cercado por um bounding box com coordenadas $p_1 = (-1, -1)$, $p_2 = (-1, 1)$, $p_3 = (-1, -1)$ e $p_4 = (-1, 1)$. O número de iterações foi descartado, fazendo com que a parada dependesse exclusivamente do parâmetro de qualidade, ajustado para 0.7. O comprimento mínimo é de 0.01 e o intervalo de tempo foi considerado como 0.2, o padrão. O resultado pode ser visto na figura 4.6.

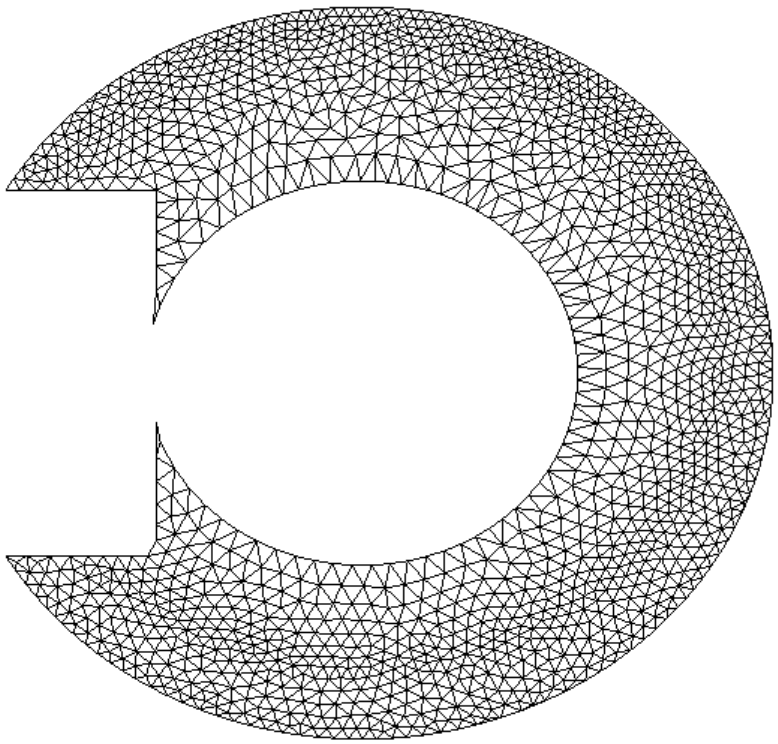


Fig. 4.4: Exemplo de malha gerada pelo método do Equilíbrio de Forças

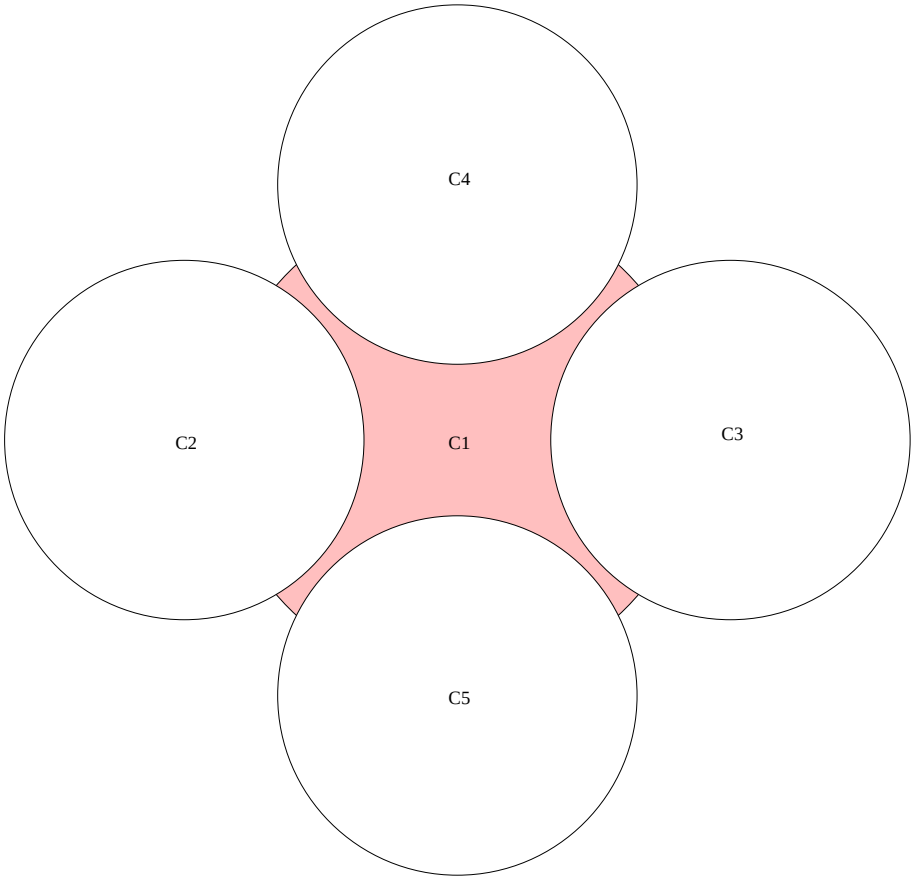


Fig. 4.5: Domínio para a geração da malha do exemplo 2

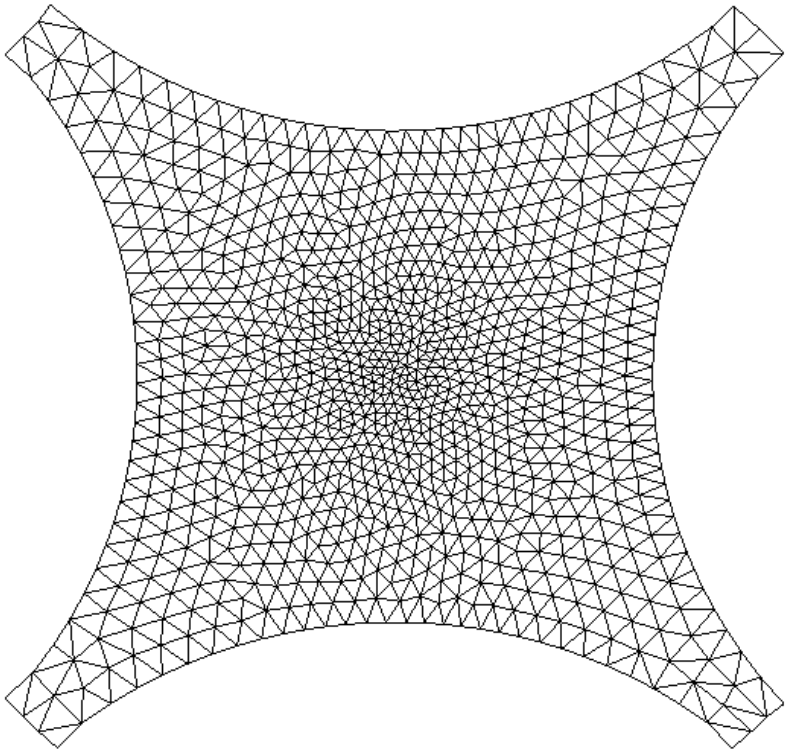


Fig. 4.6: Exemplo de malha gerada pelo método do Equilíbrio de Forças

Capítulo 5

Resultados

Com base nos métodos apresentados anteriormente - Chew, Ruppert e Equilíbrio de forças - é possível estabelecer comparações qualitativas entre eles, permitindo ressaltar as vantagens e desvantagens de um método em relação aos demais. Apresentamos ainda uma comparação quantitativa entre o método de Ruppert e o método do Equilíbrio de forças.

O método de Chew provê uma malha refinada com maior uniformidade, ou seja, contém elementos de tamanhos semelhantes e uma densidade praticamente constante. No entanto, o número de elementos pode tornar-se muito elevado dependendo do grau de refinamento escolhido.

Ao mesmo tempo em que este maior número de elementos pode trazer bons resultados em termos de qualidade de simulação, temos em contrapartida a queda do desempenho, já que os cálculos terão de considerar todos os elementos contidos no domínio. Esta queda de desempenho pode não ser aceitável em uma aplicação que tem interesse em partes específicas de um domínio, como, por exemplo, os bordos. Neste caso não há necessidade de uma malha com refinamento uniforme ao longo de todo o domínio.

O método de Chew apresenta problemas em relação a triângulos ruins próximos aos bordos, devido à restrição do tamanho da aresta, porém há um método para contornar este problema.

De modo geral este método apresenta critérios e garantias de convergência bem definidos, o que o torna bastante confiável e previsível, garantindo, teoricamente, que os ângulos da triangulação sempre estarão entre 30^o e 120^o.

O método de Ruppert, por sua vez, provê uma alternativa para o problema relacionado a aplicações que requerem refinamento em partes específicas do domínio. Esta alternativa tem como base a criação de uma malha menos uniforme e com densidade variável. Na prática, o que se nota é o aumento do tamanho dos elementos à medida que nos afastamos dos bordos do domínio. Nas fronteiras dos polígonos os elementos são bem menores, acompanhando as irregularidades típicas.

Este tipo de comportamento pode oferecer vantagens em relação a simulações numéricas que

estão focadas no comportamento próximo às fronteiras. No entanto, as porções internas do domínio ou porções que não estão próximas ao bordo terão um refinamento com elementos maiores, reduzindo a precisão nestas regiões.

Assim como o método de Chew, o método de Ruppert também apresenta boas garantias de convergência, porém também é comprometido pela presença de triângulos ruins com ângulos pequenos, caso que deve ser tratado separadamente para evitar um número ainda maior de triângulos ruins na malha.

Para o método de Ruppert e Chew, há também outras abordagens para a solução do problema dos ângulos pequenos.

O terceiro método é baseado em uma analogia com um sistema físico em equilíbrio de forças. O método apresenta resultados favoráveis em termos de qualidade, atingindo resultados equivalentes a outros métodos de refinamento Delaunay. A densidade de elementos é variável e determinada pela função $h(x, y)$, sendo que a malha pode apresentar um aspecto tão fino e uniforme quanto uma malha gerada pelo método de Chew, ou então refinada em porções específicas do domínio como no método de Ruppert.

É interessante ressaltar que através da função $h(x, y)$, é possível criar áreas de refinamento que não os bordos, possibilitando, por exemplo, um refinamento maior em regiões centrais do domínio. Outra vantagem do método é a simplicidade na definição de um domínio através de uma função distância, sendo que estas podem ser combinadas de forma a criar domínios de maior complexidade.

A analogia com o sistema físico, porém, traz desvantagens em relação ao desempenho, que pode ser consideravelmente mais baixo do que os métodos tradicionais. Isso se deve ao fato de a triangulação ter de ser refeita cada vez que os pontos têm deslocamento acima de um certo limitante. Desta forma, todas as informações das arestas são descartadas e recalculadas, ao contrários dos métodos de Chew e Ruppert, onde temos alterações locais na malha para garantir a conformidade com o critério de Delaunay, uma tarefa bem menos custosa.

As 3 etapas mais custosas do método são, em ordem, a geração de uma triangulação Delaunay, a extração das arestas e o cálculo das forças resultantes. A primeira etapa ocupa em média 61% do tempo total do método. As duas etapas seguintes ocupam, cada uma, por volta de 12% do tempo total de execução.

Há também a possibilidade de não convergência dependendo dos valores de entrada. O risco de o algoritmo não terminar cresce proporcionalmente com a complexidade da malha e com o grau de qualidade exigido.

O tratamento de ângulos pequenos e triângulos ruins não requer uma abordagem explícita, visto que a própria interação entre as forças dentro do sistema se encarrega de equilibrar a posição dos pontos, mantendo bons índices de qualidade mesmo em domínios com cantos acentuados.

5.1 Comparativo de qualidade

A análise qualitativa nos permite inferir o comportamento dos vários métodos de uma maneira mais geral, porém é de grande valia uma análise quantitativa entre os métodos, permitindo ver com maior precisão os pontos positivos e negativos dos métodos.

Promoveu-se comparativos entre o método de Ruppert e o método de Equilíbrio de forças com base em duas malhas. A primeira malha foi criada tendo-se em mente a dificuldade do método de Ruppert em lidar com ângulos pequenos de entrada no PSLG. O domínio da figura 5.1 contém um ângulo acentuado em uma de suas extremidades.

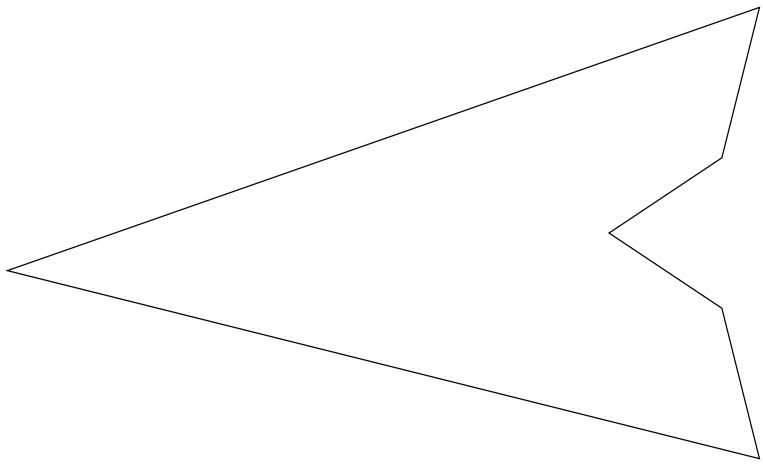


Fig. 5.1: Domínio para teste com ângulo acentuado

Ambos os métodos foram capazes de gerar as malhas corretamente, porém o método de Ruppert o fez em cerca de 2 segundos para 1507 pontos. O método do Equilíbrio de forças, devido à necessidade de geração de uma nova malha Delaunay cada vez que os pontos se deslocam consideravelmente, demorou alguns minutos, onde mais de 50% do tempo foi consumido pela reconstrução da malha Delaunay, realizando 3646 iterações no total, com 1461 pontos.

As malhas geradas por Ruppert e pelo Equilíbrio de forças podem ser vistas respectivamente nas figuras 5.2 e 5.3.

Uma análise dos ângulos dos triângulos nos permite montar um histograma com as distribuições para cada malha, mostrados respectivamente nas figuras 5.4 e 5.5.

Notamos, neste primeiro exemplo, que a malha gerada por Ruppert contém ângulos que variam de 25.259^o até 119.88^o. Observamos também que a maior concentração no histograma é de ângulos entre 50^o e 60^o, seguidos por concentrações pouco menores de ângulos entre $40^o - 50^o$ e $60^o - 70^o$. Estes dados demonstram que a malha apresenta uma boa qualidade em termos dos ângulos. Quanto maior a concentração de ângulos próximos a 60^o, melhor a forma dos trîângulos.

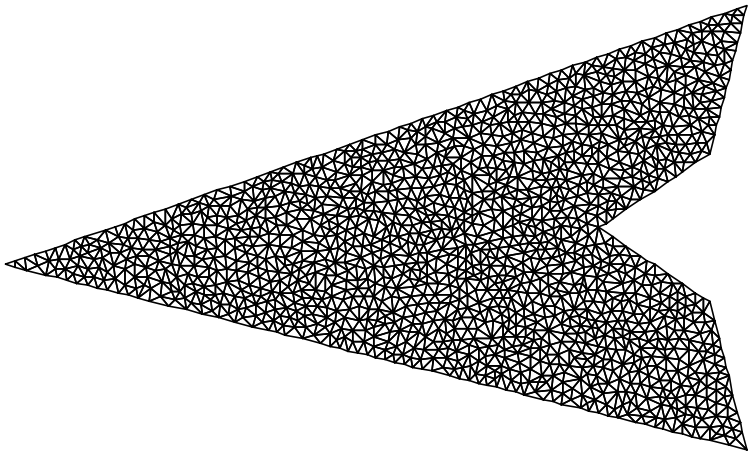


Fig. 5.2: Malha gerada por Ruppert

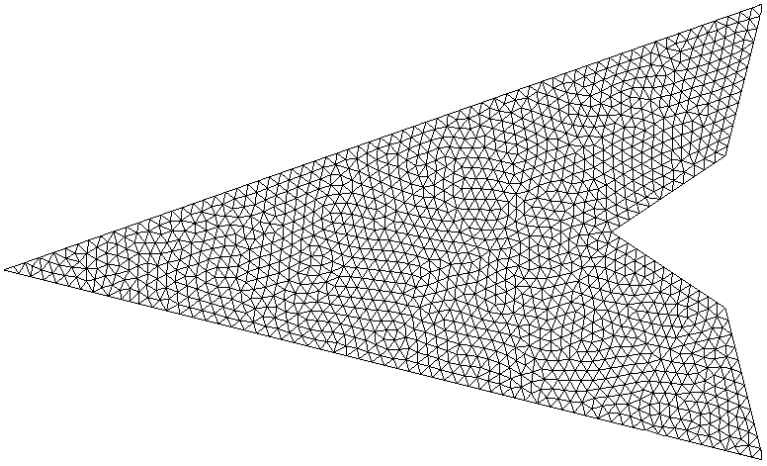


Fig. 5.3: Malha gerada pelo equilíbrio de forças

Ressaltamos que o método de Ruppert foi executado com um parâmetro adicional garantindo a qualidade da malha com um ângulo mínimo de 30^o.

A malha gerada pelo método do Equilíbrio de forças, porém, possui ângulo mínimo de 32.25^o e ângulo máximo de 105.3^o, estando estes valores compreendidos em um intervalo menor do que os ângulos na malha gerada por Ruppert. Isto demonstra uma melhora na qualidade geral da malha. Percebemos também que a concentração de ângulos no intervalo $50^o - 60^o$ é consideravelmente maior do que a equivalente no método de Ruppert. Temos respectivamente 2092 ângulos para Ruppert e 3696 ângulos para o método de Equilíbrio, compreendendo 24.5% e 44.0% do total de ângulos na malha.

Para a malha gerada pelo método de Equilíbrio de forças, apresentamos dois gráficos que destacam a distribuição dos triângulos ao longo do domínio com base em seus respectivos ângulos mínimos

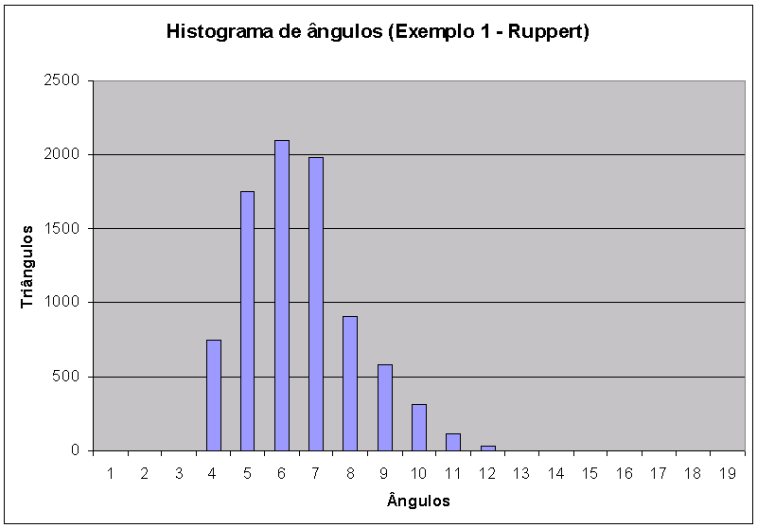


Fig. 5.4: Histograma dos ângulos para a malha gerada pelo método de Ruppert

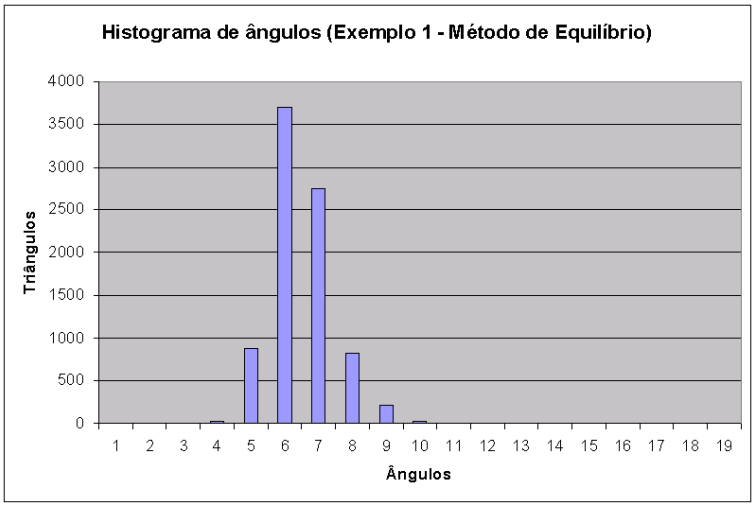


Fig. 5.5: Histograma dos ângulos para a malha gerada pelo método de Equilíbrio de forças

e máximos (figuras 5.6 e 5.7). Os triângulos de menor qualidade se encontram rentes aos bordos, com destaque para o triângulo localizado próximo ao ângulo agudo do domínio.

Concluímos, portanto, que neste exemplo o método do Equilíbrio de forças, apesar de mais lento, gerou resultados de melhor qualidade sem a necessidade de tratamento especial para os ângulos acentuados do domínio como o método de Ruppert. É importante notar que as malhas têm praticamente o mesmo número de elementos. Temos 2838 triângulos na malha de Ruppert e 2794 triângulos para o método de Equilíbrio.

É possível fixar um número de iterações para que o método de Equilíbrio finalize, resultando em uma malha levemente pior em termos de qualidade, porém ainda aceitável para a realização de

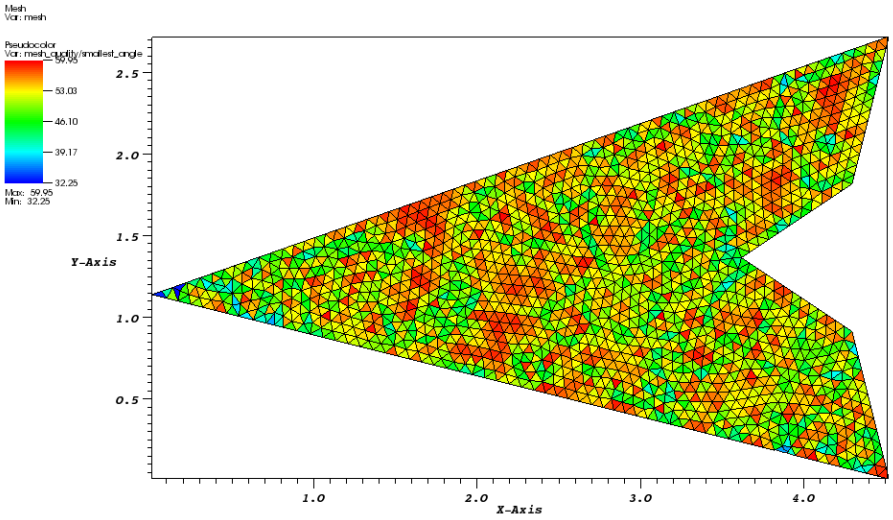


Fig. 5.6: Distribuição com destaque para os ângulos mínimos

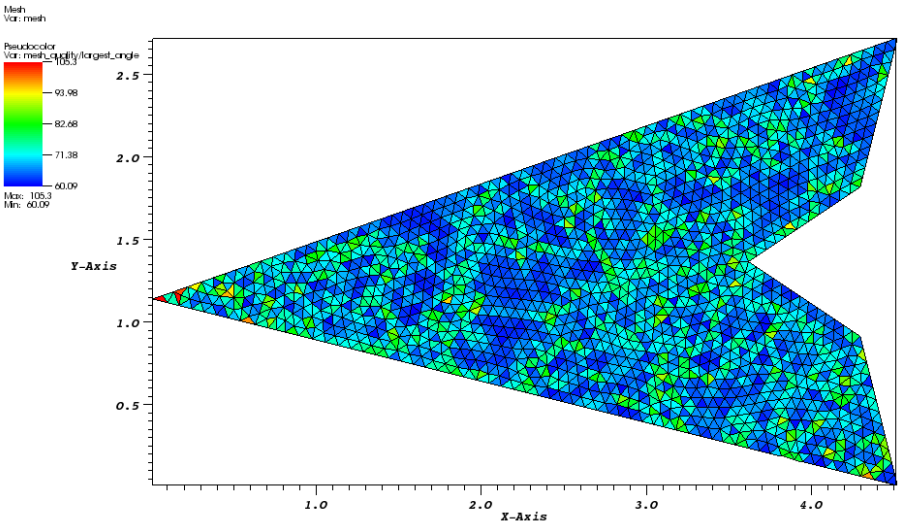


Fig. 5.7: Distribuição com destaque para os ângulos máximos

simulações.

No segundo exemplo temos um domínio que define 4 cantos acentuados (figure 5.8). As malhas geradas respectivamente pelo método de Ruppert e Equilíbrio de forças podem ser vistas nas figuras 5.9 e 5.10.

O tempo de execução se manteve para o método de Ruppert, concluindo a geração em pouco menos de 2 segundos para 1047 pontos. O método de Equilíbrio de forças também manteve o padrão com um tempo longo de execução, em torno de 75 segundos, com a reconstrução das malhas Delaunay consumindo 51% do tempo de execução do método, num total de 601 iterações para 1462 pontos.

A malha gerada por Ruppert tem ângulo mínimo de 22.834^o e ângulo máximo 116.89^o. Neste

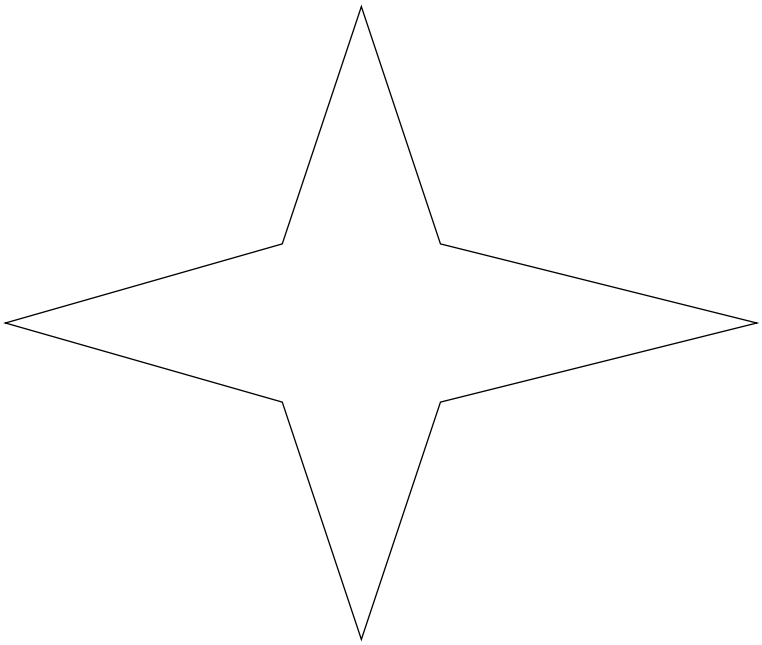


Fig. 5.8: Domínio para o segundo exemplo

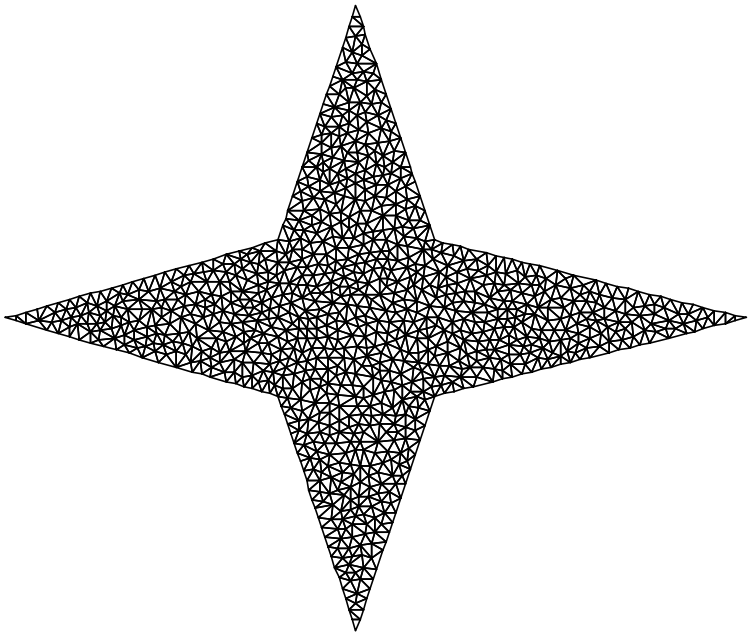


Fig. 5.9: Malha gerada por Ruppert

segundo exemplo também utilizamos um parâmetro garantindo a qualidade da malha com um ângulo mínimo de 30^o.

A malha do método de Equilíbrio de forças possui ângulos mínimo e máximo respectivamente de 28.07^o e 112.2^o. Observamos novamente que os valores estão em um intervalo menor do que os

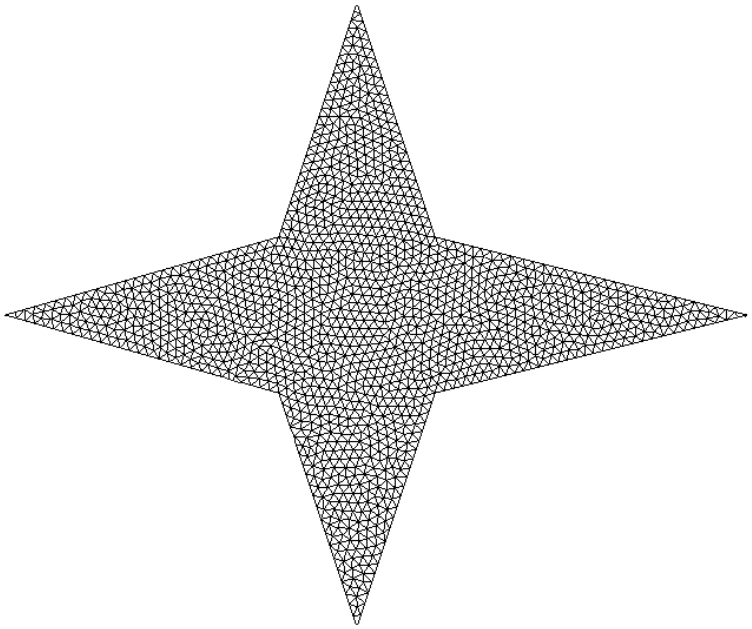


Fig. 5.10: Malha gerada pelo equilíbrio de forças

ângulos na malha gerada de Ruppert.

Analisando os histogramas das figuras 5.11 e 5.12 e os gráficos 5.13 e 5.14, temos a maior concentração no histograma por ângulos entre 50^o e 60^o, seguidos por concentrações menores de ângulos entre $40^o - 50^o$ e $60^o - 70^o$, demonstrando a boa qualidade da malha gerada. Estes ângulos compreendem 26.3% de todos os ângulos no método do Equilíbrio, enquanto no método de Ruppert eles compreendem 23.7%. Por outro lado, nesta iteração o método de Equilíbrio de forças teve um número maior de triângulos com ângulos entre 20^o e 30^o em comparação com o método de Ruppert, mas, como vimos acima, a malha gerada pelo Equilíbrio possui, em relação a Ruppert, ângulo mínimo maior e ângulo máximo menor.

A partir deste exemplo, concluímos que a malha gerada pelo método de Equilíbrio de forças é levemente superior em termos de qualidade com relação à malha gerada pelo método de Ruppert. Isto se deve ao menor intervalo entre o menor e o maior ângulo, o que reflete na forma dos triângulos da triangulação. O tempo de execução, porém, ainda é consideravelmente alto em comparação com Ruppert, o que requer trabalho adicional para o aumento de desempenho.

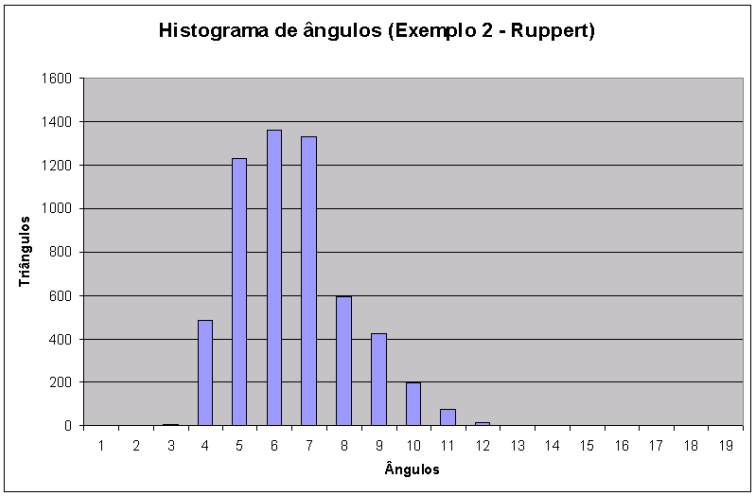


Fig. 5.11: Histograma dos ângulos para a malha gerada pelo método de Ruppert

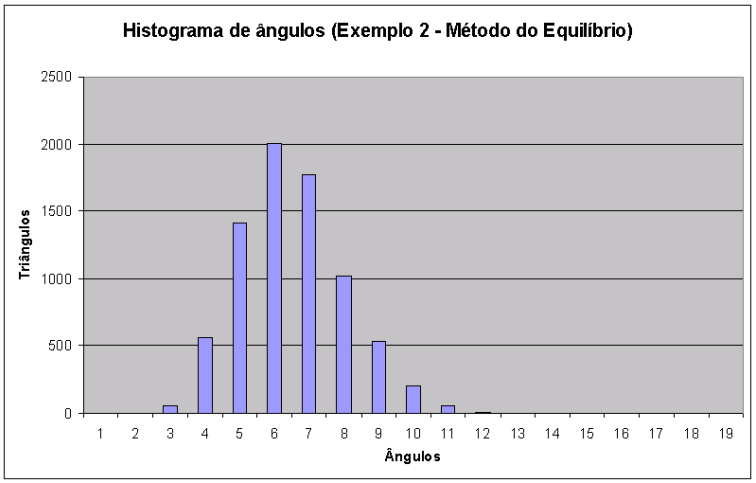


Fig. 5.12: Histograma dos ângulos para a malha gerada pelo método de Equilíbrio de forças

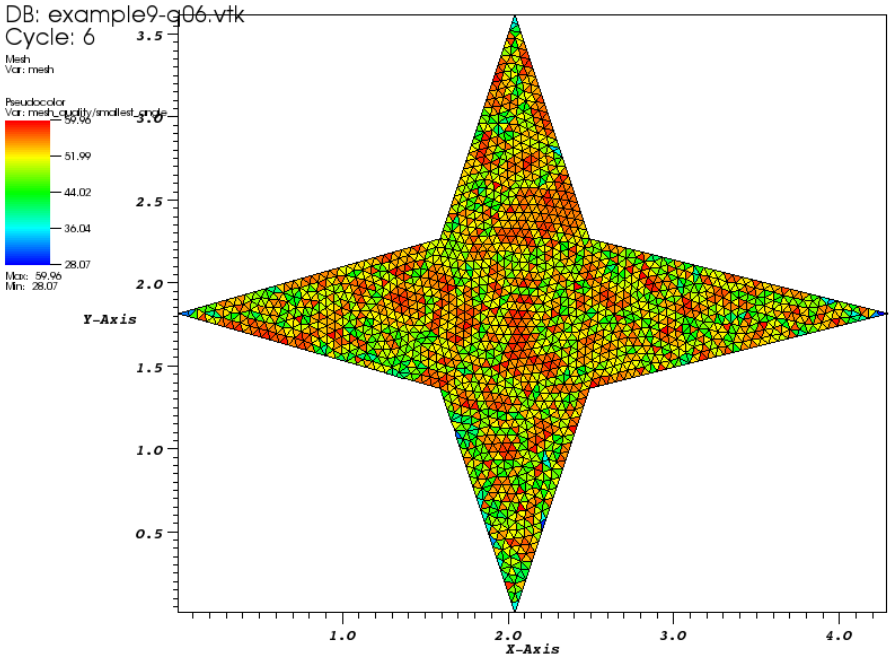


Fig. 5.13: Distribuição com destaque para os ângulos mínimos

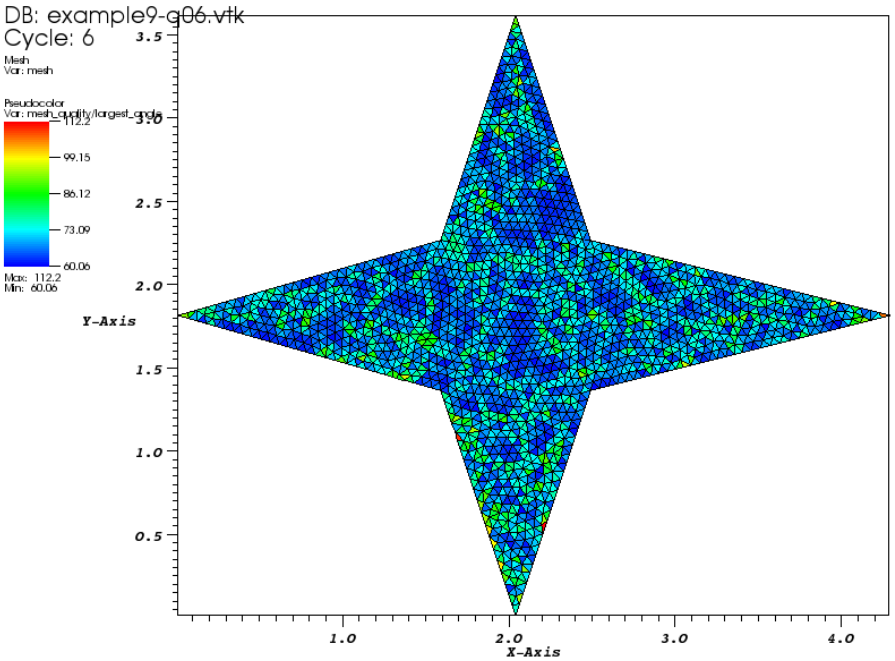


Fig. 5.14: Distribuição com destaque para os ângulos máximos

Capítulo 6

Conclusões

Este trabalho apresentou uma introdução ao contexto das simulações numéricas, área que vem se desenvolvendo com a utilização de novos métodos para obtenção de resultados de melhor qualidade. As simulações numéricas têm como base as malhas de triângulos, uma decomposição do domínio de simulação em elementos menores que permite a solução de problemas e observação do comportamento de um determinado modelo físico ao longo do tempo.

Contextualizamos também o projeto GESAR, que objetiva a construção de um sistema para simulação de enchimento de reservatório e de análise de impacto ambiental devido ao acúmulo de matéria orgânica nas águas. O projeto GESAR pode se beneficiar diretamente deste trabalho.

Grande parte das simulações utiliza malhas Delaunay para a descrição dos domínios. Destacamos três métodos de geração de malhas Delaunay que podem ser utilizados para tal fim: Método de Chew, Método de Ruppert e Método do Equilíbrio de Forças.

O método de Chew é uma abordagem robusta para geração de malhas Delaunay refinadas, preenchendo o domínio com uma grande quantidade de triângulos menores e que possuem quase o mesmo tamanho, resultando em uma distribuição uniforme, ideal para modelos de simulação que requerem alto nível de detalhamento ao longo de todo o domínio. O método de Chew também se mostra bastante robusto em termos de critério de parada e possui garantia de qualidade teórica.

O método de Ruppert traz um pouco mais de adaptatividade aos métodos de geração de malha, sendo capaz de variar a densidade de triângulos ao longo do domínio, tornando-o uma boa opção para modelos de simulação que requerem maior detalhamento nos bordos e menor detalhamento nas porções centrais do domínio.

O método do Equilíbrio de forças é uma abordagem inovadora que utiliza como base uma analogia física com um sistema de molas em equilíbrio, onde as molas são as arestas e os elos de ligação são os vértices. Apesar de ter um menor desempenho em relação ao método de Chew e Ruppert, o método se mostrou capaz de gerar malhas de ótima qualidade, superando algumas vezes os métodos

tradicionais. A definição do domínio através de uma função distância implícita e a definição de uma regra de refinamento, através da função razão de tamanho dos elementos, também traz mais flexibilidade ao método, permitindo áreas de refinamento localizadas que não seriam possíveis sem a inserção explícita de pontos adicionais nos métodos de Chew e Ruppert.

As implementações do projeto tiveram como base a estrutura experimental OF, desenvolvida pelo LCAD.

Para trabalhos futuros, pretende-se melhorar o método de Equilíbrio de Forças e promover uma maior integração com o Projeto GESAR. Como exemplo de melhoria podemos citar o suporte a restrições, principalmente restrições definidas por curvas de nível provindas de uma descrição de terreno onde planeja-se construir um reservatório. Com base nestes dados, uma malha poderia ser construída visando a simulação de enchimento de um futuro reservatório na região.

Referências Bibliográficas

[1] F. Aurenhammer and R. Klein. *Handbook of Computational Geometry*. Elsevier Sciencee, 2000.

[2] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram. *Pattern Recognition*, 17:251–257, 1984.

[3] B.G. Baumgart. A polyhedron representation for computer vision. In *AFIPS Naional Computer Conference*, volume 44, pages 589–596, 1975.

[4] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In Hwang F.K. and Du D.Z., editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 1992.

[5] M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. In *31st Annual Symposium on Foundations of Computer Science*, pages 231–241. IEEE Computer Society Press, 1990.

[6] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec. Triangulations in cgal. In *16th Annu. ACM Sympos. Comput. Geom.* ACM, 2000.

[7] S. Cheng and T Dey. Quality meshing with weighted delaunay refinement. In *Proceedings of the 13th ACM-SIAM Sympos. Discrete Alg.*, pages 137–146, 2002.

[8] L.P. Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Department of Computer Science - Cornell University, 1989.

[9] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 274–280, 1993.

[10] H. Edelsbrunner. Triangulations and meshes in computational geometry. In *Acta Numerica*, pages 1–81. Cambridge University Press, 2000.

[11] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge, 2001.

[12] S. Fortune. Voronoi diagrams and delaunay triangulation. In Hwang F.K. and Du D.Z., editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 193–233. World Scientific, Singapore, 1992.

[13] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Trans. On Graphics*, 4(2):74–123, 1985.

[14] S.H. Lo. A new mesh generation scheme for arbitary planar domains. *Int. J. for Numerical Methods in Engineering*, 21:1403–1426, 1985.

[15] R. Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[16] M. Mäntylä. *An introduction to solid modeling*. Computer Science Press, 1988.

[17] D.L. Marcum and N.P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal*, 33(8):1619–1625, 1995.

[18] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the 12th International Meshing Roundtable*, pages 91–102. Sandia National Laboratory, September 2003.

[19] G.L. Miller, D. Talmor, S.-H. Teng, and Walkington. A delaunay based numerical method for three dimensions: generation, formulation, and partition. In *27th Annu. ACM Sympos. Theory Comput.*, pages 683–692, 1995.

[20] L.G. Nonato, A. Castelo, M.A.S. Lizier, and M.C.F. Oliveira. Topological approach for detecting objects from images. In *Vision Geometry XII - Proceedings Eletronic Imaging*, volume 5300, pages 62–73. SPIE, 2004.

[21] L.G. Nonato, A. Castelo, and M.C.F. Oliveira. A topological approach for handling triangle insertion and removal into two-dimensional unstructured meshes. *Cadernos de Computação - ICMC/USP*, 3(2):210–221, 2002.

[22] S. J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of the 7 th International Meshing Roundtable*, pages 239–267, 1998.

[23] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46:329–345, 2004.

[24] J. Rossignac, A. Safonova, and A. Szymczak. 3d compression made simple: Edgebreaker on a corner table. In *Shape Modeling International*, 2001.

[25] J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[26] W.J. Schroeder, K. Martin, and W. Lorensen. *Visualization Toolkit, An Object-Oriented Approach to 3D Graphics - 2nd ed*. Prentice-Hall, 1998.

[27] J. R. Shewchuk. Delaunay refinement mesh generation. Phd thesis, Carnegie Mellon University, School of Computer Science, 1997.

[28] J.R. Shewchuk. Lecture notes on delaunay mesh generation. Technical Report CA 94720, Department of Electrical Engineering and Computer Science - Berkeley, 2000.

[29] V. Srinivasan, L.R. Nackman, J-M. Tang, and S.N. MeshKat. Automatic mesh generation using the symmetric axis transformation of polygonal domains. Technical Report RC 16132, Comp. Science, IBM Research Division, Yorktown Heights, NY, 1990.

[30] S-H. Teng and C.W. Wong. Unstructured mesh generation: Theory, practice, and perspectives. *Int. J. Computational Geometry & Applications*, 10(3):227–266, 2000.