Universidade Federal de Minas Gerais
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Geração de Nuvem de Pontos para Métodos sem Malhas

Lucas Pantuza Amorim

Dissertação de mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Renato Cardoso Mesquita

Belo Horizonte, setembro de 2011

Resumo

Uma técnica para gerar nuvens de pontos para métodos sem malha, limitada por uma função de densidade especificada e pela geometria de entrada, é apresentada.

Inicialmente, os pontos são distribuídos ao longo das bordas da geometria identificando seus limites e as fronteiras entre diferentes materiais. Para gerar pontos dentro da geometria, duas abordagens diferentes são usadas: (i) distribuição aleatória e (ii) uma subdivisão baseada em Grafos de Folhas Autônomas (ALG), onde o menor quadrado que circunda a geometria é recursivamente subdividido em quatro, com a criação de um novo ponto no centro de cada quadrado. Em ambos os casos, o processo iterativo é interrompido quando a densidade de pontos se aproxima da função de densidade especificada.

Os pontos iniciais são redistribuídos usando o algoritmo iterativo de refinamento de *Lloyd*, até a distribuição esperada ser cumprida. Tanto distribuições uniformes de pontos quanto não uniformes, onde a função de densidade não é constante, podem ser atendidas.

A qualidade final da nuvem resultante só depende do algoritmo de refinamento de *Lloyd*. No entanto, o algoritmo que gera pontos iniciais baseando-se no ALG, embora mais complexo do que a geração aleatória, necessita menos iterações do algoritmo de *Lloyd* para atingir a qualidade esperada. Isto é devido ao fato de que seus pontos de distribuição inicial já levam em conta a função de densidade especificada.

Abstract

A technique to generate point clouds for meshless methods which are constrained by a specified density function and by the input geometry is presented.

Initially, points are distributed along the geometry edges identifying their limits and the boundaries between different materials. To generate points inside the geometry, two different approaches are used: (i) random distribution and (ii) a subdivision based on Autonomous Leaves Graphs (ALG), where the smallest square that surrounds the geometry is recursively subdivided into four, with the creation of a new point in the center of each square. In both cases, the iterative process stops when the point's density approximates the specified density function.

The initial points are redistributed using an iterative Lloyd refinement algorithm, until the expected distribution is met. Uniform distributions of points, as well non-uniform ones, where the density function is not constant, can be met.

The final quality of the resulting cloud just depends on the Lloyd refinement algorithm. However, the algorithm that generates initial points based on ALG, although more complex than the random generation, needs less iterations of the Lloyd's algorithm to achieve the expected quality. This is due to the fact that its initial points distribution already takes into account the specified density function.

Sumário

Lista de Figuras

Lista de Tabelas

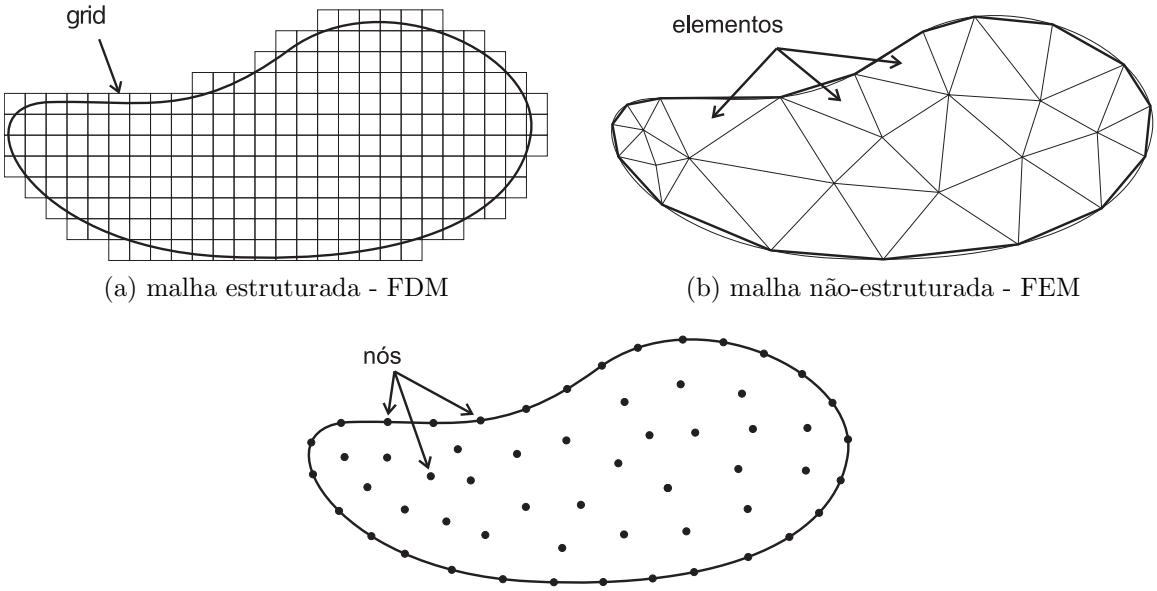
Capítulo 1

Introdução

Vários fenômenos físicos tratados pelas Engenharias e pela Física podem ser descritos por equações diferenciais parciais em conjunto com restrições adicionais, as chamadas condições de contorno ou condições de fronteira [14]. A solução para este tipo de problema, é aquela que satisfaz a equação diferencial e as condições de contorno, sendo que para as condições dadas para o problema, ela deve ser única e depender continuamente das condições de entrada.

Alguns poucos problemas homogêneos e de geometria simples podem ser resolvidos analiticamente, mas para a maioria dos problemas de interesse prático não existe solução analítica e, nestes casos, métodos computacionais devem ser utilizados. Dentre os diversos existentes, os mais tradicionais e explorados são o método dos elementos finitos (FEM) [13] e o método das diferenças finitas (FDM) [29].

Ambos possuem como característica a necessidade de uma malha que representa o domínio do problema. Para o FDM a malha é estruturada (Figura 1.1a): os intervalos entre nós adjacentes nas direções x e y (em duas dimensões) são constantes. Apesar de ser de fácil geração, ela não é capaz de se adaptar exatamente à geometria do problema, sobretudo quando existem curvas, o que gera erros de aproximação da solução numérica. Já a malha do FEM, por ser geralmente não estruturada, se ajusta ao contorno, adaptando-se melhor à geometria do problema (Figura 1.1b).



(a) malha estruturada - FDM

(b) malha não-estruturada - FEM

(c) nós de métodos sem malha

Figura 1.1: Representação do domínio em diferentes métodos numéricos. (a) Malha estruturada (*grid*) usada no FDM. (b) Malha não-estruturada usada no FEM. (c) Métodos sem malha, onde nós são distribuídos sobre o domínio e sua fronteira. Figura retirada de [10].

Por dependerem de uma malha, a precisão dos resultados numéricos destes métodos está diretamente ligada à boa formação dos elementos que a constituem. Apesar da geração da malha ser uma tarefa simples, garantir que seus elementos sigam os critérios de qualidade desejados não é tão simples, especialmente em três dimensões. Ainda, para problemas cuja geometria se altera ao longo do tempo, se faz necessária uma nova malha a cada instante analisado, visto que elementos que antes atendiam aos requisitos de qualidade eventualmente deixam de atendê-los.

Os métodos sem malha são um método numérico alternativo que não necessita de uma malha para representar o domínio, dependendo exclusivamente de uma nuvem de um conjunto de nós independentes, distribuídos ao longo da sua fronteira e área (para problemas em duas dimensões) (Figura 1.1c) ou volume (para problemas em três dimensões). Com isso, os problemas relacionados com a malha, sobretudo com a geometria dos elementos que a constituem, deixam de existir. No entanto, a distribuição dos pontos precisa ser tal que garanta a representatividade de todo o domínio, o que exige métodos criteriosos para geração.

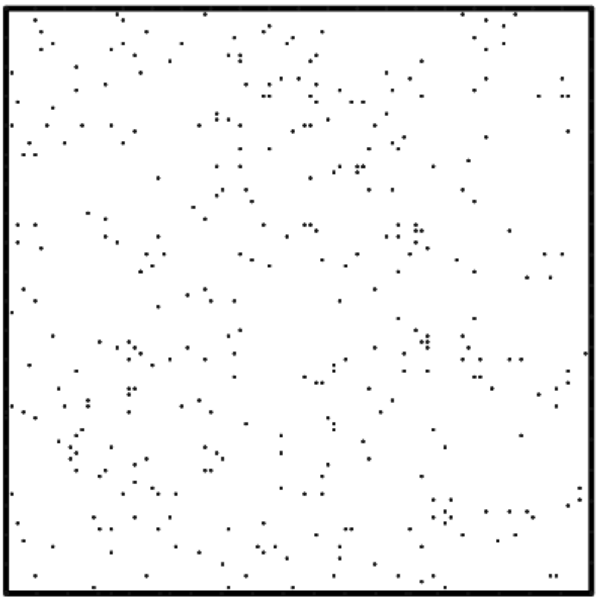
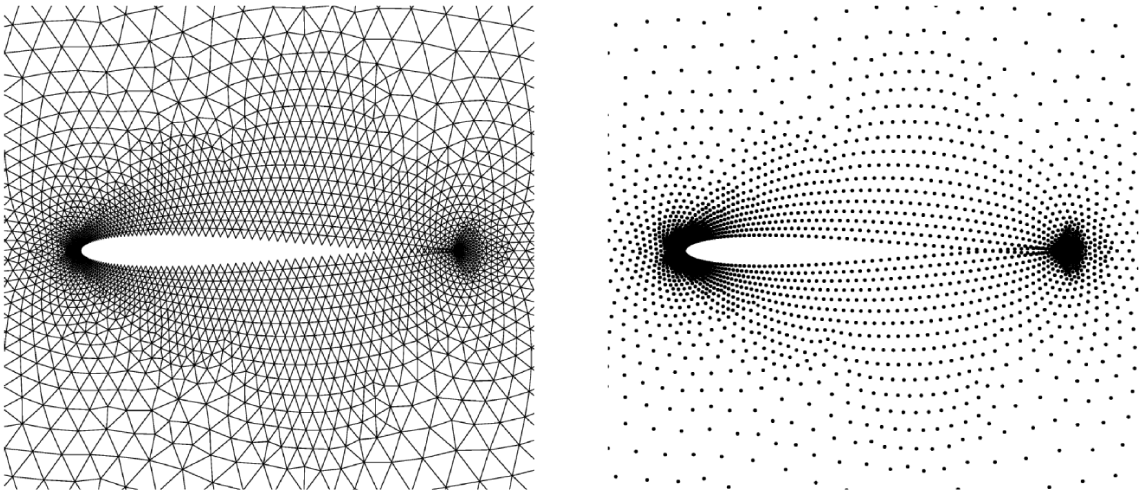


Figura 1.2: Resultado da distribuição aleatória de pontos.

A forma mais intuitiva para geração da nuvem de pontos é a distribuição aleatória de pontos cujo resultado pode ser visto na Figura 1.2. Apesar de ser um processo simples e de baixo custo computacional, gera uma distribuição ruim para o método numérico. Devido à escolha das coordenadas para cada novo ponto ser feita de forma aleatória, pontual, desconsiderando a existência dos pontos já existentes e dos que ainda serão criados, surgem espontaneamente regiões muito densas e outras pouco densas, sem refletir as características desejadas.



(a) (b)

Figura 1.3: Nuvem de pontos (b) extraída a partir de uma malha FEM (a).

Outra forma, bastante utilizada devido à grande utilização do FEM e consequentemente uma grande variedade de ferramentas existentes, é a extração da nuvem de pontos a partir de uma malha gerada para o domínio, como pode ser visto na Figura 1.3. Uma vez garantida uma malha de boa qualidade, seja qual for a escolha do elemento que a constitui, extraídos os vértices e descartada a estrutura, teremos uma nuvem de pontos também de qualidade. Para problemas em duas dimensões é uma boa solução, mas se considerarmos que a principal motivação dos trabalhos em métodos sem malha é justamente não precisar de uma malha e, consequentemente, não ser necessário tratar a complexidade do seu processo de geração, percebemos que é contrassenso investir nesta solução.

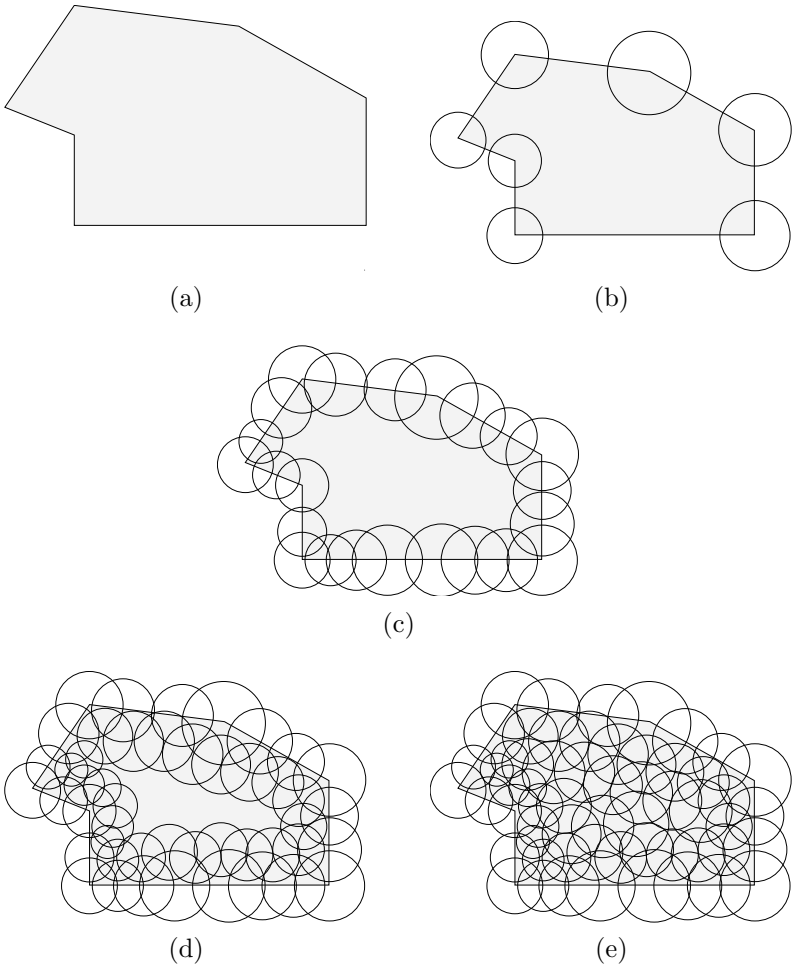


Figura 1.4: Método de geração de nuvem de pontos baseado na distribuição de círculos para a representação do domínio de influência do ponto, proposto por Li et al. [19].

Partimos então para soluções de geração criteriosa de nuvem de pontos, mas que não exijam, mesmo que indiretamente, a existência de uma malha. Uma abordagem utilizada por Li et al. [16, 17, 18] é espalhar ao longo do domínio elementos de formas geométricas bem definidas

como esferas, retângulos ou polígonos simples. Cada um deles é associado a um ponto, e sua área, à região de influência deste ponto. O tamanho do elemento, sua forma e a sobreposição de elementos remetem à precisão da simulação numérica.

É o caso, por exemplo, da solução ilustrada na Figura 1.4, proposta por Li et al. [19]. Nela, o elemento escolhido é o círculo, relacionando o raio com a função de espaçamento e o centro com a coordenada do ponto que integra a nuvem. Quanto maior o raio, maior a área de influência do ponto em questão e, consequentemente, menos densa é a distribuição nesta região. O algoritmo funciona da seguinte forma: partindo-se de um objeto como o ilustrado na Figura 1.4a, distribuem-se círculos nos vértices (Figura 1.4b), depois ao longo do comprimento das arestas (Figura 1.4c), seguida pela distribuição ao longo da margem do objeto, como em camadas, da mais externa (Figura 1.4d) até se chegar à mais interna (Figura 1.4e). Assim, toda a área é discretizada em círculos que se sobrepõem em parte de sua área, mas nunca coincidindo as coordenadas do centro, permitindo que eles sejam mapeados como pontos da nuvem.

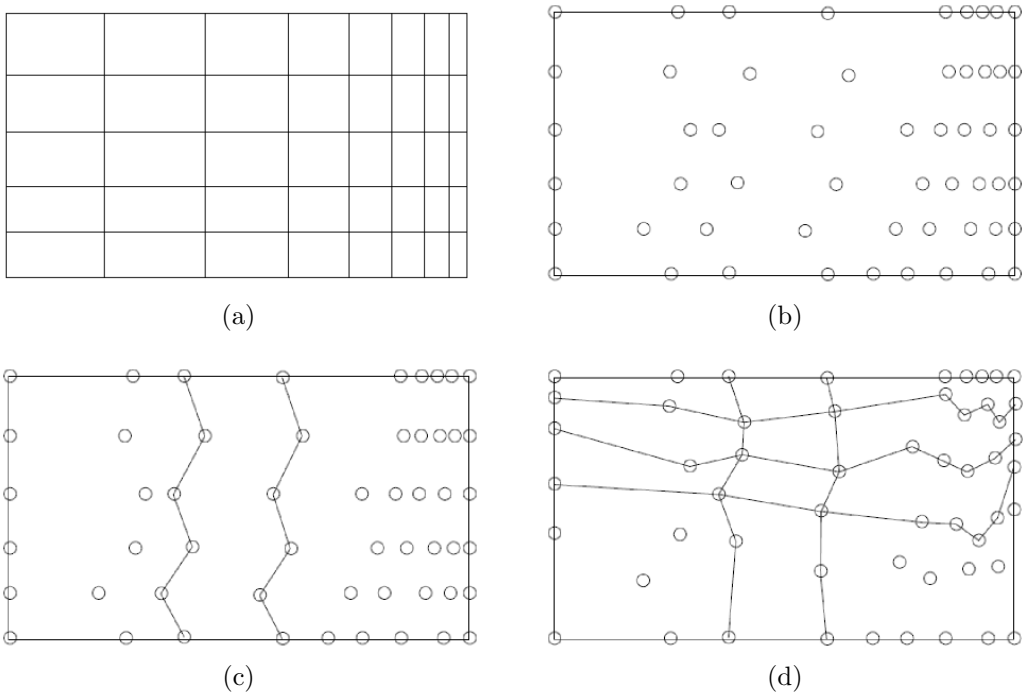
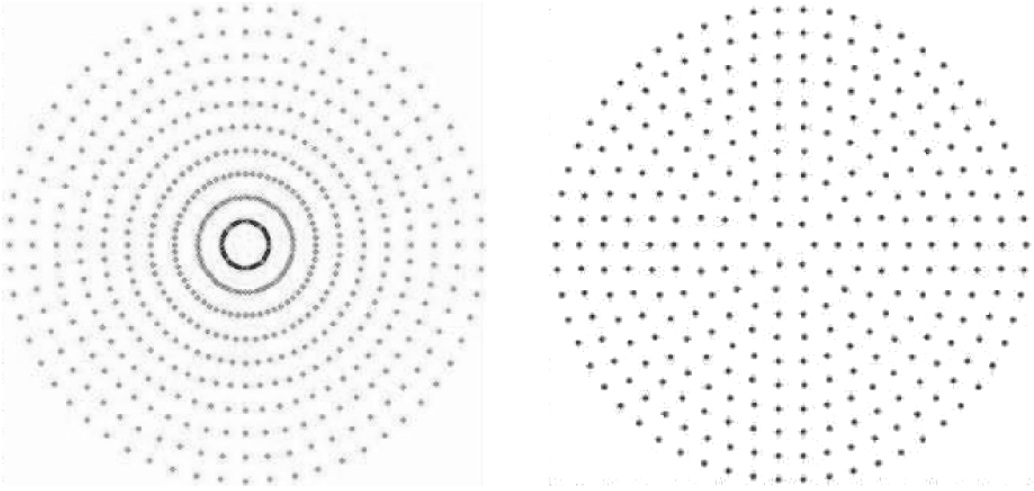


Figura 1.5: Etapas do refinamento da malha adaptativa proposto por Shanazari and Hosami [28]. Malha adaptativa em um retângulo (a) e etapas de transformação do retângulo em outra geometria (b), (c) e (d).

Outra abordagem, apresentada por Shanazari and Chen [27], é baseada na distribuição de pontos ao longo das linhas de grade nas direções de coordenadas para a produção de pontos em regiões retangulares, como ilustrado na Figura 1.5a. No entanto, devido ao uso de linhas de grade, este método é limitado ao caso de domínios retangulares e cúbicos, respectivamente, duas e três dimensões.

Shanazari and Hosami [28] estendem esta solução para casos mais gerais com limites irregulares em duas dimensões. Inicialmente, gera-se uma nuvem de pontos como uma malha adaptativa em um retângulo, como na figura 1.5a. Então, é feito o mapeamento dos nós gerados para o domínio real do objeto tratado, empregando uma transformação adequada. As Figuras 1.5b, 1.5c e 1.5d mostram etapas desta transformação e a Figura 1.6a mostra o resultado do mapeamento para um objeto circular. Após a transformação, a nuvem é refinada até que certas propriedades definidas inicialmente sejam alcançadas, sendo que nesta etapa, para se garantir a densidade desejada, pontos podem ser retirados da nuvem. O inverso, inclusão de pontos, não faria sentido, uma vez que o retângulo inicial gerado garantidamente com a densidade desejada possui área maior que o objeto no qual ele é transformado, e consequentemente, a nuvem inicial possui pontos sobressalentes. A Figura 1.6b mostra o resultado do refinamento da distribuição ilustrada em 1.6a, para problemas que necessitam de distribuição homogênea de pontos.



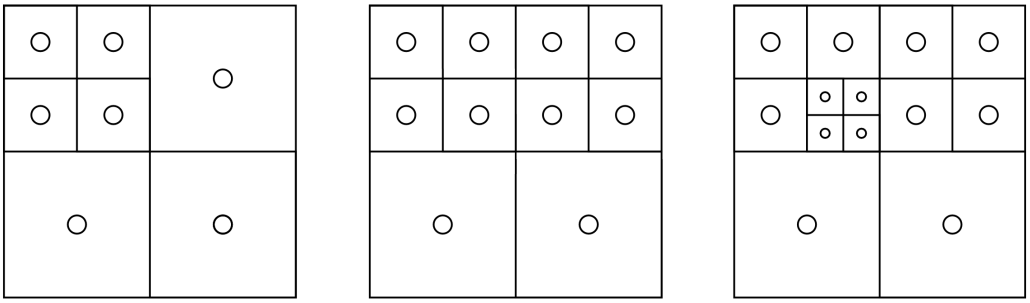
(a) (b)

Figura 1.6: Os pontos de malha transformada para um domínio em forma de círculo, antes (a) e depois (b) da etapa de refinamento.

Outra solução importante, não originalmente aplicada à geração de nuvem de pontos, mas que trata também do problema da representação de um domínio utilizando pontos é o trabalho desenvolvido por Burgarelli [3]. Descritos com detalhes na Seção 3.2.3, os algoritmos apresentados realizam o refinamento adaptativo de malhas, permitindo uma região, que em determinado instante de tempo ganhe maior interesse, ser refinada tornando-se mais densa. Assim que esta região deixa de ser foco de interesse, ela é desrefinada e a densidade anterior é restaurada.

Para possibilitar o refinamento e desrefinamento adaptativo, usa-se a divisão do espaço em regiões quadradas subdivididas em quadrantes. Quando necessário, todo um quadrante é refinado, sendo subdividido em quatro novos quadrantes, ou desrefinado, unificando quatro quadrantes em um único quadrante de nível superior.

Na Figura 1.7 podemos visualizar etapas do processo de refinamento. Partindo de uma configuração já avançada como ilustrado em 1.7a, 1.7b seria o resultado do refinamento do seu primeiro quadrante, enquanto que 1.7c é resultado do refinamento do segundo quadrante do quarto quadrante de 1.7b. De forma similar, mas em sentido inverso, o desrefinamento de 1.7c poderia gerar uma configuração como a ilustrada em 1.7b, que por sua vez, quando desrefinada, poderia gerar uma configuração como a ilustrada em 1.7a.



(a) (b) (c)

Figura 1.7: Exemplo de três passos, (a), (b) e (c), do refinamento adaptativo.

Apesar de existirem soluções como as citadas acima, a geração de nuvem de pontos para métodos sem malha de forma eficiente e de qualidade ainda é pouco explorada. Possivelmente isto se deve ao fato de a comunidade científica que trabalha com estes métodos se preocupar mais com o desenvolvimento do método numérico em si.

É neste cenário que apresentamos este trabalho, que objetiva a geração de uma nuvem de pontos distribuídos ao longo do domínio e seu contorno, segundo restrições de densidade e qualidade definidas inicialmente.

Inicialmente, identificaremos a geometria do problema a partir de esboços feitos pelo usuário e as regiões de interesse diferenciado, com respectivas configurações de densidade da distribuição a ser gerada. Depois, faremos a distribuição de uma nuvem de pontos inicial no interior do objeto sem, necessariamente, atender aos requisitos de qualidade. Faremos isto de duas formas diferentes: segundo uma distribuição aleatória e segundo uma distribuição baseada na divisão recursiva do espaço, como a solução apresentada por Burgarelli [3]. Completando a nuvem, distribuiremos pontos nos limites externos (contornos dos objetos), fronteiras (contorno compartilhado entre dois objetos), limites internos (contorno de buracos no objeto), vértices (extremidades dos segmentos que compõem os contornos, podendo ser compartilhados por vários objetos), e nos pontos e segmentos identificados como regiões de particular interesse.

Nosso objetivo ao implementar duas abordagens para um mesmo problema é poder comparar o impacto de diferentes configurações de nuvem inicial na tarefa de refinamento, tipicamente de maior custo computacional, que neste trabalho é tratada usando o algoritmo de *Lloyd* (Seção 2.4.4) iterativamente até se atingir um diagrama de *Voronoi* centroidal (Seção 2.4.3).

O presente texto está organizado da seguinte forma: o Capítulo 2 faz uma revisão geral sobre os conceitos básicos da geometria computacional que foram utilizados neste trabalho. O Capítulo 3 apresenta a solução proposta enfatizando as técnicas adotadas; os resultados de geração de nuvens com diversas configurações são mostrados e analisados no Capítulo 4, com uma das geometrias sendo escolhida para ser usada na solução de um problema prático usando o método sem malha. Por fim, no Capítulo 5 discutimos as impressões a respeito da solução, avaliamos resultados e propomos melhorias como trabalhos futuros.

Capítulo 2

Fundamentos

O problema da discretização da superfície de objetos em duas dimensões, objeto deste trabalho, é de natureza geométrica. Desde a representação dos contornos do objeto até a localização dos elementos do método numérico, sejam pontos, triângulos, quadrados, retângulos, esferas ou outro polígono, incluindo todas as operações e manipulações de dados necessárias para a solução do problema tratado, são assuntos tratados pela geometria computacional, compondo e mantendo as estruturas de dados a partir da geometria básica dos objetos.

A geometria computacional, procura desenvolver e analisar algoritmos e estruturas de dados para resolver problemas geométricos diversos. Neste contexto, a área de projeto e análise de algoritmos tem grande importância, uma vez que procura-se caracterizar a dificuldade de problemas específicos, determinando a eficiência computacional dos algoritmos e usando técnicas de análise de complexidade assintótica [15, 1].

Neste capítulo descrevemos os conceitos geométricos que foram utilizadas em nossa proposta para geração da nuvem de pontos, mesmo que indiretamente. Abordaremos conceitos básicos de estruturas geométricas até os algoritmos usados na manipulação, como algoritmos de operações em polígonos, diagrama de *Voronoi* e o algoritmo de *Lloyd*.

2.1 Elementos básicos

Na geometria computacional, cada objeto é codificado usando um ou mais pares de coordenadas, o que permite determinar sua localização e aparência visual. O mais elementar deles é o ponto (Figura 2.1a), que em duas dimensões é o par ordenado (x, y) de coordenadas espaciais. Sendo p_1 e p_2 dois pontos distintos no plano, a combinação linear $\alpha \cdot p_1 + (1 - \alpha)p_2$, onde α é qualquer número real, é uma reta no plano (Figura 2.1b). Quando $0 \leq \alpha \leq 1$, se tem um segmento de reta no plano, que tem p_1 e p_2 como pontos extremos [4].

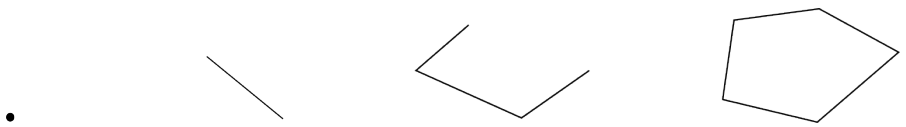


Figura 2.1: Objetos da Geometria Computacional: (a) ponto, (b) reta, (c) linha poligonal, (d) polígono.

Quando temos um conjunto de segmentos de retas no plano interligadas em sequência através de um dos pontos extremos, temos uma linha poligonal (Figura 2.1c). Quando uma linha poligonal é fechada, ou seja, o último segmento de reta do conjunto se conecta com o primeiro, sendo que nenhuma faz interseção com outra em um ponto que não seja de seus extremos, temos um polígono (Figura 2.1d).

2.2 Estrutura de Dados: DCEL

A DCEL, *Doubly-Connected Edge List*, é uma estrutura de dados para armazenar informações sobre a topologia de uma superfície em duas dimensões (possivelmente localizada no espaço em três dimensões, como na Figura 2.2) de forma eficiente. As superfícies assim representadas normalmente são consideradas equivalentes às subdivisões planares, apesar de não necessariamente com a mesma capacidade de representação. A superfície em questão é composta por faces (polígonos), bordas (segmentos que não se cruzam e são fronteiras entre duas faces

adjacentes) e vértices (fronteiras entre duas bordas adjacentes). A Figura 2.3 mostra um polígono 2.3a e sua respectiva DCEL 2.3b representada graficamente.

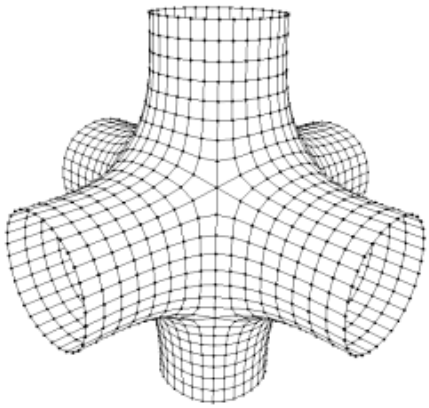


Figura 2.2: Exemplo de uma superfície em duas dimensões localizada no espaço em três dimensões. Figura retirada de [12].

O conceito de aresta, base da estrutura de uma DCEL, é apresentado de uma forma menos intuitiva. Cada aresta passa a ser representada como um par de semi-arestas (*half-edges*), ditas gêmeas, com os seguinte ponteiros:

- Para seu vértice de origem;
- Para a *half-edge* gêmea, chamada de *twin*;
- Para a próxima *half-edge*, cujo vértice de origem coincide com o seu vértice de destino;
- Para a face, que é o polígono localizado à esquerda desta *half-edge*, resultante do circuito de *half-edges*.

Esta estratégia organizacional implica que cada *half-edge* tem uma orientação, e que esta orientação é sempre oposta à orientação de sua *half-edge twin*, e se precisarmos saber o nó destino de uma *half-edge*, por exemplo, basta buscar o nó de origem da *half-edge twin*. Note que esta estrutura é de tamanho fixo. Mesmo para malhas compostas por triângulos, quadriláteros e polígonos em geral, as *half-edge* contêm a mesma quantidade de informação.

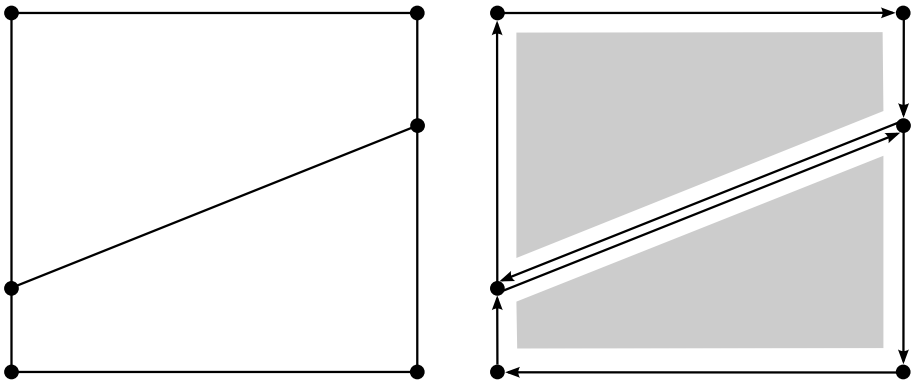


Figura 2.3: Exemplo de um polígono (a) e a representação gráfica da DCEL que o descreve (b).

A face, apontada por uma *half-edge*, é também uma estrutura própria de dados que possui um ponteiro para uma *half-edge* qualquer dentre aquelas que apontam para ela como sua face. A iteração da lista da estrutura de faces permite acesso a todos os objetos da DCEL, sejam elas mesmas ou os elementos que os constituem: os vértices e *half-edges*. Na prática, isso significa que os objetos estão em um *array*, lista ligada, ou algum outro tipo de estrutura de dados que permite a varredura ordenada, e que dependendo da nossa necessidade, permite a inserção e exclusão de objetos.

Até então, descrevemos todas as bordas como tendo uma face de cada lado. No entanto, se temos uma fronteira com a região externa, teremos arestas que possuem uma face de um lado e nada do outro lado. Para manter a estrutura consistente introduzimos faces especiais, de tamanho infinito, que não será considerada pelos iteradores, mas continuará acessível para comparação e testes, mantendo a estrutura consistente. Ela é, portanto, a face que está fora de todas as bordas do limite.

A Figura 2.4 ilustra as três estruturas básicas de uma DCEL: 2.4a estrutura do vértice, 2.4b estrutura da aresta e 2.4c a estrutura da face. Organizando desta forma, temos acesso a qualquer elemento que compõe a DCEL, além da possibilidade de iterar em todos eles.

Por fim, em grande parte das aplicações que utilizam DCEL, é comum armazenar dados associados aos objetos base, seja ele uma face, vértice ou *half-edge*. Estes dados podem

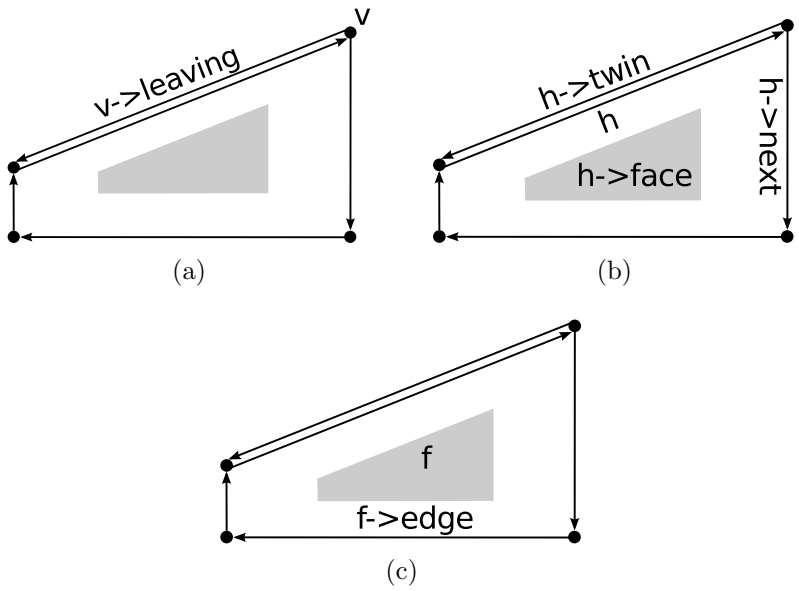


Figura 2.4: Estruturas que compõem uma DCEL: (a) estrutura do vértice, (b) estrutura do *half-edge*, (c) estrutura da face.

ser computacionalmente caros o suficiente a ponto de justificar seu cálculo e armazenamento junto ao objeto (como por exemplo a normal de uma face ou vértice), ou podem ser dados temporários necessários a algum algoritmo de tratamento da estrutura (como por exemplo, um *flag* que informa que o objeto já fora visitado pelo iterador). Há inúmeras maneiras de lidar com dados associados e a solução adotada dependerá das necessidades do problema tratado: dados comuns para todos os objetos podem ser adicionados a eles ou armazenados em subclasses dos objetos base, enquanto que para dados menos frequentes talvez seja mais interessante a criação de estruturas auxiliares criadas sob demanda, como por exemplo uma lista indexada ou uma tabela *hash*, vinculando cada elemento a um objeto na DCEL.

2.3 Operações

2.3.1 Verificando se um ponto está dentro do polígono

Determinar se um ponto Q qualquer está dentro ou fora de um polígono é um dos problemas mais importantes da geometria computacional, sendo parte crucial de algoritmos de localização, tratamento de eventos de *mouse* (onde é necessário saber se o usuário clicou em determinada região) etc. Apesar da solução ser relativamente simples, devido a casos especiais e problemas numéricos uma implementação verdadeiramente robusta é difícil de alcançar [4].

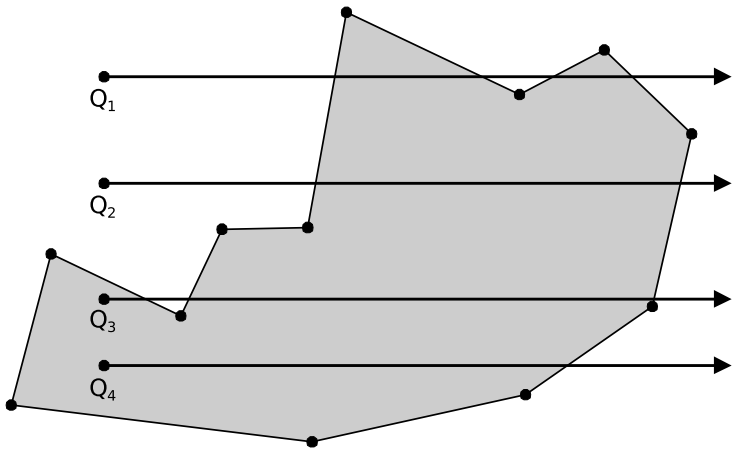


Figura 2.5: Verificando se Q está dentro ou fora do polígono. Q_1 e Q_2, cortando um número par de vezes as arestas do polígono, são considerados "fora", e Q_3 e Q_4, cortando um número ímpar de vezes as arestas do polígono, são considerados "dentro".

O princípio fundamental da solução, como descrito por Berg et al. [1], é traçar uma semirreta infinita na horizontal (aumentando o valor da coordenada x e mantendo o valor de y fixo) a partir de Q. Deve-se então contar quantas vezes esta semirreta é cortada por arestas do polígono tratado, alternando a resposta a cada corte entre "dentro" e "fora". Em outras palavras, se a semirreta traçada interceptar arestas do polígono um número ímpar de vezes (pontos Q_3 e Q_4 da Figura 2.5), este ponto pode ser considerado "dentro". Se for um número par (pontos Q_1 e Q_2 da Figura 2.5), pode ser considerado "fora". Este é o chamado teorema da curva de Jordan [11].

Para a maior parte dos casos a solução é tão simples quanto descrita acima. No entanto, existem casos degenerados que precisam de um tratamento um pouco mais complexo:

- A semirreta traçada passa por uma aresta do polígono (a aresta é parte da semirreta);

  Neste caso, a interseção não deve ser considerada.

- A semirreta passa por um vértice do polígono;

  Neste caso, a interseção deve ser considerada caso cada um dos segmentos adjacentes ao vértice esteja de um lado diferente da semirreta traçada. Se ambos estiverem do mesmo lado, a interseção deve ser desconsiderada.

- O ponto Q está sobre a fronteira ou vértice do polígono;

  Neste caso, a interseção é considerada, o que quer dizer que todos os pontos sobre as arestas do polígono são considerados "dentro" dele.

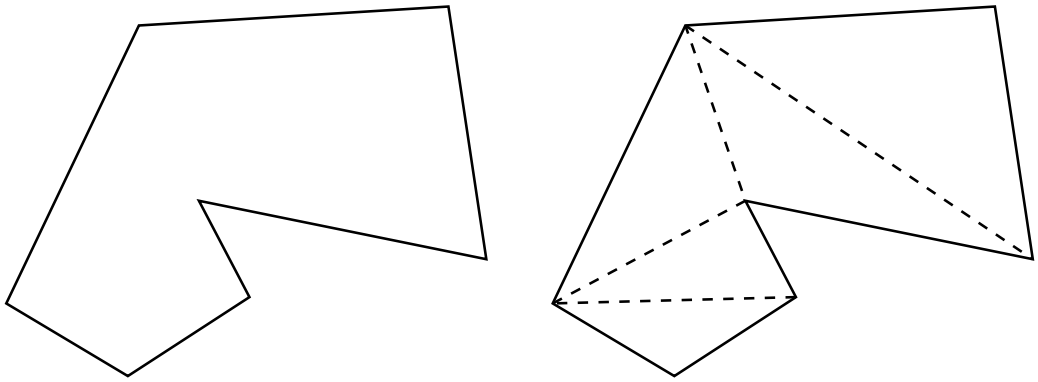
2.3.2 Verificando se um polígono é convexo

Um polígono é dito convexo quando, para qualquer par de pontos do conjunto, o segmento formado por eles estiver inteiramente contido no conjunto. Trata-se de uma verificação de implementação computacional bastante simples: considerando v_1, v_2, ..., v_m, os vértices do polígono, verifica-se o lado que v_{n+2} está em relação à reta que contém os vértices v_n e v_{n+1}, sendo que o lado a ser considerado na avaliação depende da orientação seguida para percorrer os pontos. Se o lado for o mesmo para todos os m pontos, este polígono é convexo. Caso contrário, é dito côncavo ou reflexo.

2.3.3 Calculando a área de um polígono

O cálculo da área de polígonos regulares é bastante simples, bastando poucos parâmetros sobre eles (por exemplo, o número de lados e o comprimento de um deles) e uma fórmula conhecida [5]. No entanto, aplicações geométricas do mundo real costumam utilizar polígonos irregulares, exigindo um conjunto de cálculos mais elaborados.

Partindo de um polígono irregular como o ilustrado na Figura 2.6a, precisamos, antes de mais nada, subdividi-lo em triângulos, como descrito na Seção 2.4.1. Uma configuração possível



(a) (b)

Figura 2.6: Subdivisão do polígono irregular em triângulos para o cálculo da área. (a) polígono irregular original, (b) polígono irregular subdividido em triângulos.

para esta subdivisão pode ser vista na Figura 2.6b, no entanto, qualquer outra configuração é igualmente correta, desde que os polígonos resultantes da subdivisão sejam triângulos. O cálculo da área total será a soma das áreas de todos estes triângulos.

Para o cálculo da área de cada triângulo, a fórmula elementar da geometria ("a área de um triângulo é igual à metade do produto entre sua base e sua altura") não é muito prática. Em vez dela, utilizaremos dois resultados equivalentes da álgebra linear:

- O produto vetorial de dois vetores, que determina a área de um paralelogramo, o dobro da área do triângulo que interessa, como ilustrado na Figura 2.7;
- O cálculo direto da área por meio de um determinante $3x3$.

Como descrito em Davis [4], a área do paralelogramo ilustrado na Figura 2.7 pode ser calculada, sendo U e V vetores (e lados do paralelogramo), por $|U \times V|$, que pode ser calculado a partir do seguinte determinante:

$$\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_U & y_U & z_U \\ x_V & y_V & z_V \end{vmatrix} = (y_U z_V - z_U y_V)\hat{i} + (z_U x_V - x_U z_V)\hat{j} + (x_U y_V - y_U x_V)\hat{k} \qquad (2.1)$$

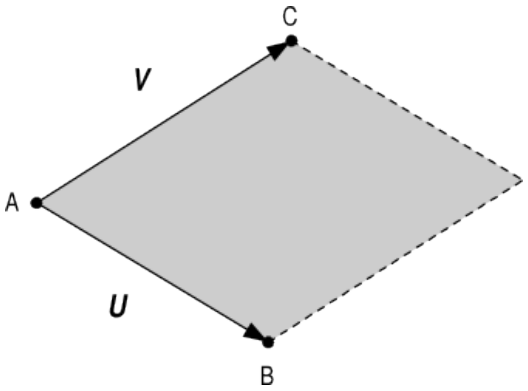


Figura 2.7: O módulo do produto vetorial dos vetores U e V fornece o dobro da área do triângulo ABC (Figura retirada de [4]).

onde $\hat{i}$, $\hat{j}$, $\hat{k}$ são vetores unitários nas direções x, y e z respectivamente. Como estamos tratando de vetores bidimensionais, temos $z_U = z_V = 0$, e portanto a área S do triângulo é dada por

$$S = \frac{(x_U y_V - y_U x_V)}{2} \tag{2.2}$$

Mas, na realidade, $U = B - A$, e $V = C - A$. Portanto, a expressão acima pode ser reescrita como

$$S = \frac{1}{2}(x_A y_B - y_A x_B + y_A x_C - x_A y_C + x_B y_C - y_B x_C) \tag{2.3}$$

A área calculada pela expressão acima será positiva se os vértices A, B e C formarem um circuito em sentido anti-horário, negativa se formarem um circuito no sentido horário, e exatamente zero se os três vértices estiverem alinhados. Substituindo a coordenada z por 1, o cálculo do determinante pode ser feito segundo a expressão:

$$S = \frac{1}{2}\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} = \frac{1}{2}(x_A y_B - y_A x_B + y_A x_C - x_A y_C + x_B y_C - y_B x_C) \tag{2.4}$$

Da mesma forma que o produto vetorial, a área será negativa se a sequência de vértices estiver orientada em sentido horário, e positiva caso contrário.

2.3.4 Calculando distâncias

O cálculo de distância é um dos mais elementares da geometria, sendo que dois deles nos serão particularmente importantes: a distância entre dois pontos quaisquer e a distância entre um ponto e um segmento de reta.

A primeira delas, entre dois pontos, pode ser calculada usando o teorema de Pitágoras: "em um triângulo retângulo, o quadrado do comprimento da hipotenusa (lado maior) é igual à soma dos quadrados dos comprimentos dos outros dois lados". Considerando as coordenadas dos dois pontos (x_1, y_1) e (x_2, y_2), respectivamente, a distância d será:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.5}$$

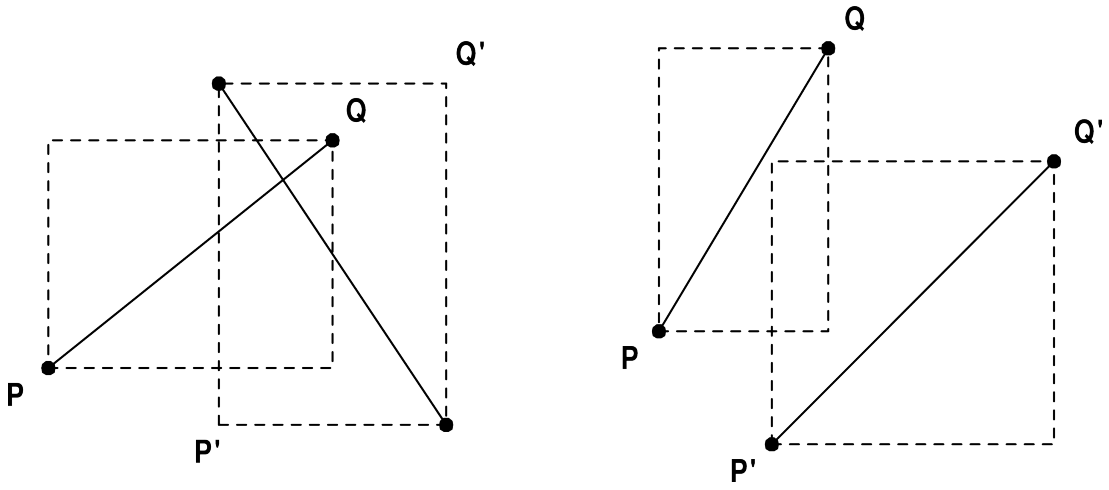
Já a distância entre um ponto Q e um segmento de reta definido por outros dois, A e B, pode ser calculada através do produto vetorial, calculando-se inicialmente a área S do triângulo AQB (Equação 2.4). Assim, a distância do ponto Q à reta definida pelos pontos A e B pode ser calculada como:

$$d = \frac{|S|}{dist(A, B)} \tag{2.6}$$

onde $dist(A, B)$ é a distância euclidiana entre os pontos A e B (Equação 2.5).

2.3.5 Verificando se dois segmentos se interceptam

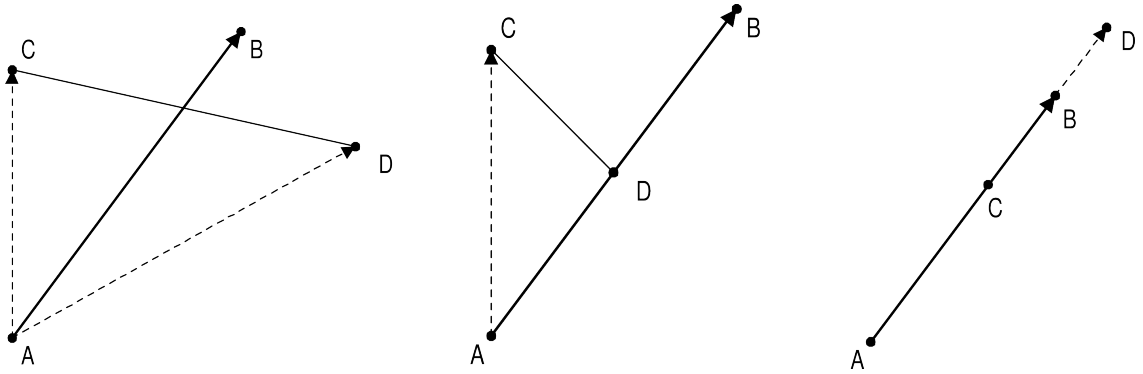
Devido ao baixo custo computacional, a verificação da interceptação entre segmentos é um teste preliminar em diversas outras tarefas, uma vez que o resultado deste teste simples pode evitar que outro muito mais complexo seja realizado. É o caso, por exemplo, da interseção de polígonos: se as arestas de um não interceptam as do outro, podemos concluir que não existirá polígono interseção antes mesmo de tentar computá-lo.



(a) (b)

Figura 2.8: Interseção de retângulos envolventes mínimos (Figura retirada de [4]).

Para realização do teste de interseção determinamos, inicialmente, se os retângulos mínimos definidos pelos segmentos se tocam. Se eles não se tocam, os segmentos também não terão interseção, mas não se pode afirmar o contrário, como acontece na Figura 2.8b. Uma vez certificado que os retângulos mínimos se tocam (o que acontece tanto na Figura 2.8a quanto em 2.8b), verificamos se os segmentos efetivamente se interceptam. Isto ocorre quando os pontos extremos de um segmento ficam de lados opostos da reta definida pelo outro, e vice-versa. Os resultados do produto vetorial têm que ter sinais opostos (Figura 2.9a). Se apenas um dos produtos for nulo, então um ponto extremo de um segmento está contido na reta



(a) (b) (c)

Figura 2.9: Possibilidades na interseção de segmentos (Figura retirada de [4]).

definida pelo outro (Figura 2.9b). Se ambos os produtos forem nulos, os segmentos são colineares (Figura 2.9c), com interseção (a possibilidade de colinearidade sem interseção foi descartada pelo teste dos retângulos). O teste precisa ser aplicado duas vezes, usando cada segmento como base, ou seja, não basta verificar se C e D estão de lados opostos da reta definida por AB, também é preciso verificar se A e B estão de lados opostos da reta CD [4].

2.3.6 Calculando o centro de massa de um polígono

O centro de massa ou centroide de um polígono (Figura 2.10a) pode ser obtido a partir da sua subdivisão em triângulos (Figura 2.10b), como descrito na Seção 2.4.1, calculando em seguida a média ponderada dos centros de gravidade dos triângulos (Figura 2.10c) usando suas áreas como peso (Figura 2.10d). O centro de gravidade de um triângulo ABC, por exemplo, é calculado da seguinte forma:

$$x_G = \frac{x_A + x_B + x_C}{3} \tag{2.7}$$

$$y_G = \frac{y_A + y_B + y_C}{3} \tag{2.8}$$

O centroide de um polígono formado por dois triângulos T_1 e T_2, cujos centroides são, respectivamente, (x_{G1}, y_{G1}) e (x_{G2}, y_{G2}) é o ponto (x_G, y_G), onde

$$x_G = \frac{x_{G1}S(T_1) + x_{G2}S(T_2)}{S(T_1) + S(T_2)} \tag{2.9}$$

$$y_G = \frac{y_{G1}S(T_1) + y_{G2}S(T_2)}{S(T_1) + S(T_2)} \tag{2.10}$$

O centroide do polígono pode ser determinado de maneira incremental, adicionando um triângulo e seu centroide por vez e calculando as coordenadas do centroide do conjunto. Assim, independente de quantos vértices um polígono possua, será possível identificar seu centro de massa.

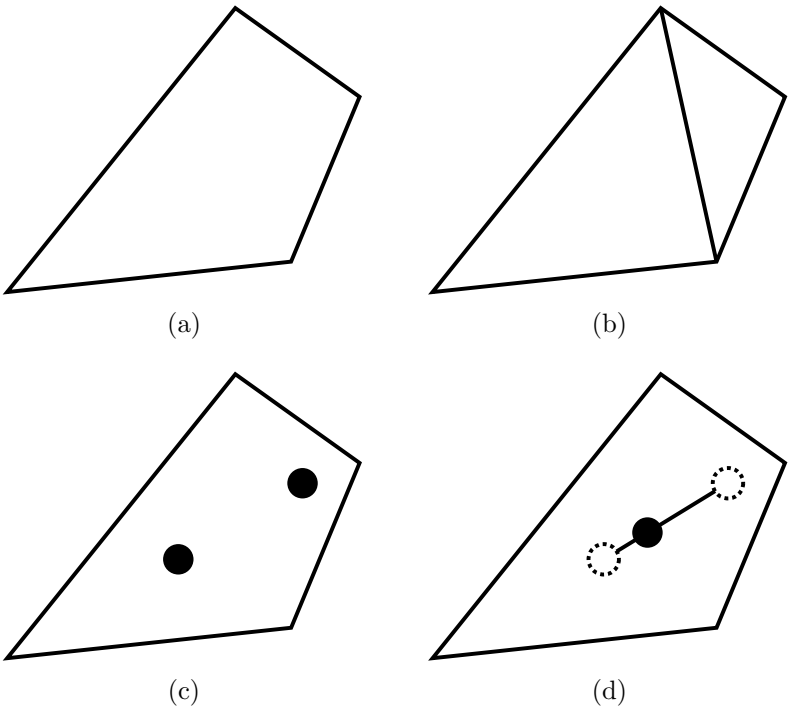


Figura 2.10: Identificação do centro de massa de um polígono. (a) polígono original, (b) polígono após a triangulação, (c) centros de massa dos triângulos, (d) média ponderada para localização do centro de massa do polígono.

2.4 Algoritmos geométricos

2.4.1 Triangulação de polígonos

O problema da triangulação de polígono é definido como um processo de se encontrar diagonais que o particione em triângulos [1], sendo que a solução pode não ser única, como pode ser visto na Figura 2.11. Trata-se de um problema tradicional cuja solução é parte da solução em diversos outros problemas da geometria inclusive deste trabalho, como por exemplo, o cálculo do centro de massa de um polígono (Seção 2.3.6) e o cálculo da área de um polígono (Seção 2.3.3). Segundo uma abordagem restrita, os triângulos resultantes podem ter apenas os vértices do polígono original. Já uma menos rigorosa permite que novos pontos sejam adicionados em qualquer lugar dentro do polígono para servir como vértices de triângulos. Além disso, os casos de triangulação de uma área poligonal com buracos são tratados separadamente.

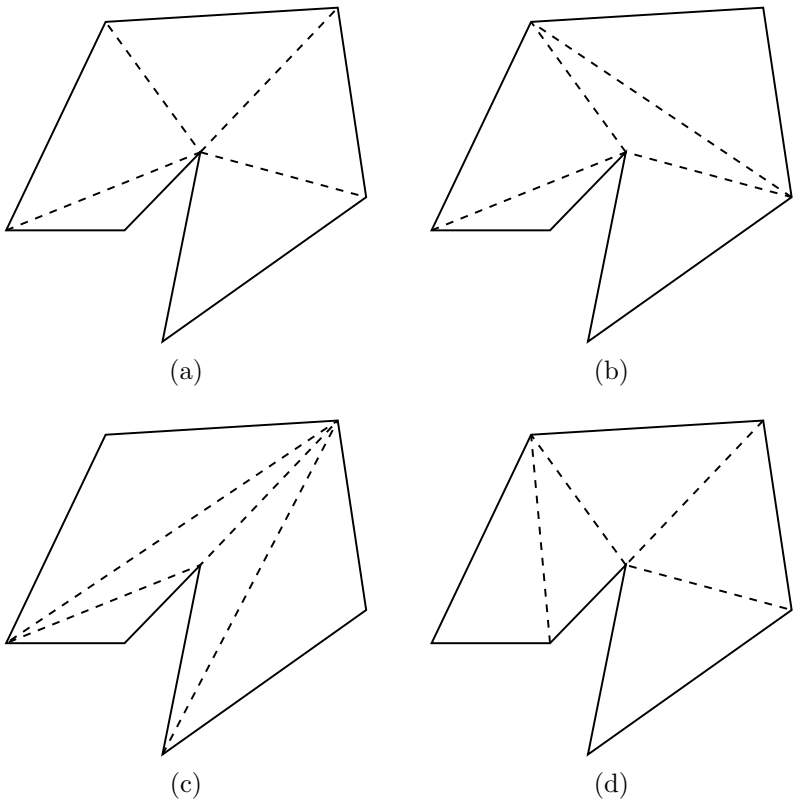


Figura 2.11: Algumas configurações possíveis para a solução do problema da triangulação em um mesmo polígono.

Desde o início do século passado, foram feitas muitas propostas diferentes para resolver o problema [21], basicamente tentando um resultado equivalente mas com custo computacional menor. Citaremos dois métodos: "corte de orelhas" e "partição em polígonos monotônicos".

O primeiro, apresentado por Meisters [22], é baseado no fato de que qualquer polígono simples (lados não adjacentes não se interceptam) com pelo menos quatro vértices e sem furos tem pelo menos duas "orelhas": triângulos com dois lados sendo as bordas do polígono e o terceiro completamente dentro dele, como visto na Figura 2.12. O algoritmo então consiste em encontrar e remover estes triângulos do polígono, resultando em um polígono que ainda preenche as condições iniciais. A cada iteração, um novo triângulo (aquele que é retirado) é descoberto, e o processo se repete até que o polígono final seja, também, um triângulo.

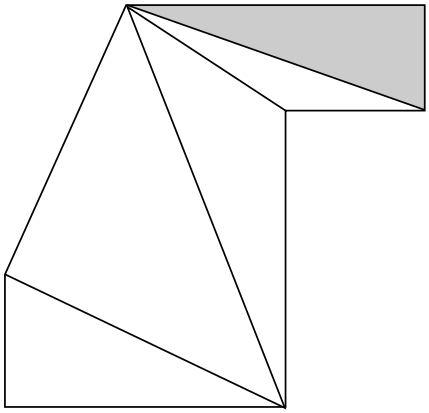


Figura 2.12: Triangulação de polígono segundo algoritmo "corte de orelhas".

Esta solução apesar de ser de fácil implementação, tem alto custo computacional ($O(n^2)$) e é restrita a polígonos sem furos. ElGindy et al. [7] então propôs uma evolução do trabalho de Meisters [22], criando uma tarefa de pré-processamento do polígono subdividindo-o em polígonos monotônicos antes da etapa de triangulação.

Polígonos monotônicos são polígonos simples sem buracos, com a particularidade de que, para uma determinada direção definida por uma reta, esta (ou qualquer reta que lhe seja paralela), não intercepta o polígono em mais do que dois pontos. No exemplo ilustrado na Figura 2.13, o polígono é monotônico em x: qualquer reta perpendicular ao eixo das abcissas intersepta o polígono em apenas dois pontos.

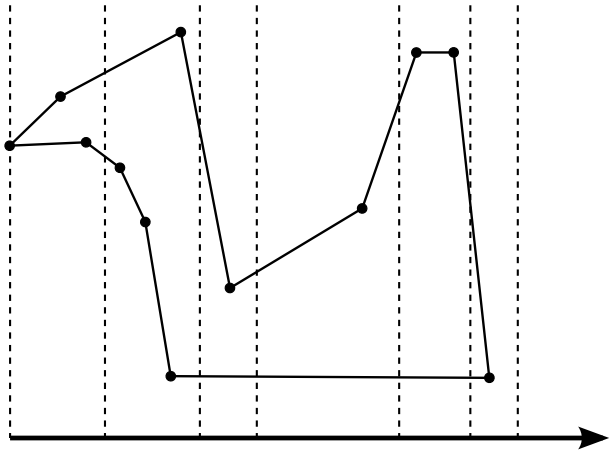


Figura 2.13: Polígono monotônico.

Uma abordagem para particionar um polígono qualquer em polígonos monotônicos, consiste em traçar uma reta de varredura e percorrer o polígono em uma direção qualquer, horizontal ou vertical. No exemplo ilustrado na Figura 2.14, percorremos verticalmente. Para cada ponto é verificado se os pontos vizinhos são ambos do mesmo lado de uma linha de varredura. Se eles forem, verifica-se a próxima linha de varredura no outro lado. O polígono deve, então, ser quebrado traçando um novo segmento entre o ponto original e um outro ponto sob a segunda linha, sendo estes pontos os pontos de divisão do polígono (Figura 2.14). O custo computacional desta tarefa é $O(n \log n)$.

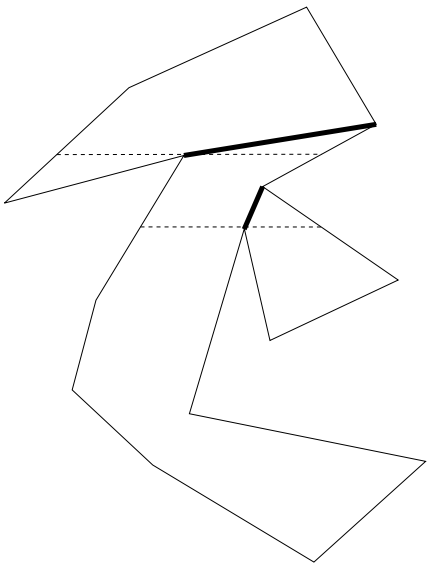


Figura 2.14: Partição de um polígono em polígonos monotônicos.

Uma vez tratando apenas de polígonos monotônicos, o processo de triangulação é simples e de custo $O(n)$. Trata-se de uma variação do algoritmo de varredura do plano. Inicialmente, os pontos são ordenados segundo sua coordenada x crescente. Percorre-se os pontos tentando triangular tudo que puder à esquerda do ponto corrente, adicionando-se diagonais. As regiões já trianguladas deixam de ser consideradas nos próximos passos. Este algoritmo é bastante intuitivo, e é ilustrado passo a passo na Figura 2.15.

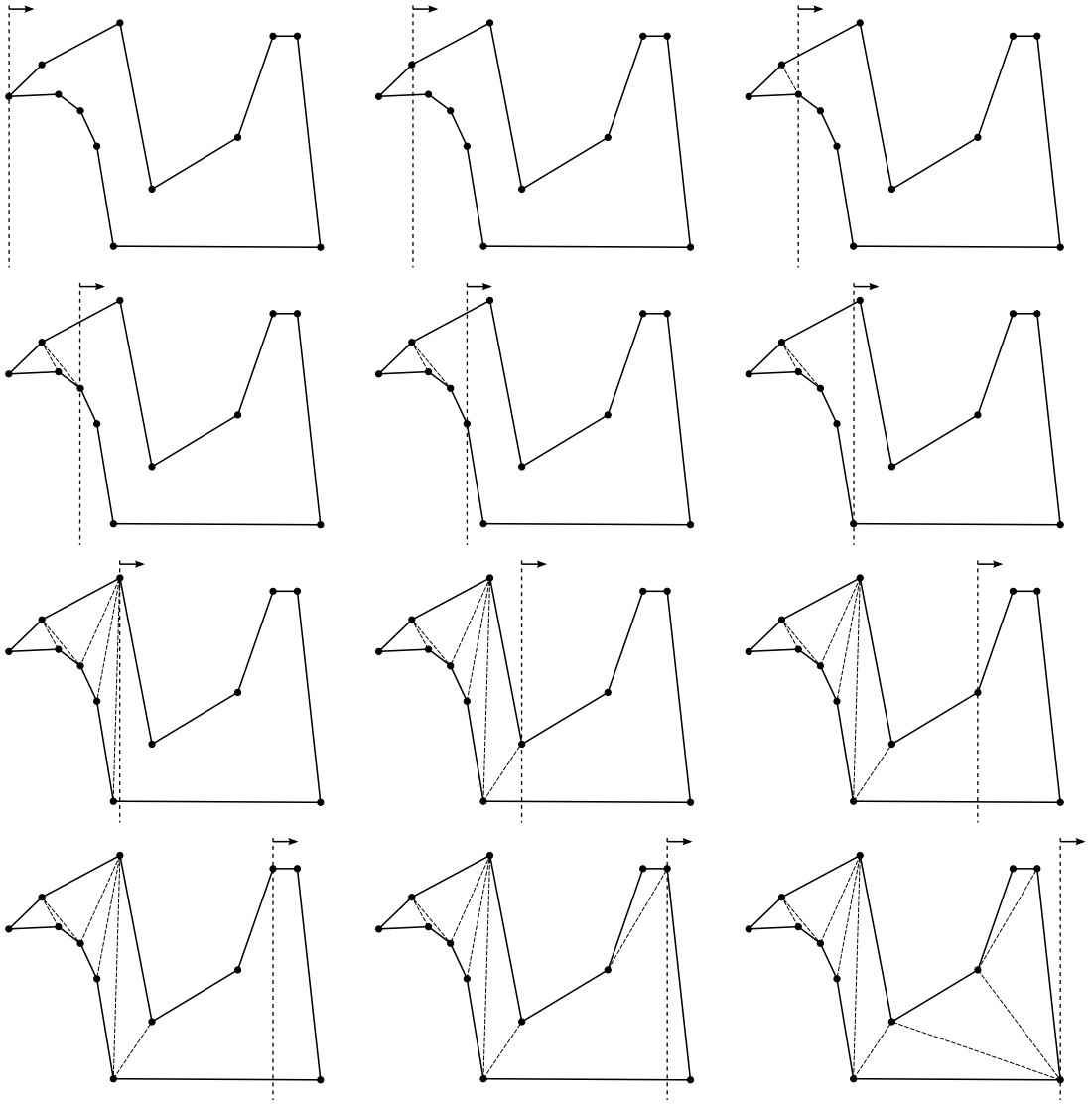


Figura 2.15: Passo a passo da triangulação de polígono monotônico.

2.4.2 Operações em polígonos: união, interseção e diferença

Interseção, união e diferença de polígonos são operações fundamentais usadas com frequência em aplicações de análises topológicas (de objetos, terrenos, etc.), recortes de polígonos em aplicações baseadas na computação gráfica, entre outras [4]. É comum que esta tarefa seja executada continuamente e com repetições intensas. Nossa maior preocupação, portanto, é equilibrar o desempenho e precisão. Para isso, a primeira tarefa realizada na maioria das aplicações que tratam destas operações é aplicar um passo preliminar para, talvez logo de início e sem grandes complicações, resolver o problema. Por exemplo, se os retângulos envolventes mínimos que contornam dois polígonos quaisquer que serão analisados nem mesmo se tocam, podemos concluir que estes polígonos não possuem interseção nem diferença, e que a união é o conjunto formado pelos dois polígonos disjuntos.

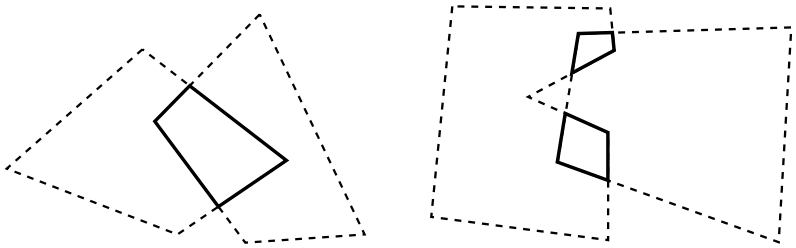
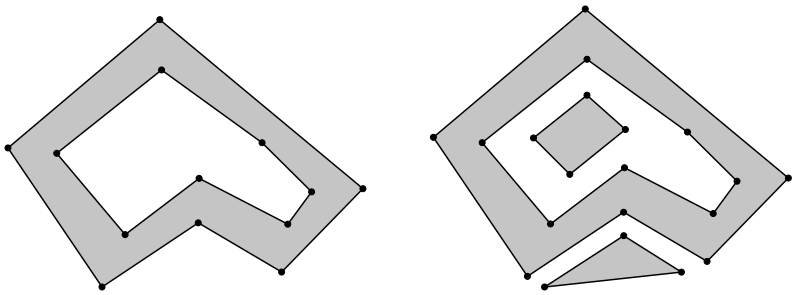


Figura 2.16: Interseção de polígonos. (a) interseção entre dois polígonos convexos sem furo, resultando em um único polígono também convexo e (b) interseção entre um polígono não convexo com outro convexo, resultado em dois polígonos.

De um modo geral, as operações fundamentais envolvendo dois polígonos podem gerar como resultado diversos polígonos, ou mesmo polígonos com buracos. Particularmente importante para este trabalho, a interseção de dois polígonos sem buracos será também um ou mais polígonos igualmente sem buracos (Figura 2.16), ou vazio, caso não exista interseção. Para casos especiais onde ambos os polígonos são convexos e se interceptam (Figura 2.16a), o resultado será um único polígono convexo [24].

Discutiremos com mais detalhes uma dentre muitas possíveis abordagens para a solução do problema das operações com polígonos, proposta por Margalit and Knott [20]. Ela permite realizar operações de interseção, união e diferença com custo computacional final $O(n \log n)$, permitindo que os polígonos sejam do tipo:



(a) (b)

Figura 2.17: Exemplo de polígono buraco (a) e polígono ilha (b).

- buraco, quando um polígono é tal que a região fora do seu contorno pertença ao objeto modelado, e a região dentro do contorno não, como por exemplo o polígono interno ilustrado na Figura 2.17a. É possível que um mesmo objeto possua mais de um buraco;
- ilha, quando um polígono é tal que a região dentro do seu contorno pertença ao objeto modelado, e a região fora do contorno não. É possível que um mesmo objeto possua mais de uma ilha e, neste caso, ele é o agrupamento de mais de um polígono, como ilustrado na Figura 2.17b.

O algoritmo possui seis etapas listadas a seguir, sendo P e Q os polígonos tratados:

1. Normalizar a orientação dos polígonos de entrada P e Q, invertendo a orientação de Q dependendo do tipo de operação e da natureza (ilha ou buraco) dos dois polígonos envolvidos, que deve estar de acordo com a Tabela 2.1;

Polígonos		Operações			
P	Q	$P \cap Q$	$P \cup Q$	$P - Q$	$Q - P$
ilha	ilha	manter	manter	inverter	inverter
ilha	buraco	inverter	inverter	manter	manter
buraco	ilha	inverter	inverter	manter	manter
buraco	buraco	manter	manter	inverter	inverter

Tabela 2.1: Ajuste na orientação dos dois polígonos envolvidos de acordo com a operação a ser realizada.

2. Usando o algoritmo descrito na Seção 2.3.1, classificar cada vértice, verificando se cada um está dentro, fora ou na fronteira do outro polígono inserindo-os, de acordo com o resultado deste teste, em duas listas circulares, P_L e Q_L, onde aparecerão em sequência de modo a definir as arestas por adjacência;

3. Encontrar as interseções entre arestas dos dois polígonos usando o algoritmo descrito na Seção 2.3.5. Os pontos de interseção devem ser inseridos na lista PL ou QL apropriada, respeitando sua posição segundo os vértices existentes e identificando-os como vértices de fronteira. Percebemos a partir desta etapa, que o conjunto de segmentos originais deu lugar a um novo conjunto, formado pelos vértices originais e de fronteira;

4. Classificar cada segmento do novo conjunto (par de pontos) como "interno", "externo" ou "de fronteira", analisando para isto o outro polígono. Se um segmento possuir um vértice interno ao outro polígono, ele será considerado "interno", assim como se possuir um vértice externo a ele, será considerado como "externo". Lembre-se que devido a etapas anteriores, não existe segmento que cruza o outro polígono. Para casos especiais onde ambos os vértices estão na fronteira, é necessário ainda verificar se existe algum ponto intermediário fora do polígono, classificando-o como "externo", ou dentro, classificando-o como "interno". Caso contrário, será de fato "de fronteira";

5. Selecionar e organizar as arestas para formar os polígonos de resultado, consolidando as listas PL e QL em uma única, chamada RL. Para esta consolidação, somente as arestas que interessam para a operação desejada (união, interseção ou diferença) deverá ser considerada.

6. Identificar o caminho fechado formado pela sequência de segmentos existentes em RL (possivelmente desordenados na lista), criando assim o polígono resultante.

2.4.3 Diagrama de Voronoi

O Diagrama de *Voronoi*, criado pelo matemático ucraniano *Georgii Feodosevich Voronoi* [8], é uma decomposição de um espaço em regiões de acordo com a distância a determinados pontos, determinando assim, a área de cobertura de cada ponto. Na prática, podemos dizer que qualquer coordenada dentro de uma célula do diagrama de *Voronoi* está mais perto do ponto gerador desta célula do que de qualquer outro ponto do conjunto gerador [1].

A construção do diagrama pode ser melhor compreendida observando a evolução incremental, ponto a ponto. Vamos considerar inicialmente dois pontos, P_1 e P_2, como pode ser visto na Figura 2.18a. Vamos imaginar uma área de abrangência de cada ponto, e uma reta dividindo as duas áreas existentes até então, perpendicular ao segmento P_1P_2. Todos os pontos da reta são igualmente próximos a P_1 e a $P2$. Pontos no semi-plano que contém P_1 constituem o polígono de *Voronoi* correspondente a P_1, e analogamente o outro semi-plano corresponde a $V(P_2)$.

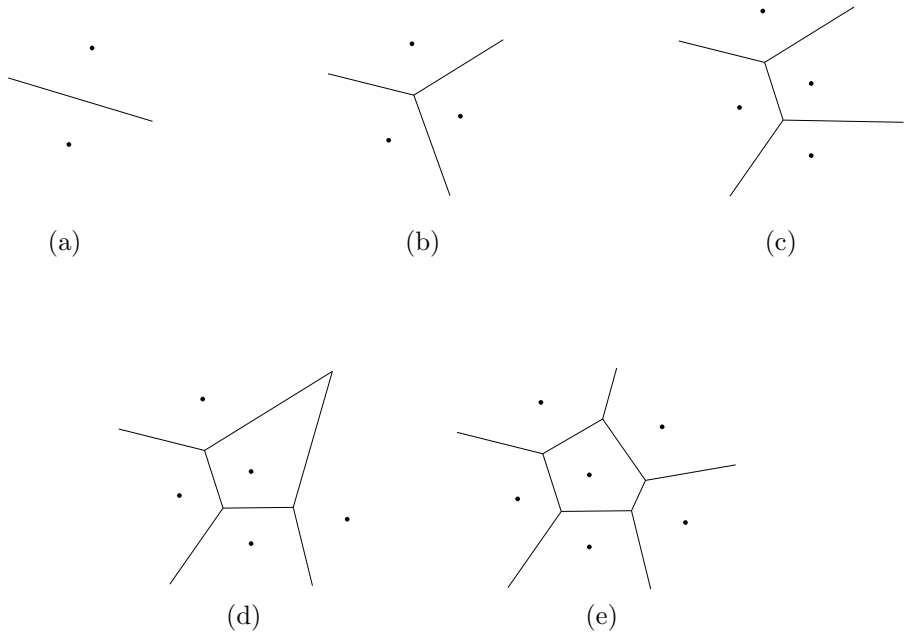


Figura 2.18: Evolução incremental do algoritmo de *Voronoi*, sendo (a) com dois pontos, (b) com três pontos, (c) com quatro pontos, (d) com cinco pontos, (e) com seis pontos.

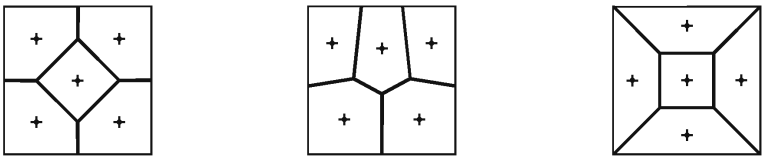
Inserindo um novo ponto, P_3, percebemos que o diagrama de *Voronoi* será reconfigurado pela inserção de novas semirretas, que cortam na perpendicular todos os segmentos do arranjo com os três pontos: P_1P_2, P_2P_3, P_1P_3, como pode ser visto na Figura 2.18b.

De forma geral, para um número qualquer de pontos, fica intuitivo o entendimento de como serão inseridos novos segmentos ao conjunto, sempre levando em conta as mediatrizes dos segmentos definidos entre cada par de pontos P_i e P_j, denotada como M_{ij}.

Seja S_{ij} o semi-plano definido por M_{ij} e que contém P_i. Então S_{ij} contém todos os pontos do plano que estão mais próximos de P_i do que de P_j. Para obter o polígono de *Voronoi* de P_i, é necessário combinar todos os semi-planos S_{ij} com $i \neq j$, e portanto

$$V(P_i) = \bigcap_{i \neq j} S_{ij} \tag{2.11}$$

Como semi-planos são convexos, não existe nenhum segmento definido entre dois pontos do semi-plano e que contém pontos que não pertençam a ele. Como a interseção de conjuntos convexos é também um conjunto convexo (Veja Seção 2.4.2), qualquer polígono do diagrama de *Voronoi* é também convexo.



(a) (b) (c)

Figura 2.19: Exemplo de três diagramas de *Voronoi* centroidais com cinco pontos em um quadrado.

Um caso particular de diagrama de *Voronoi*, chamado centroidal, possui a propriedade da coincidência de coordenadas entre os pontos de geração de cada célula de *Voronoi* e o que indica seu centro de massa. A Figura 2.19 ilustra três configurações possíveis deste diagrama, particularmente útil no algoritmo *K-means* e *Lloyd*, este último usado neste trabalho e descrito na Seção 2.4.4.

2.4.4 Algoritmo de Lloyd

O algoritmo de *Lloyd*, também chamado de iteração *Voronoi* ou relaxamento, é um algoritmo usado no espaço euclidiano para agrupamento de pontos em determinado número de categorias [6]. O algoritmo possui quatro etapas, descritas a seguir, que são executadas iterativamente até a convergência para um diagrama de *Voronoi* centroidal. Na Figura 2.20 podemos acompanhar graficamente a alteração das coordenadas do conjunto de pontos em cada iteração.

1. Computa o diagrama de *Voronoi* do conjunto de pontos atual;

2. Para cada célula do diagrama, calcula seu centro de massa;

3. Desloca o ponto gerador para a posição do centro de massa encontrado na etapa anterior;

4. Verifica se o deslocamento médio dos pontos geradores do conjunto é minimamente aceitável para a aplicação, interrompendo o processo de refinamento caso o seja.

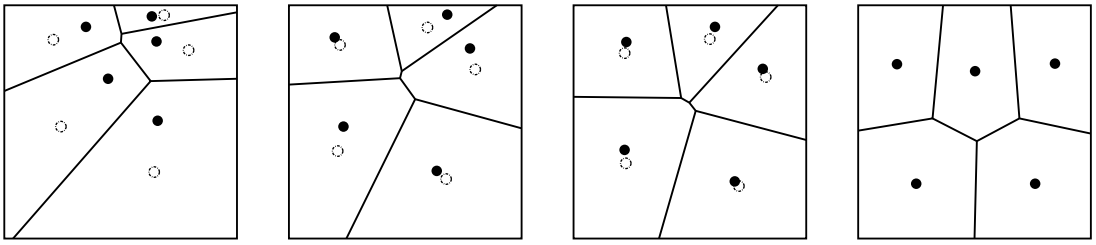


Figura 2.20: Exemplo do algoritmo de *Lloyd* em quatro iterações diferentes. O diagrama de *Voronoi* do conjunto de pontos atuais em cada iteração é mostrado limitado por um quadrado. Os pontos do conjunto são ilustrados por um círculo preenchido, enquanto que o centro de massa, para onde os pontos serão deslocados, são ilustrados por círculos tracejados. Note que a cada iteração a distância entre eles fica menor, até que na última iteração é encontrado o diagrama de *Voronoi* centroidal.

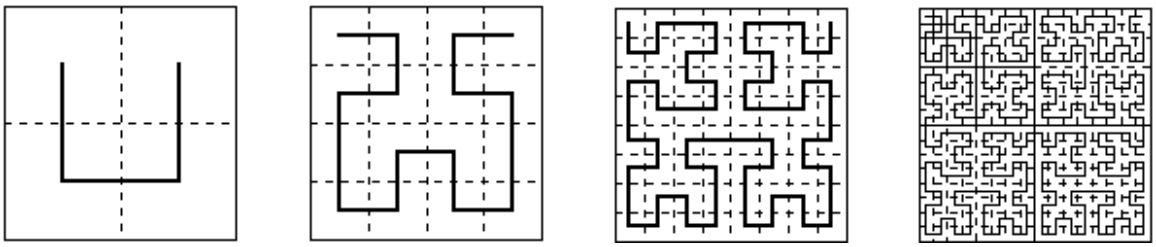
A menos que um conjunto de pontos tenha sido gerado segundo a organização de um diagrama de *Voronoi* centroidal, é de se esperar que as coordenadas de cada ponto sejam diferentes da coordenadas do ponto gerador. No entanto, é objetivo do algoritmo de *Lloyd* iterar até que a distância média dos pontos geradores ao centroide de sua célula seja menor que um valor heurístico previamente definido, ajustando as coordenadas dos pontos geradores com as coordenadas do centro de massa da respectiva célula *Voronoi*. O valor da média é calculado da seguinte forma:

$$d_{average} = \frac{\sum_{i=0}^{n-1}(\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2})}{n} \tag{2.12}$$

Por definição, o algoritmo consegue aproximar cada vez mais do posicionamento ótimo para seus pontos e, portanto, o valor da média dos deslocamentos tende a ser menor a cada iteração.

2.4.5 Curva de Hilbert

A curva de Hilbert é uma curva fractal (figuras formadas pela repetição de padrões de determinadas funções ou de modelos geométricos) do tipo contínua, descrita pela primeira vez em 1891 por *David Hilbert*.



(a) (b) (c) (d)

Figura 2.21: Curva de *Hilbert*: (a) figura geradora com o primeiro nível da curva, (b) com o segundo, (c) com o terceiro e (d) com o quarto. Figura retirada de [25].

A figura geradora é um quadrado subdividido em quatro novos quadrados (Figura 2.21a), com pontos centrais unidos formando uma lista duplamente encadeada. Intuitivamente, partindo de um único quadrado de lado 1, verificamos que o $k - ésimo$ passo possui 4^k quadrados, com comprimento de lado $\frac{1}{2^{k-1}}$ [23]. A propriedade mais relevante desta curva é a divisão dos espaços em quadrantes. Somente após um dos quadrantes ter sido totalmente percorrido, é que o próximo será percorrido, e assim sucessivamente até a imagem ter sido completamente percorrida.

No escopo deste trabalho, a curva de *Hilbert* se mostrou bastante apropriada, uma vez que sua navegação, quadrante a quadrante, segue o mesmo molde da divisão recursiva do espaço quadrangular usado como métodos de geração de pontos aprensentado na Seção 3.2.3. Assim, quando precisamos iterar entre os pontos da nuvem gerada por este método, o fazemos seguindo a mesma ordem percorrida pela curva de *Hilbert*.

Capítulo 3

Geração de Nuvem de Pontos para Métodos sem Malha

Neste capítulo, descreveremos a proposta de geração de nuvem de pontos deste trabalho. Em linhas gerais, identificamos três etapas distintas seguidas até a solução: a modelagem gráfica dos objetos, a geração em si e o refinamento. Apresentaremos a visão geral de cada uma delas, sobretudo quanto a seus princípios de funcionamento e decisões de projeto.

3.1 Modelagem geométrica

Para proceder com a geração da nuvem de pontos, precisamos definir os contornos do objeto que, neste trabalho, são apresentados em um espaço bidimensional. As coordenadas de todos os pontos que integram a nuvem estão, portanto, dentro dos limites de polígonos definidos, de forma independente para cada um deles. Apesar de ser possível buscar as coordenadas de todos os vértices ordenados do contorno de cada objeto em arquivo, partiremos do princípio que ele ainda não existe, e que o usuário fará a modelagem utilizando as ferramentas oferecidas pela aplicação.

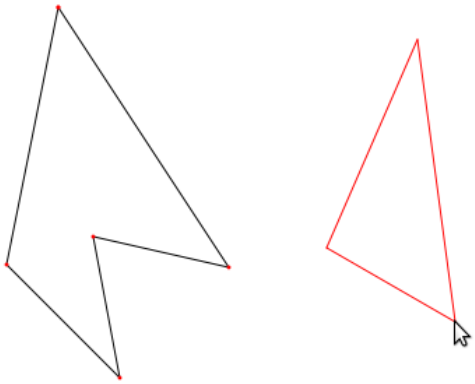
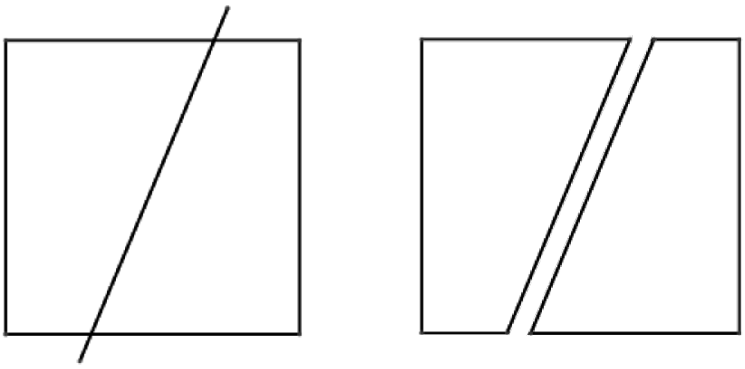


Figura 3.1: Momento de inserção de segmentos utilizando os recursos de desenho da aplicação. O polígono à esquerda já fora finalizado, enquanto que o à direita ainda está recebendo pontos a partir de cliques do mouse (tratados como vértices).

O processo de desenho pode ser dividido em duas etapas distintas. Na primeira, inserimos um conjunto de segmentos isolados ou interconectados (formando caminhos fechados como pode ser visto na Figura 3.1). Neste momento podem existir sobreposições e desconexão de vértices dentre os segmentos inseridos. O fato de uma sequência de segmentos ser desenhada como um caminho fechado não garante que o polígono resultante, onde será gerada a nuvem de pontos, será tal qual desenhado. É possível até mesmo que sejam gerados mais de um polígono a partir deste desenho.

Do emaranhado de segmentos inseridos precisamos inferir os polígonos resultantes. Todos os caminhos fechados existentes, independentemente de como se chegou a eles, devem ser considerados como polígono. Assim sendo, um quadrado com um segmento atravessado (Figura 3.2a) dará origem a dois polígonos distintos (Figura 3.2b).

Na figura 3.3 podemos analisar uma entrada um pouco mais complexa. No desenho central, um emaranhado de segmentos inseridos livremente, percebe-se cinco elementos distintos: três triângulos e dois segmentos. A partir dele podemos inferir oito polígonos distintos, todos eles formados por um conjunto conexo de segmentos e por vértices de grau dois - exatamente dois dos segmentos do polígono conectado a cada um de seus vértices.



(a) (b)

Figura 3.2: A partir do desenho de, por exemplo, um quadrado com um segmento atravessado feito pelo usuário (a), precisa-se inferir a existência de dois polígonos distintos (b).

3.1.1 Identificação de Polígonos

O processo de identificação de polígonos consiste, basicamente, em decompor o emaranhado inserido C e recompor em uma DCEL, onde poderemos iterar entre as faces considerando apenas as que forem identificadas como caminho fechado. Para entender melhor este processo, vamos primeiramente analisar a Figura 3.4, que nos mostra os elementos da estrutura de dados que queremos identificar.

Uma possível abordagem é trabalhar com uma construção incremental da DCEL, a partir da inserção um a um dos segmentos de reta de C. A cada inserção, precisamos encontrar as interseções e atualizar a DCEL, como pode ser visto na Figura 3.5. Intuitivamente, percebemos que o tempo gasto para inserir uma reta é proporcional ao número de retas que são atravessadas neste processo.

Outra abordagem, implementada na CGAL (detalhes desta biblioteca no Apêndice A), é a decomposição de C no máximo de curvas monotônicas possíveis, obtendo-se um conjunto de curvas C'. Formalmente, uma curva monotônica é aquela cuja derivada não muda de sinal. No entanto, neste caso em particular, podemos simplificar o conceito dizendo que é cada segmento de rete inserido pelo usuário, contendo em suas extremidades dois dos vértices do conjunto, podendo ou não ser interceptado por outra curva monotônica.

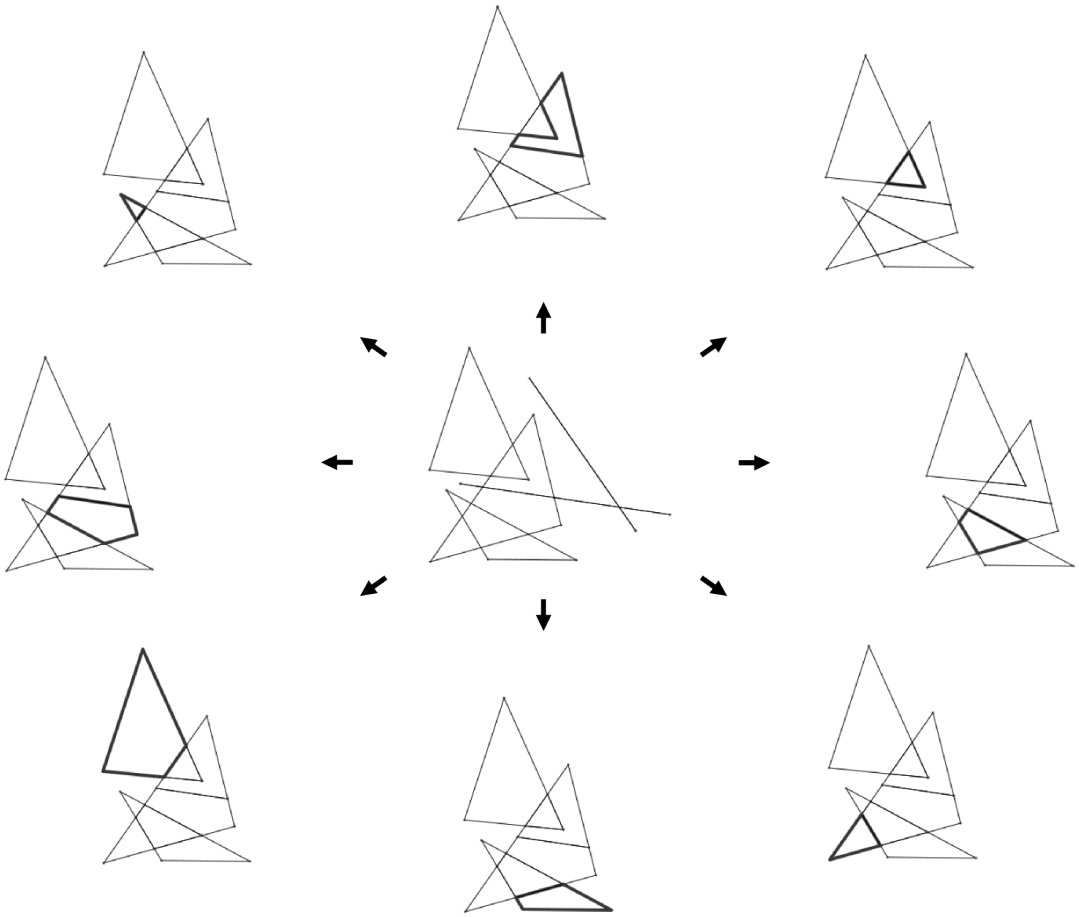


Figura 3.3: Exemplo de desenho um pouco mais complexo (centro), onde foram inferidos oito polígonos (marginais).

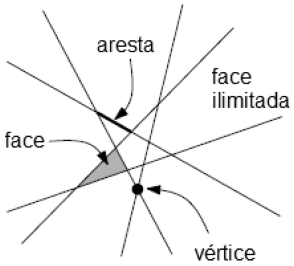


Figura 3.4: Arranjo de segmentos de reta e identificação dos elementos de interesse para composição de uma DCEL.

Em seguida, decompõe-se cada curva de C' no máximo de segmentos conectados, mas que não cruzam com qualquer outro segmento, obtendo-se C'' [9]. Consideramos os conjuntos C' e C'' como níveis em uma hierarquia de curvas, onde a união das sub-curvas em cada nível é a

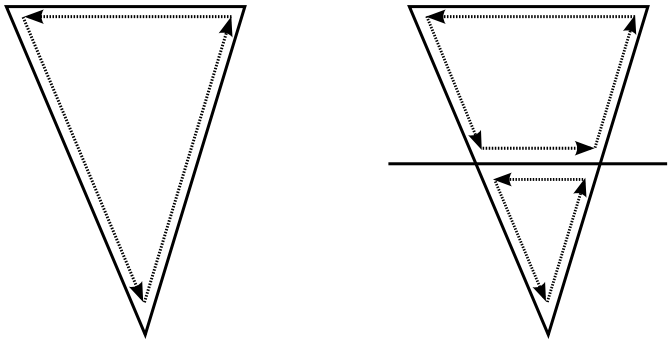


Figura 3.5: Atualização da DCEL após a inserção de um novo segmento de reta.

curva original em C'', podemos então armazená-los em uma árvore hierárquica. Esta estrutura consiste geralmente em três níveis:

1. Curva da raiz da árvore, que mantém a curva original C;
2. Sub-curva da raiz, que são oriundos da decomposição da curva original;
3. Bordas ou folhas da árvore, que mantêm as curvas correspondentes às bordas do mapa planar induzido pelo arranjo.

A Figura 3.6 mostra a divisão hierárquica com as curvas correspondentes de um exemplo de C. A árvore de hierarquia resultante nos permite cruzar as curvas sem perda de informação, pois tanto as curvas originais quanto as sub-curvas intermediárias são armazenadas.

A partir desta estrutura, fica simples compor uma DCEL, bastando iterar entre os elementos folhas, C'' encontrando as continuidades e criando incrementalmente a DCEL, de forma semelhante da realizada para tratamento de interseção de polígonos, descrita com mais detalhes na Seção 2.4.2.

3.1.2 Considerações Sobre Furos em Polígonos

Existe uma configuração especial onde um ou mais polígonos estão completamente dentro de outro, sem compartilhar nenhum vértice nem interceptar os segmentos do contorno (Figura

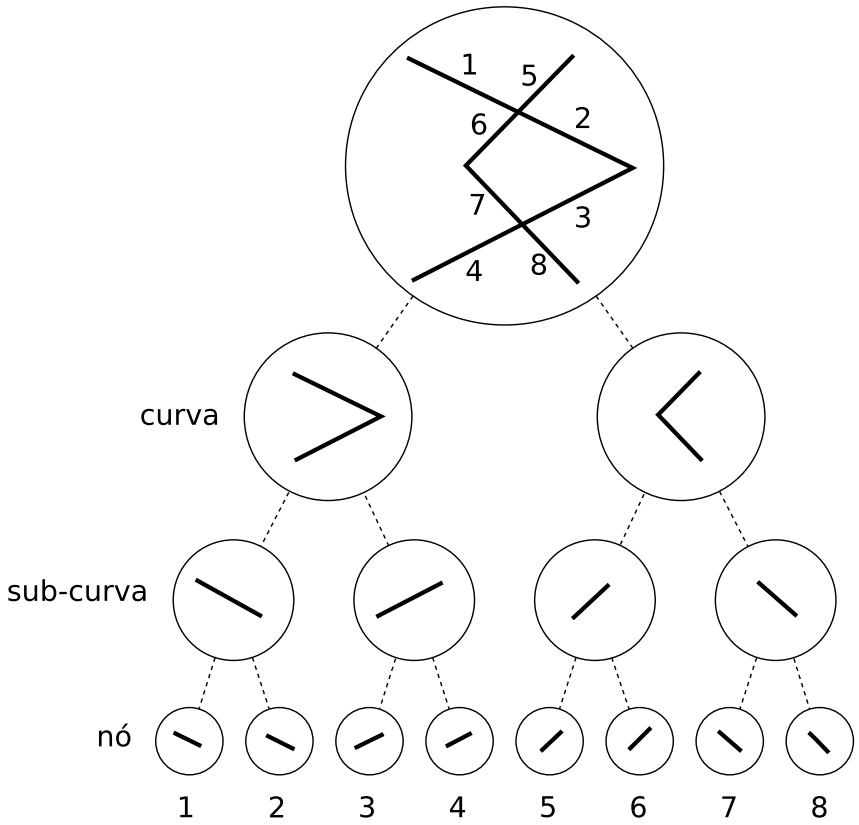


Figura 3.6: Subdivisão hierárquica de um arranjo de segmentos formado por duas linhas poligonais. Note que as bordas do arranjo são numeradas de acordo com sua ordem na árvore [9].

3.7). São, por exemplo, ocorrências de furos dentro de um objeto modelado ou mesmo mudança de material. Em soluções de problemas reais, a área do polígono interno pode ter uma densidade de pontos diferente da externa, inclusive densidade nula, quando não haverá distribuição de pontos. Isso quer dizer que, no exemplo ilustrado na Figura 3.7, teríamos dois polígonos, cada um com suas próprias características.

Inicialmente, o polígono interno será tratado como qualquer outro. No entanto, precisamos considerar algumas particularidades, sendo P o polígono externo e P' o interno:

- Se a densidade de nuvem de pontos de P' for maior do que a de P, a distribuição dos pontos sobre as arestas de P', fronteira com P, seguirá a densidade de P' e vice versa. Este mesmo comportamento acontece quando dois polígonos fazem fronteira, mas sem

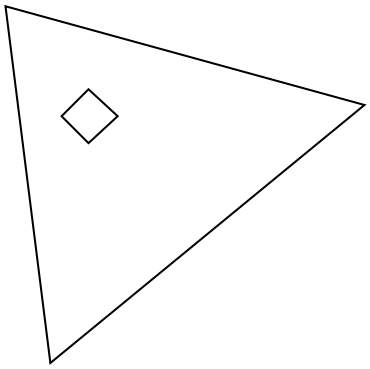


Figura 3.7: Exemplo de polígono "furado".

que nenhum deles seja interno ao outro, ou seja, compartilham no máximo $n-1$ arestas, sendo n o número de arestas do contorno;

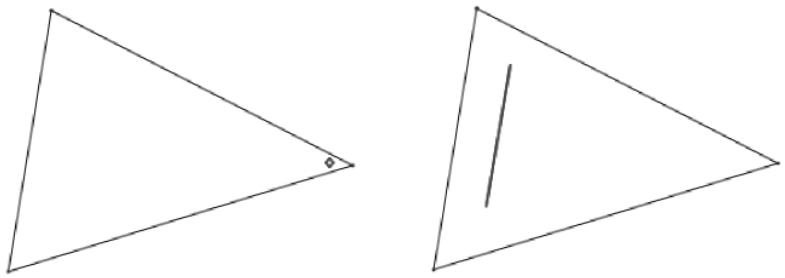
- Se a densidade de P' for nula, ainda assim serão distribuídos pontos na fronteira com P, seguindo a densidade de P;
- A área ocupada pelo polígono interno, independente de sua densidade, não pode ser considerada na distribuição de pontos do polígono externo;

Além de independentes, as nuvens dos dois polígonos podem também ter densidade diferentes, o que permite que a densidade do polígono interno seja zero, no caso de ser considerado um furo. A única particularidade desta configuração é que, no polígono externo, a região ocupada pelo polígono interno será desconsiderada na distribuição da nuvem de pontos. Já as margens serão tratadas como qualquer outra margem compartilhada entre diferentes materiais. Discutiremos com mais detalhes o tratamento dado às arestas de fronteira na Seção 3.2.

3.1.3 Regiões de Interesse Diferenciado

É comum existir em grande parte dos problemas do mundo real uma área de particular interesse no objeto representado pelo polígono. O uso de uma malha uniforme de pontos para discretizar o problema diferencial não é adequado nestes casos, pois para que seja seguida a mesma densidade da distribuição de pontos da região de maior interesse ao longo de toda a superfície é necessário um grande número de pontos e, consequentemente, o custo computacional será muito alto.

Para aperfeiçoar a distribuição e permitir que regiões críticas tenham densidade diferenciada, precisamos considerar a identificação destas regiões. Para os casos onde o interesse do problema é maior nas bordas dos objetos ou em fronteiras, podemos postergar a solução para a fase de refinamento, tratada com detalhes na Seção 3.3. No entanto, para casos onde não é possível inferir a localização do ponto crítico, precisamos que esta informação seja fornecida ainda durante a fase de entrada das coordenadas dos objetos.



(a) (b)

Figura 3.8: Identificação das áreas de interesse do polígono: (a) a partir de um ponto, (b) a partir de um segmento.

É o caso exemplificado na Figura 3.8, onde em 3.8a é identificada a área de interesse usando um ponto e em 3.8b usando um segmento. Estes elementos podem estar localizados em qualquer lugar no espaço bidimensional, seja sobre os segmentos ou vértices, dentro, fora ou mesmo cortando os limites dos polígonos. Eles recebem tratamento diferenciado em cada etapa da geração da nuvem, sendo que:

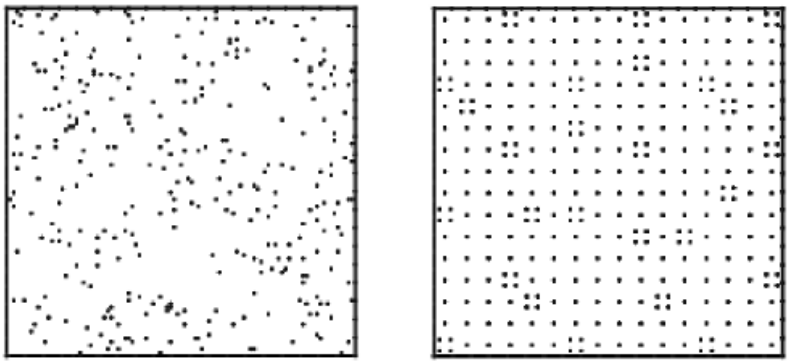
- Na tarefa de inferir os polígonos discutida na Seção 3.1.1, eles são desconsiderados e não interferem na geometria do objeto;
- Na tarefa de geração da nuvem inicial, eles são devidamente representados recebendo um ponto em sua coordenada, quando o elemento que identifica a área de interesse for um ponto, ou vários pontos ao longo de seu comprimento, conforme sua densidade configurada, quando se tratar de um segmento de reta. Em ambos os casos, a distribuição de pontos sobre eles só acontece quando são internos ao objeto, total ou parcialmente (neste último caso, somente a parte que é interna ao objeto recebe pontos);
- Na tarefa de refinamento da nuvem de pontos, tratada na Seção 3.3, eles interferem no cálculo dos pesos dos pontos e, consequentemente, no tamanho da área de representatividade de cada um deles. Quando mais próximo do elemento identificador (ponto ou

segmento), mais conforme a densidade dele a nuvem será e menos conforme a densidade configurada para o objeto modelado.

3.2 Geração de Nuvem de Pontos

Uma vez definidos os polígonos que representam os objetos do problema tratado, podemos então distribuir pontos ao longo de sua superfície e limites. Nosso objetivo neste momento é descobrir a quantidade de pontos necessários e distribuí-los de forma a se aproximar da densidade necessária para resolver o problema tratado sem, necessariamente, garantir a qualidade desta distribuição.

Neste trabalho implementamos duas abordagens distintas para a geração inicial de pontos e, assim, avaliar as contribuições de cada uma separadamente e comparar os resultados. A primeira delas é baseada em um algoritmo de geração aleatória de coordenadas dentro do domínio, gerando uma distribuição como a que é ilustrada na Figura 3.9a. A segunda é baseada em um algoritmo que segue os princípios de um refinamento adaptativo, gerando recursivamente malhas cada vez mais refinadas com disposição dos pontos em forma de grade, cujo resultado pode ser visto na Figura 3.9b.



(a) (b)

Figura 3.9: Resultado da geração aleatória de pontos (a) e baseada em refinamento adaptativo (b).

3.2.1 Definição do número de pontos gerados

Antes de se iniciar a criação dos pontos é necessário calcular quantos pontos serão gerados. Este cálculo é aproximado e a quantidade final dependerá também do processo de refinamento, que pode inserir ou retirar pontos durante suas iterações. É desejado, no entanto, o valor mais aproximado possível, pois cada vez que um grupo de pontos é inserido ou retirado na etapa de refinamento, todo ele precisa ser refeito.

O número inferido de pontos, n, é resultado da multiplicação da área do polígono P pela densidade d desejada:

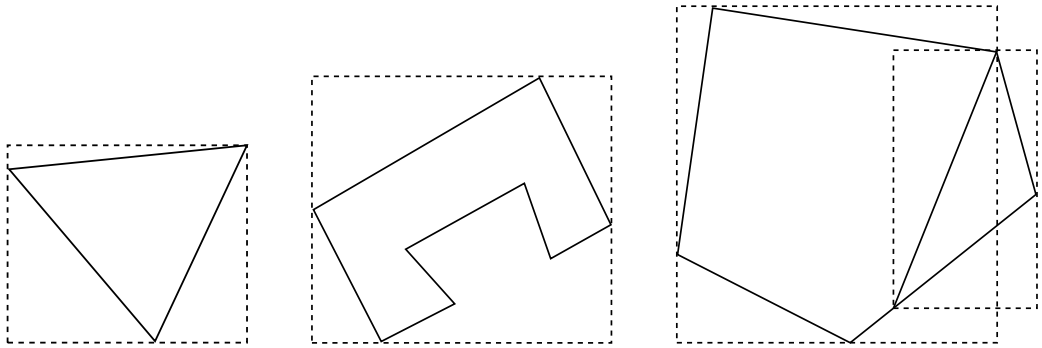
$$n = area(P) \times d \tag{3.1}$$

Desta forma, garantimos que a densidade configurada pelo usuário seja respeitada independente do tamanho do polígono, e que a distância média entre os pontos distribuídos seja diretamente proporcional à densidade pedida, sem sofrer interferência do formato ou disposição do objeto.

3.2.2 Geração de pontos com coordenadas aleatórias

A geração aleatória, uma vez conhecido o número de pontos que devem ser gerados, acontece para cada polígono do problema de forma independente (Figura 3.10), mesmo que na etapa de desenho ele tenha sido criado compartilhando alguma(s) aresta(s) com outro(s) qualquer, como nos polígonos da Figura 3.10c.

Iterativamente e para cada polígono, a geração de pontos acontecerá seguindo duas tarefas básicas descritas abaixo, até se atingir o objetivo estipulado pelo cálculo de pontos necessários para a solução do problema na densidade configurada.



(a) (b) (c)

Figura 3.10: Definição da área limite de dois exemplos de polígono (a) e (b) e para polígonos compartilhando aresta de fronteira (c) para o processo de criação aleatória de pontos.

1. Definimos um retângulo envolvente mínimo que contém o polígono em seu interior, como visto tracejado na figura 3.10. Todas as coordenadas dos pontos gerados, apesar de usar um algoritmo de geração de números aleatórios, estarão dentro dos limites deste retângulo;

2. Até que todos os pontos esperados sejam gerados, calculam-se coordenadas aleatórias para x e y de forma independente, e as submetemos a avaliação que verifica se este é um ponto válido ou não. Percebemos que, mesmo dentro do retângulo, existem áreas que não fazem parte do polígono, e que eventualmente irão receber pontos gerados. Para resolver este problema, precisamos verificar, a cada ponto gerado, se ele está interno ao polígono. Caso esta regra não seja cumprida, este ponto será descartado e não será deduzido no número de pontos total a ser gerado. Nesta etapa, pontos que estão sobre os segmentos também são descartados, uma vez que pontos sobre as arestas do polígono serão gerados em uma etapa exclusiva para isso.

3.2.3 Geração de pontos baseada na estrutura de Grafos de Folhas Autônomas (ALG)

Percebemos que, usando o método de geração aleatória, a variação de distâncias médias entre pontos é grande. Observando a disposição dos pontos ilustrada na figura 3.9b, percebemos de forma evidente a existência ora de clarões, ora de agrupamentos muito mais densos do que

o desejado ao longo da superfície. Partimos então para uma solução que permite a criação de pontos de forma organizada, até a quantidade desejada de pontos.

A implementação do trabalho Burgarelli [3] utiliza um refinamento adaptativo para obter malhas adequadas à representação de equações diferenciais parciais discretizadas. A solução é, basicamente, a construção automática de uma malha pouco densa em regiões de menor interesse e mais densa onde houver maior interesse, mantendo tanto quanto possível uma transição suave. Assim construindo, pode-se reduzir drasticamente o número de pontos necessários para obter uma solução precisa para problemas suaves, e atingir uma ordem de convergência ótima para problemas não suaves. Essencialmente, trata-se da subdivisão recursiva do espaço utilizando a estrutura *Autonomous Leaves Graph* (ALG) [2]. Essa estrutura é uma evolução da bem conhecida estrutura *quadtree* para uso em problemas de refinamento adaptativo de malhas.

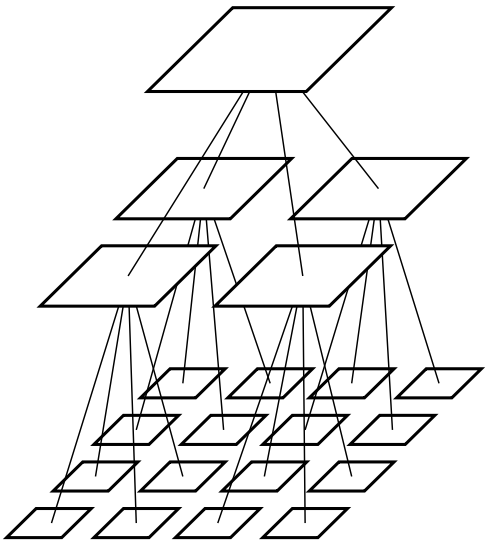


Figura 3.11: Subdivisão recursiva da *quadtree*.

Uma árvore de refinamentos, comumente chamada de *quadtree*, é uma estrutura frequentemente utilizada para construir sub-malhas mais refinadas a partir de malhas menos refinadas, gerando uma hierarquia que pode ser vista na Figura 3.11. A partir de uma malha inicial, chamada de raiz, são criados filhos como sub-malhas de grau de refinamento maior e área da célula quatro vezes menor. Esta subdivisão acontece sucessivamente até se alcançar a granularidade desejada.

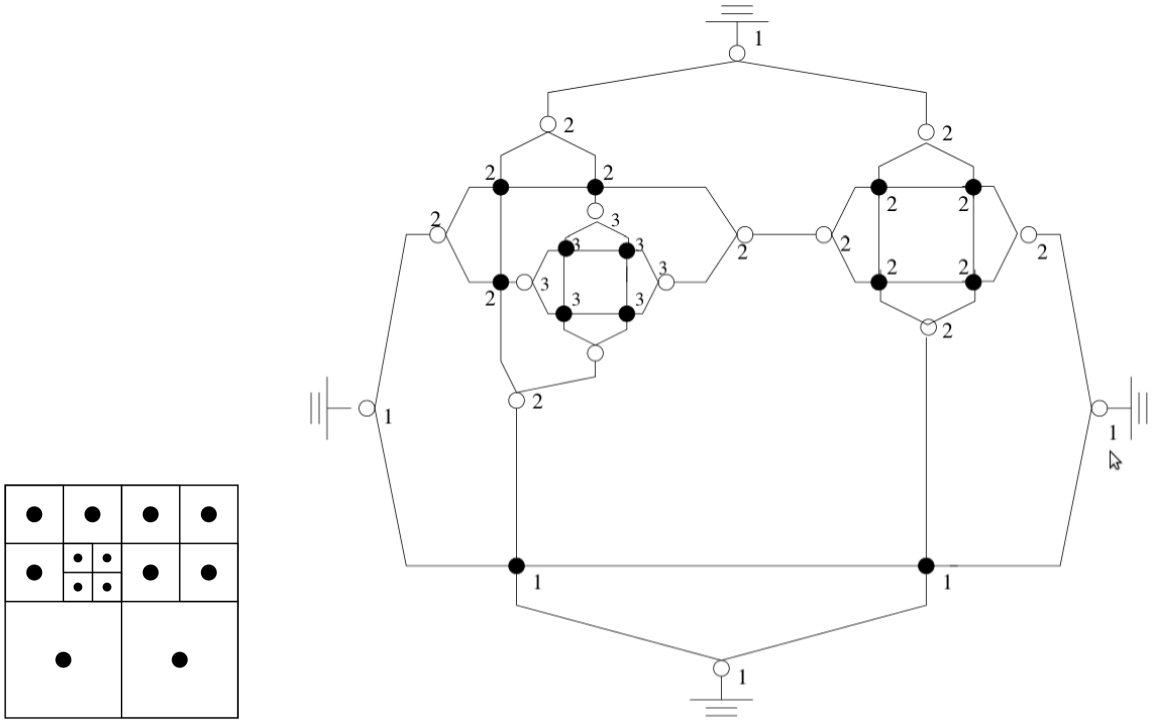
Já a ALG é uma estrutura de dados baseada em grafos proposta por Burgarelli et al. [2], onde o custo de comunicação entre as células de um domínio discretizado é $O(1)$, independente do número de células presentes na discretização. Para isso, os nós pais são substituídos por nós filhos que, devidamente identificados seus respectivos níveis hierárquicos, convivem na mesma estrutura dos nós que não precisaram ser refinados. No entanto, a qualquer momento é possível o processo inverso ao refinamento, apesar deste procedimento não ser necessário para o problema tratado neste trabalho.

A Figura 3.12 mostra um exemplo de uma malha adaptativa e respectiva representação gráfica da estrutura de dados (ALG). Observe que para cada ponto, centroide de uma célula em 3.12a, existe um nó mapeado na estrutura 3.12b como uma bolinha preta. Cada nó possui quatro ponteiros que apontam para os seus nós vizinhos (acima, abaixo, à direita e à esquerda), ou para $null$, caso seja um nó que limita o objeto. Quando o nível entre dois nós é igual, o apontamento é direto (um nó aponta para o outro), mas quando o nível é maior ou menor é necessário um elemento especial intermediando o apontamento, representado por uma bolinha branca para cada nível de diferença. Assim, independente de quantos níveis de diferença existam entre dois nós, a representação do nó na estrutura de dados será a mesma.

Por convenção, partimos do espaço quadrangular dividido em quatro células iguais, cujo nível hierárquico é classificado como 1. Uma célula de nível n, após refinada, será substituída por quatro células de nível $n+1$. Tanto na proposta original apresentada por Burgarelli [3] quando neste trabalho, vinculamos o nível da célula ao seu tamanho: uma célula de nível $n+1$ será $\frac{1}{4}$ do tamanho da célula de nível n.

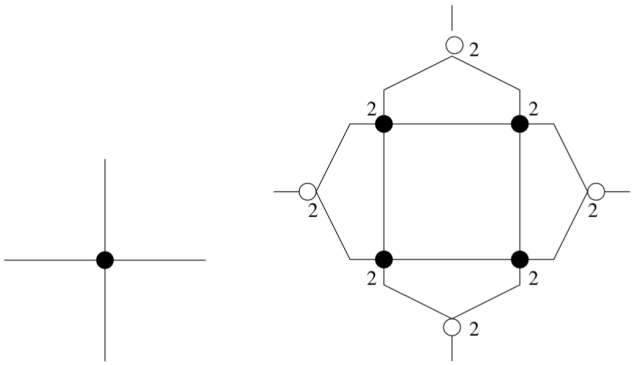
Cada nó refinado, como por exemplo, o nó exemplo da Figura 3.13a, é substituído por quatro vértices, como em 3.13b. O número 2 ao lado de cada vértice indica seu nível de refinamento. Repare que nas Figuras 3.12a e 3.13b, que são representações gráficas da estrutura de dados, existem vértices brancos e pretos: os pretos são de fato os nós criados. Já os brancos são vértices especiais que não representam nós da malha, mas servem de conexão entre nós pretos de diferentes níveis, onde um nó preto "pai" precisa se conectar a dois "filhos". São, portanto, imprescindíveis para o processo de busca de vizinhos, tanto do processo de refinamento, quanto de desrefinamento, se necessário à aplicação.

A iteração na lista de nós, nesta abordagem, é garantida pela atualização a cada etapa de refinamento ou desrefinamento de uma lista duplamente encadeada gerada a partir de uma



(a) (b)

Figura 3.12: Exemplo de malha adaptativa (a) e a representação gráfica da estrutura de dados (b). Figuras retiradas de [3].



(a) (b)

Figura 3.13: Processo de substituição de um nó por quatro outros de nível de refinamento maior em sua estrutura de dados: (a) estado inicial, (b) após a substituição. Figura retirada de [3].

adaptação do algoritmo de contrução da curva de *Hilbert* (versão original descrita na Seção 2.4.5), permitindo percorrer todos os nós da nuvem mesmo que sejam de níveis diferentes (Figura 3.14).

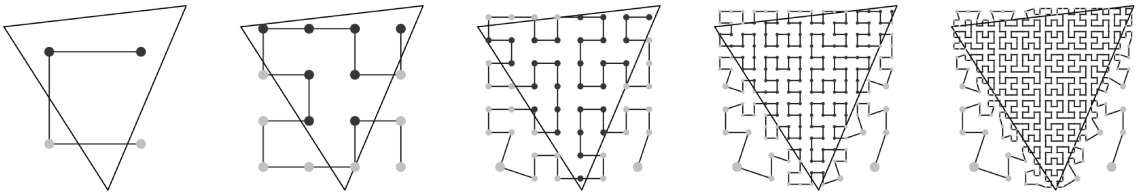


Figura 3.14: Adaptação da curva de *Hilbert* em cinco níveis de refinamento.

Neste trabalho em particular, adaptamos os algoritmos de refinamento da malha adaptativa para, seguindo o mesmo propósito original, refinar recursivamente a malha somente na região de interesse, que no nosso problema é o interior do polígono. Posteriormente, transformar a malha em uma nuvem de pontos cujo resultado pode ser visto na Figura 3.15. Observe que todo o espaço ocupado pelo quadrado imaginário que contém a figura é preenchido por pontos, mas somente refinamos a medida que se faz necessário para se obter o nível de refinamento desejado no interior do objeto. Ainda, como nosso interesse é somente os pontos internos ao polígono, todo aquele cujas coordenadas estão fora dos limites será desconsiderado na nuvem resultante.

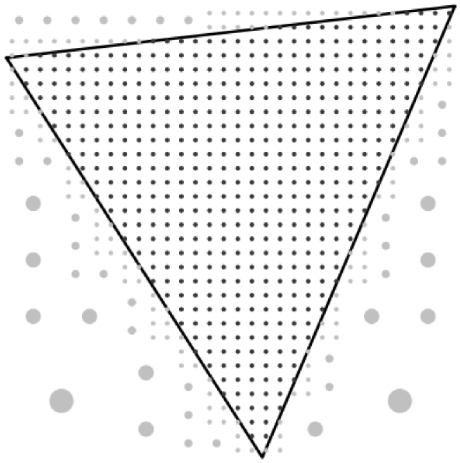
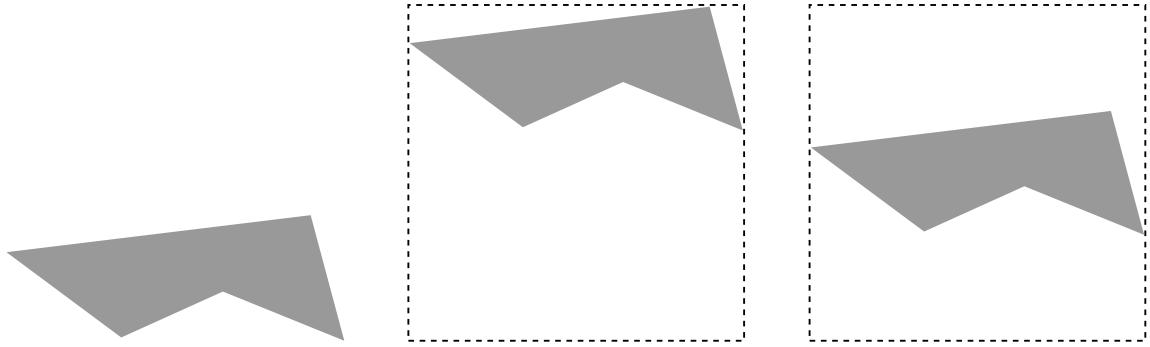


Figura 3.15: Nuvem de pontos extraída da malha adaptativa.

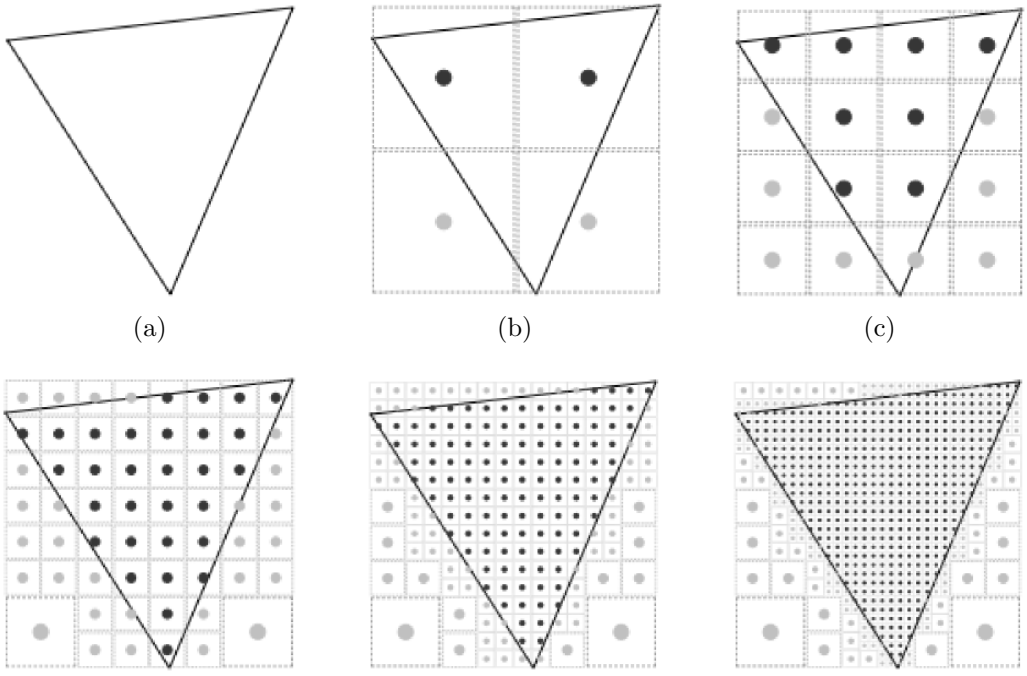
Para definir o espaço do quadrado raiz, a aplicação calcula o menor quadrado envolvente possível que comporta o polígono. Por se tratar de quadrado, é bem provável que, ou na vertical ou na horizontal, ele exceda o polígono e, neste caso, o quadrado é deslocado de forma a deixar o polígono exatamente no meio (Figura 3.16).

Como ilustrado na Figura 3.17, o quadrado é dividido em quatro partes (3.17b), igualmente quadradas, que chamamos de células. Cada célula pode ser traduzida em um ponto cujas



(a) (b) (c)

Figura 3.16: Criação do quadrado envolvente (b) de um polígono (a). Para aperfeiçoar as subdivisões do quadrado, ele foi deslocado verticalmente (c), neste exemplo, para centralizar o polígono.



(a) (b) (c)

(d) (e) (f)

Figura 3.17: Exemplo de iterações de subdivisão recursiva (c), (d), (e), (f) das quatro células (b), do polígono (a).

coordenadas são as do seu centro. Quando este ponto fica localizado fora do polígono, ele não fará parte da nuvem de pontos gerada, mas a célula que ele representa ainda pode ser subdividida, caso ela tenha intercessão com algum segmento do polígono. Percebemos esta necessidade com mais detalhes na Figura 3.18.

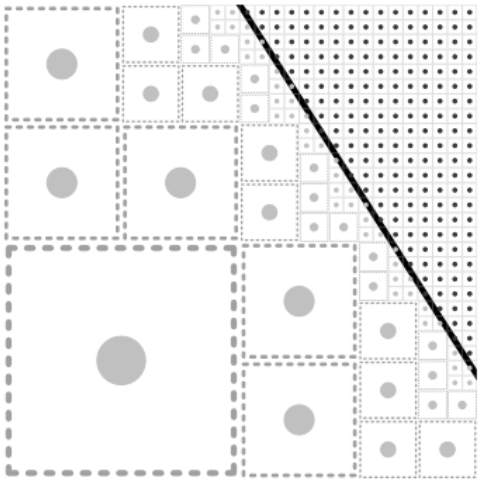


Figura 3.18: Detalhe da variação de gradual e sobre demanda das células à medida que se aproxima do polígono.

Há ainda um caso especial no processo de refinamento onde nem todos os nós são refinados, mesmo estando internos ao polígono. Para explicar esta particularidade, vamos partir de um exemplo onde o polígono é exatamente um quadrado, como visto na Figura 3.19. O processo de subdivisão é recursivo e, por isso, aumenta o número de pontos sempre quatro vezes de uma iteração para outra. Isto gera um problema: supondo que para atingir a densidade pedida seja necessária a distribuição de aproximadamente 300 pontos ao longo da superfície do polígono. Sabendo-se que a subdivisão escolhida divide cada célula em outras 4 células, temos como resultado em cada iteração a seguinte quantidade de pontos:

$$4 \to 16 \to 64 \to 256 \to 1024 \to \cdots \tag{3.2}$$

Ou seja, na quarta iteração ainda não temos a quantidade desejada, mas a quinta insere muito mais pontos do que o necessário. Se parássemos na quarta iteração ou realizássemos a quinta, provocaríamos um alto custo a ser pago na fase de refinamento, fazendo com que qualquer benefício desta distribuição de pontos inicial criteriosa seja ínfimo perante seu malefício.

Para resolver este problema, quando atingimos a iteração cuja quantidade de pontos é menor do que a desejada e que a próxima seja maior, interrompemos a subdivisão completa das células e passamos a fazer de forma parcial e criteriosa, somente em uma quantidade de

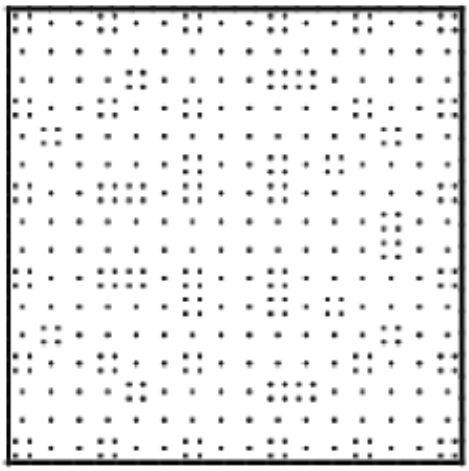


Figura 3.19: Resultado da geração de pontos usando ALG com iteração executada parcialmente.

células mínimas necessárias para se atingir o que falta do número de pontos desejados. As células escolhidas, cujo grau será um nível acima das demais, são distribuídas no polígono para que não provoque uma densidade maior indesejada em uma determinada região. A iteração parcial procede da seguinte forma:

1. Inferimos o número de pontos faltantes, rp, subtraindo o número de pontos esperados do número de pontos resultante da iteração atual;

2. Calculamos o número de células rc que precisam ser refinadas para somar rp ao número total de pontos. Sabemos que cada célula dará lugar a quatro novas, ou, em outras palavras, cada célula refinada (que já contribui com 1 ponto) somará 3 no número total de pontos. Em sumo, $rc = rp/3$;

3. Calculamos o passo (δ) usado nesta iteração, ou seja, calculamos de quantas em quantas células uma será escolhida para ser refinada. Este cálculo é feito dividindo-se o número de células do nível atual, tc, pelo número de células cujo refinamento se faz necessário. Ou seja, $\delta = tc/rc$;

4. Procedemos com o refinamento parcial, percorrendo todas as células do nível atual escolhendo para ser refinada uma a cada δ células.

O resultado do processo de geração de pontos onde se fez necessário uma iteração parcial das células pode ser visto na Figura 3.19.

3.2.4 Distribuição de pontos sobre as arestas dos polígonos

Em ambos os métodos de geração de pontos, tratamos apenas a distribuição no interior dos polígonos, além dos pontos nos vértices aproveitados da fase de desenho. No entanto, ainda precisamos distribuir pontos ao longo das arestas, que no nosso problema representam o contorno do problema ou, em casos onde objetos compartilham arestas, interfaces entre diferentes materiais.

Para esta distribuição de pontos tratamos isoladamente cada segmento: de posse dos pontos inicial e final, calculamos a distância entre eles e dividimos pela densidade escolhida. Assim, criamos a quantidade exata necessária de pontos e distribuímos de forma equidistante ao longo da aresta. Estes novos pontos estão sujeitos a alteração durante o refinamento assim como qualquer outro localizado no interior do polígono.

Percebemos, então, a existência de três tipos de pontos em um mesmo polígono:

- Pontos de vértices;
- Pontos de arestas;
- Pontos internos.

A representação computacional dos pontos possui informação sobre seu tipo, importante tanto para o processo de refinamento, que trata cada um destes grupos de forma diferente, quanto para a geração do arquivo de saída da aplicação, permitindo que a aplicação cliente trate diferente cada tipo de ponto.

3.3 Refinamento

O processo de refinamento é o que de fato garantirá que seja respeitada a densidade exigida para solucionar o problema e também a qualidade da distribuição de pontos. Basicamente, existem dois princípios para a distribuição de pontos: o princípio da equi-distribuição, uniformidade ou distribuição homogênea e o princípio variacional ou distribuição heterogênea. No primeiro, intuitivamente entendemos que se trata de espalhar pontos da nuvem ao longo da superfície, mantendo tanto quanto possível a mesma distância entre os pontos. Já o segundo propõe a utilização de um número mínimo de nós, diminuindo o custo computacional durante o tratamento da nuvem e, ainda assim, obter a precisão desejada. Isto é conseguido através da distribuição mais densa de pontos nas áreas onde a aproximação tende a ter variações abruptas, e menos pontos em áreas com variações mais suaves.

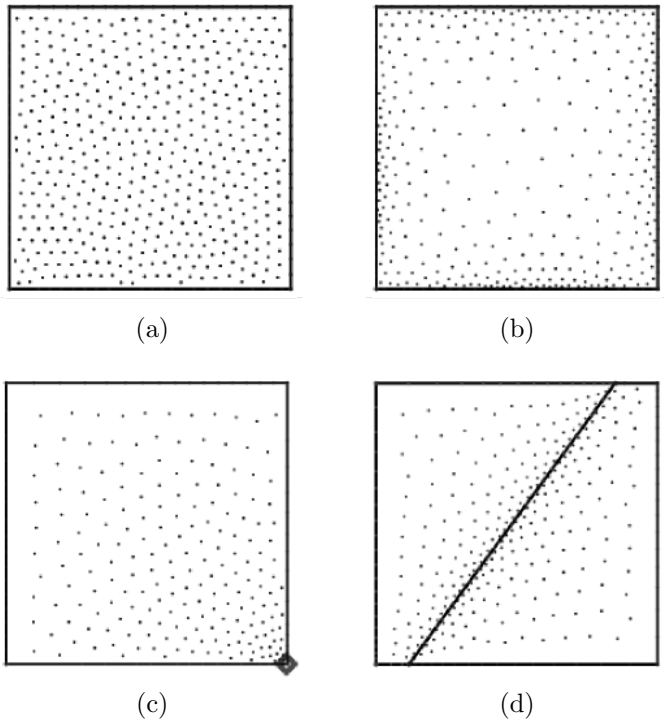


Figura 3.20: Resultado do refinamento segundo as quatro configurações tratadas: (a) homogêneo, (b) heterogêneo com ênfase nas bordas, (c) heterogêneo com ênfase nas regiões de interesse, (d) heterogêneo com ênfase nas bordas compartilhadas.

A qualidade de uma distribuição está relacionada à uniformidade das distâncias entre pontos, seja para um problema de distribuição homogênea ou heterogênea. É desejado que os pontos sejam dispostos equidistantes entre si na distribuição homogênea (como ilustrado na Figura

3.20a), ou seguindo uma graduação uniforme quando for necessário maior detalhamento em uma área específica do polígono (como ilustrado na Figura 3.20b, 3.20c e 3.20d). O processo de refinamento independe do método usado para a geração de pontos, mas o custo computacional de sua execução pode variar, uma vez que a distribuição segundo um método pode garantir uma qualidade menos distante da ideal do que em outro, exigindo menos iterações de refinamento. Analisaremos as diferenças percebidas no Capítulo 4, que discute os resultados alcançados.

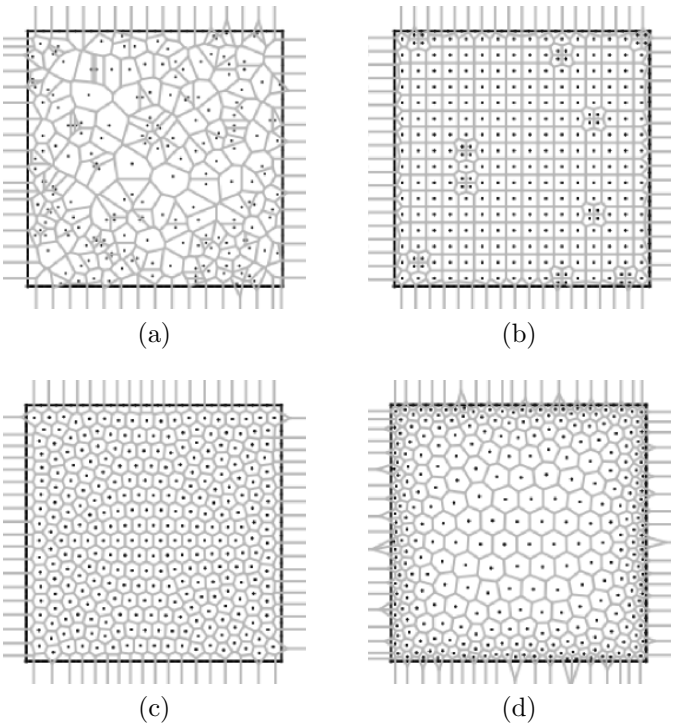


Figura 3.21: Distribuição de pontos com respectivas células *Voronoi*: (a) após distribuição inicial aleatória, (b) após distribuição inicial baseada no ALG, (c) após refinamento homogêneo, (d) após refinamento heterogêneo com ênfase nas bordas.

O processo de refinamento proposto neste trabalho é iterativo e utiliza o algoritmo de *Lloyd* no reposicionamento dos pontos para uma posição melhor a cada iteração. Como descrito na Seção 2.4.4, o algoritmo basicamente gera o diagrama de *Voronoi* usando os pontos da nuvem atual e os reposiciona para a posição do centro de massa de sua célula. Na Figura 3.21a e 3.21b podemos ver como fica o diagrama de *Voronoi* inicial para os dois métodos de geração de pontos apresentados nas Seções 3.2.3 e 3.2.2, e em 3.21c e 3.21d, o diagrama final, segundo a distribuição homogênea e heterogênea com ênfase nas bordas respectivamente.

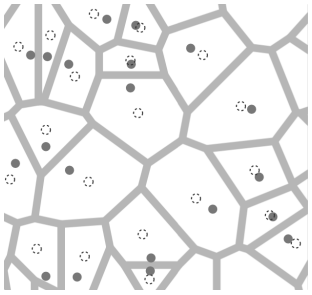


Figura 3.22: Exemplo de ajuste de coordenadas dos pontos da nuvem de uma iteração. Nas células de *Voronoi*, os pontos geradores são representados preenchidos, enquanto que o centro de massa, que possui as coordenadas que serão aplicados nos geradores, com borda tracejada.

Na Figura 3.22 podemos ver o detalhe do deslocamento a ser realizado nas iterações iniciais (deslocamentos mais significativos): um conjunto de pontos geradores (ponto preenchido), o diagrama de *Voronoi* (formação em células) e centro de massa de cada célula (ponto de borda tracejada).

Quando um ponto do diagrama de *Voronoi* permite atribuição de peso, temos o chamado *Power Diagrams* [8]. Neste diagrama, base para a distribuição heterogênea, os pontos são representados por círculos cujo tamanho é proporcional ao peso atribuído, comumente calculado pela distância euclidiana. Neste trabalho especificamente, a distância em questão é a do centro de massa da célula de *Voronoi*, x_1, até a área de interesse, x_2 que, dependendo do problema e das configurações para o processamento da solução, são todas as arestas do polígono ao qual x_1 pertence (Figura 3.20b), um segmento ou ponto extra (Figura 3.20c), ou uma aresta compartilhada (Figura 3.20d). Dados os pontos

$$\mathrm{x}_1 = (x_1, y_1) \tag{3.3}$$

$$\mathrm{x}_2 = (x_2, y_2) \tag{3.4}$$

a distância euclidiana entre eles é calculada fazendo

$$|\mathrm{x}_1 - \mathrm{x}_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.5}$$

e o peso atribuído ao ponto gerador do diagrama de *Voronoi*, $\mathrm{C_i}$, é definido como

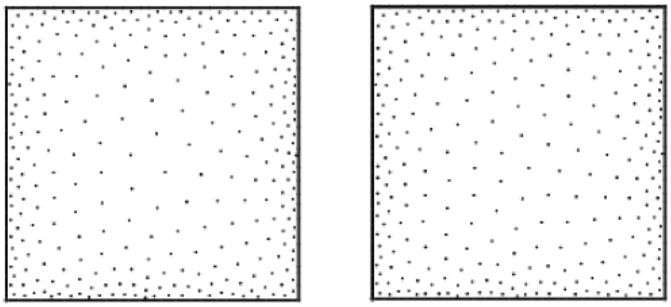
$$\mathrm{C_i} = \frac{\int\limits_A \mathrm{x}\rho(\mathrm{x})dA}{\int\limits_A \rho(\mathrm{x})dA} \tag{3.6}$$

onde A é a região, x é a posição e $\rho(\mathrm{x})$ é a função densidade. Para uma região de densidade constante ρ, temos o que neste trabalho chamamos de "refinamento homogêneo", uma vez que as células do diagrama tendem a ter uma área igual (Figura 3.21c). Já um exemplo de ρ variável pode ser visto na Figura 3.21d, quando usamos a distância da borda para calcular seu valor. Neste caso, o peso é proporcional à distância da borda, ou seja, quanto mais distante (ou mais ao centro do polígono), maior é o valor de ρ e consequentemente, maior o tamanho das células *Voronoi* e maior o espançamento entre pontos.

Vincular o valor de ρ ao peso de um ponto gera uma variação de tamanho das células desejado. Mas eventualmente pode gerar uma representatividade insuficiente nas regiões onde os pontos receberam pesos muito altos. Para tratar este problema e garantir um mínimo de representatividade, incluímos a possibilidade de limitar o peso máximo inferido para cada configuração de densidade. O efeito deste limite pode ser visto na Figura 3.23: em 3.23a, as células centrais possuem um tamanho que pode não ser suficiente para representar a área mais ao centro do polígono; em 3.23b garantimos que o peso atribuído, apesar de maior do que qualquer outra célula com valor de ρ original menor, seja igual ou menor que um valor mínimo estipulado.

3.4 Reavaliação da Densidade

Após uma quantidade de iterações do algoritmo de *Lloyd*, mas ainda distante da solução final, reavaliamos a densidade prévia da nuvem resultante. Esta ação é importante para garantir



(a) (b)

Figura 3.23: Resultado do refinamento heterogêneo com ênfase nas bordas sem limitar peso mínimo para os pontos 3.23a e limitando o peso mínimo 3.23b.

que a nuvem resultante tenha a densidade necessária para a solução do problema, o que talvez seja a mais importante característica de uma nuvem de pontos a ser usada em um método sem malha. Se a densidade for maior ou menor que a desejada, considerando um limite de erro, pontos são excluídos ou criados.

Para calcular o número de pontos necessários para realizar o ajuste de densidade, usamos uma heurística baseada em regra de três simples: a média das distâncias entre pontos atual, ma, está para o número de pontos atual da nuvem, na, assim como a média desejada (extraída da densidade desejada), md, está para o número de pontos que estamos calculando, nd (Equação 3.7).

$$nd = \frac{na \times md}{ma} \tag{3.7}$$

A diferença entre o número de pontos atual e nd é a quantidade que deve ser inserida ou excluída, n:

$$n = na - nd \tag{3.8}$$

Sabendo-se o valor de n, procedemos com o devido ajuste: caso seja para incluir, n pontos são distribuídos equidistantemente ao longo da superfície. Caso seja para excluir, n pontos são

escolhidos segundo suas coordenadas, de forma que sejam excluídos pontos mais equidistantes quanto possível ao longo da superfície. Após esta ação, podemos imaginar que o refinamento realizado até então seja comprometido e, por este motivo, o processo é reiniciado e executado até que a qualidade esperada seja alcançada ou uma nova alteração na quantidade de pontos seja necessária.

3.5 Particularidades da Implementação

Normalmente, as células de *Voronoi* originadas dos pontos localizados nas arestas do polígono seriam células ilimitadas e, consequentemente, a área desta célula seria infinita e seu centro de massa impossível de ser calculado, como ilustrado na Figura 3.24.

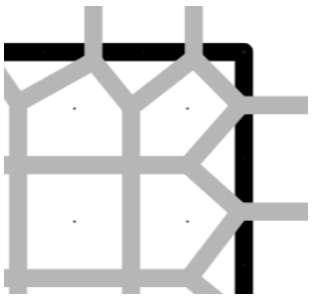


Figura 3.24: Células sem limites definidos, cujos pontos geradores estão localizados nas arestas do polígono.

Para resolver este problema, distribuímos quatro pontos imaginários no exterior do polígono, como pode ser visto na Figura 3.25. Estes pontos são tratados como integrantes da nuvem de pontos geradores, sendo considerados também na geração do diagrama de *Voronoi*. No entanto, no momento em que a nuvem é exportada para uso em um método sem malha, estes pontos são descartados.

Com a presença destes quatro pontos garantimos que todas as células que realmente fazem parte da nuvem sejam células fechadas, o que facilita o tratamento usando os algoritmos geométricos adotados e discutidos no Capítulo 2. As quatro células que ficaram com limites indefinidos não são de nosso interesse e não nos trarão nenhum problema.

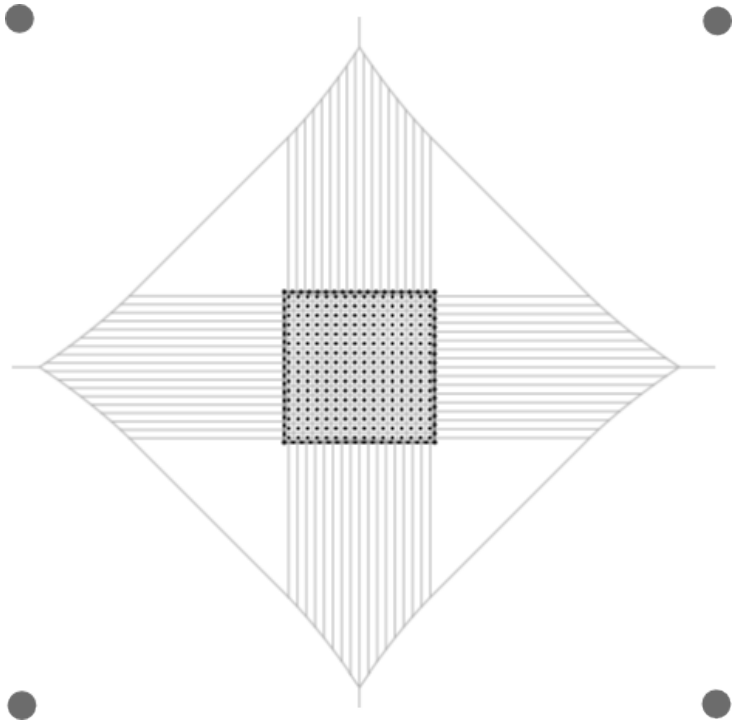


Figura 3.25: Células de *Voronoi* após a inserção dos quatro pontos imaginários no nordeste, sudeste, sudoeste e noroeste do polígono.

Apesar de agora fechadas, as células localizadas nas extremidades do polígono ainda não podem ser trabalhadas como as demais. Analisando um pouco mais a fundo, percebemos que precisamos resolver outro problema: estas células extrapolam os limites do polígono. Se não tratássemos estes casos de forma diferenciada, seus centros de massa teriam coordenadas externas ao polígono, o que faria com que os pontos geradores fossem deslocados para fora dele, descaracterizando a nuvem de pontos.

Para resolver este problema, substituímos a célula original pela geometria formada pela intercessão entre ela e o polígono. Garantidamente, a nova geometria estará contida dentro dos limites e, consequentemente, as coordenadas do seu centro de massa também. Todo este tratamento especial precisa acontecer somente nos pontos localizados nas arestas e vértices. Saber se um ponto é interno, se é vértice ou está sobre aresta garante que este cálculo não seja feito desnecessariamente.

Para o caso particular de pontos sobre arestas, temos ainda uma particularidade a tratar. Como descrito na Seção 3.3, sabemos que os pontos geradores serão deslocados para as coordenadas dos centros de massa de suas respectivas células, o que não pode acontecer com pontos do

tipo "vértice" ou "aresta". Caso contrário, eles seriam deslocados para o interior do polígono e deixariam de representar as extremidades.

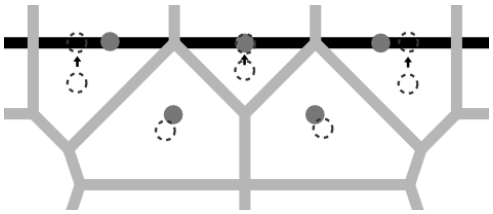


Figura 3.26: Deslocamento especial na iteração do algoritmo de *Lloyd* dos pontos localizados nas arestas ou vértices.

Para resolver este problema, tratamos os deslocamentos destes pontos de forma diferente da dos demais. Realizamos o cálculo do centro de massa da mesma forma, mas antes de usar as coordenadas, realizamos as seguintes tarefas, ilustradas na Figura 3.26:

1. Identificamos o fragmento da aresta do polígono que corta a célula;
2. Identificamos a coordenada sobre esta aresta que está sobre o ângulo reto formado por ele e a linha imaginária que seria a menor distância até o ponto original;
3. Usamos estas coordenadas para o deslocamento do ponto original da célula tratada.

Desta forma, garantimos que, mesmo após várias iterações com consequentes alterações de coordenadas dos pontos, teremos pontos sobre os limites do polígono.

Capítulo 4

Resultados

Abordaremos neste Capítulo os resultados obtidos, partindo de exemplos práticos de geração de nuvem de pontos até a análise de desempenho e impressões a respeito da solução de forma geral. Mostraremos também o comportamento da solução quando na solução de um problema eletrostático.

Nosso intuito não é abordar problemas de métodos numéricos, mas precisamos avaliar se a solução de geração de nuvem de pontos proposta é viável para a discretização dos domínios tratados. A qualidade da distribuição afeta diretamente a precisão numérica da solução de um problema real.

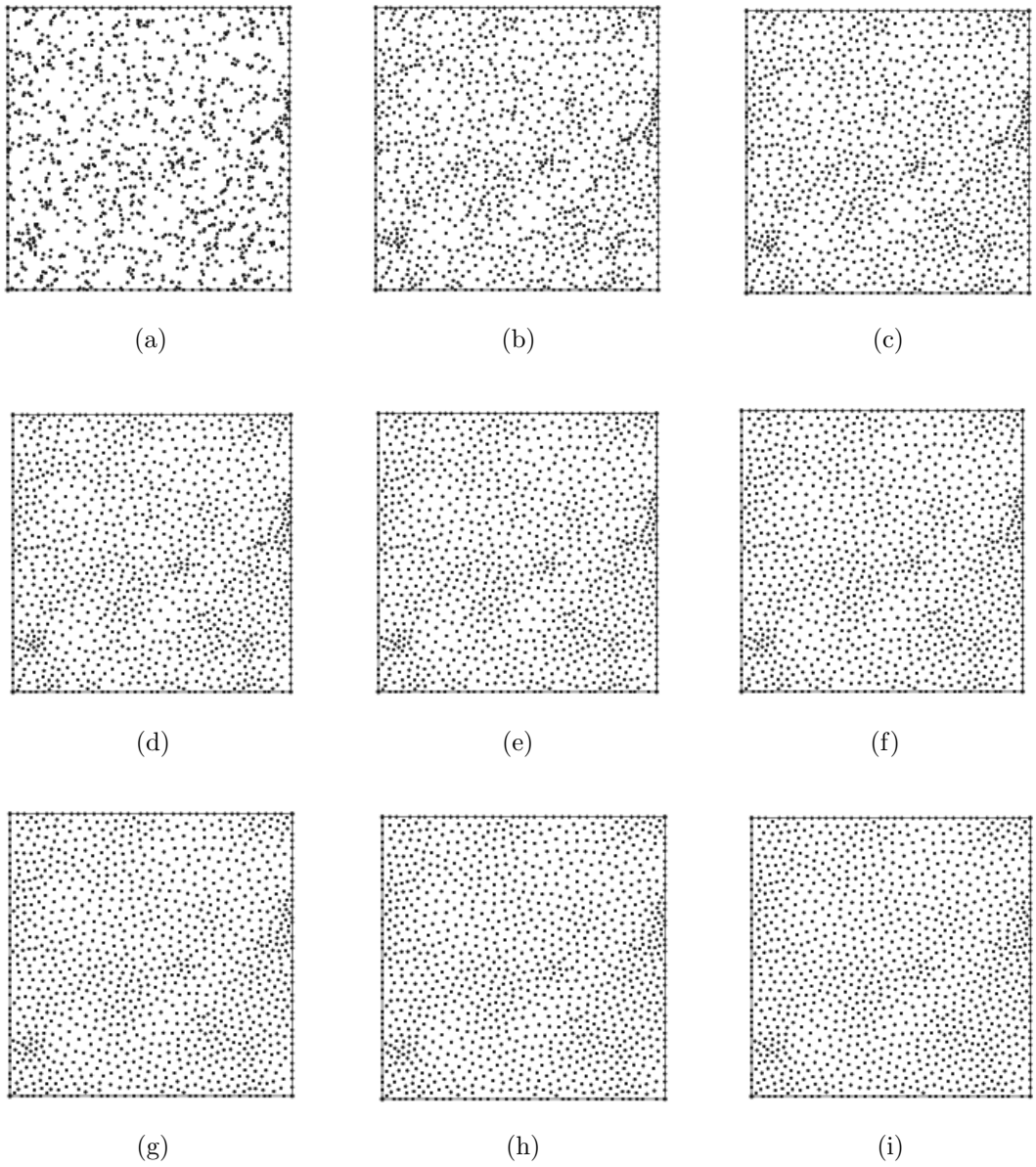
4.1 Avaliação do Impacto da Escolha do Método no Resultado da Nuvem

Como já discutido em detalhes no Capítulo 3 deste trabalho, implementamos dois métodos distintos para a realização da tarefa de geração da nuvem inicial de pontos. Nosso objetivo, com isto, é comparar o desempenho de ambos, principalmente quanto ao custo de execução e o impacto na qualidade da nuvem resultante.

Abordaremos isoladamente as tarefas de geração e refinamento da nuvem de pontos. Faremos um comparativo entre os dois métodos desde a geração até o resultado final do refinamento. Trataremos o problema segundo três abordagens: o tempo para execução da tarefa de geração, o tempo para o refinamento e o número de iterações do algoritmo de *Lloyd* que foram necessários para se atingir a qualidade estipulada.

Para entender o que cada abordagem significou, vamos analisar as Figuras 4.1, 4.2, 4.3 e 4.4. Isolamos as primeiras iterações do algoritmo de refinamento, *Lloyd*, quando aplicado sobre uma nuvem gerada segundo a abordagem aleatória (Figuras 4.1 e 4.2) e a abordagem baseada no ALG (Figuras 4.3 e 4.4).

Notamos de forma evidente que o refinamento quando a nuvem foi criada de forma organizada, baseada no refinamento adaptativo (ALG), é muito mais suave. Basicamente, quando a densidade já é acertada, só são necessários ajustes das bordas. No entanto, isso não quer dizer que o custo computacional seja menor, uma vez que o processo de análise de cada célula *Voronoi* será realizado de uma forma ou de outra. No entanto, é notória a distribuição final de pontos mais organizada.

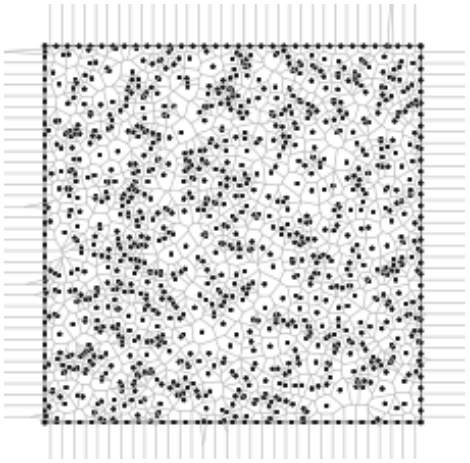


(a) (b) (c)

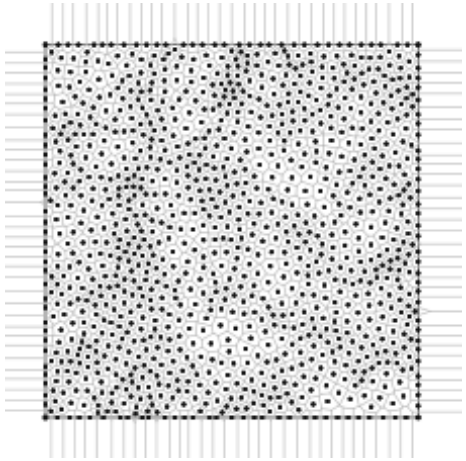
(d) (e) (f)

(g) (h) (i)

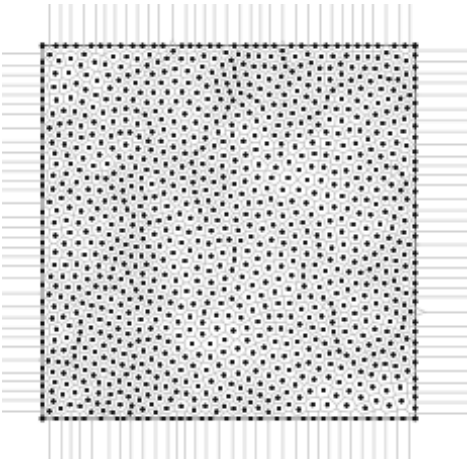
Figura 4.1: Iterações *Lloyd* passo a passo a partir de uma nuvem de pontos gerados aleatoriamente.



(a)



(b)



(c)

Figura 4.2: Iterações *Lloyd* passo a passo a partir de uma nuvem de pontos gerados aleatoriamente, mostrando as células *Voronoi*. (a) refere-se à Figura 4.1a, (b) refere-se à Figura 4.1e e (c) refere-se à Figura 4.1i.

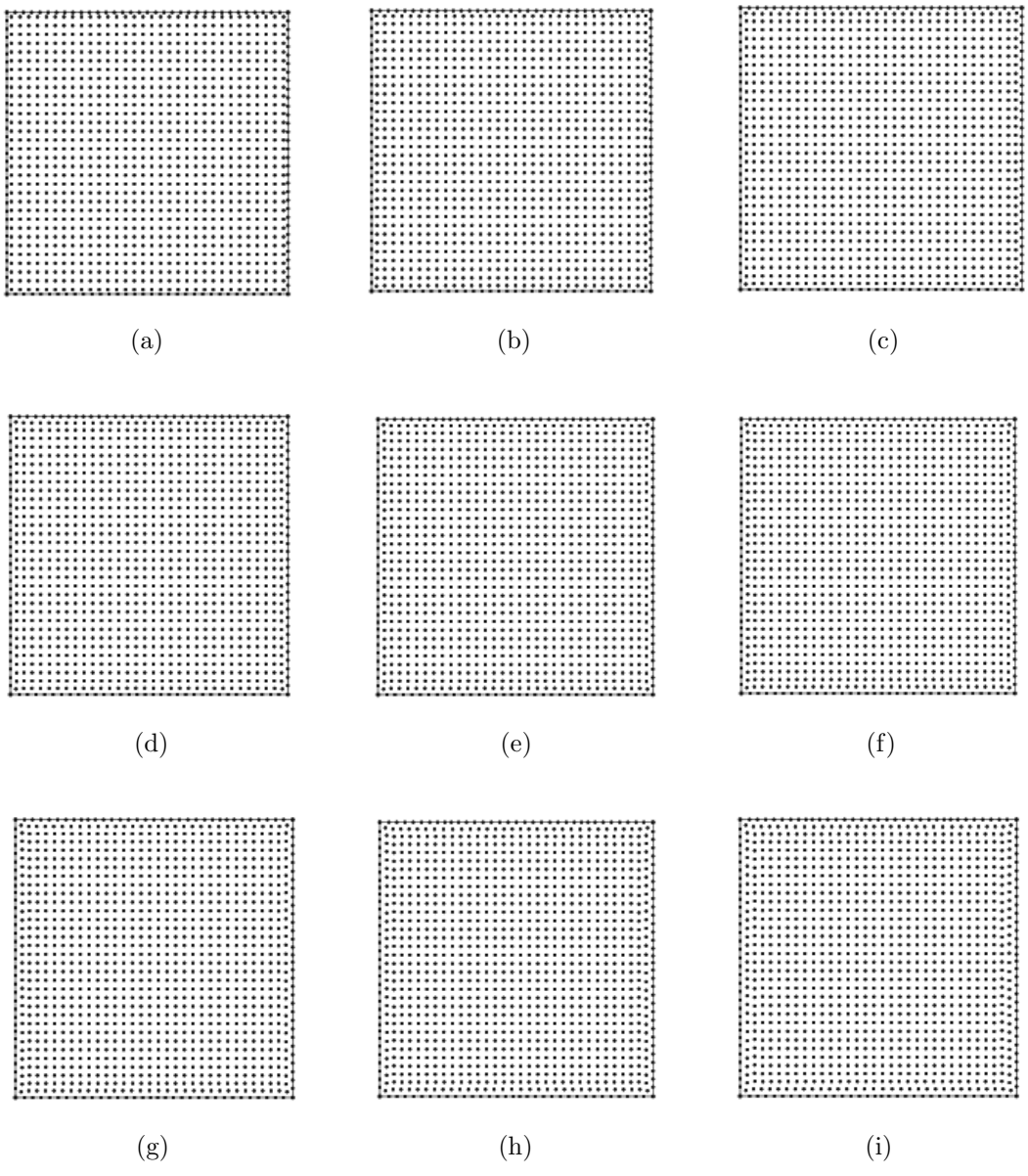
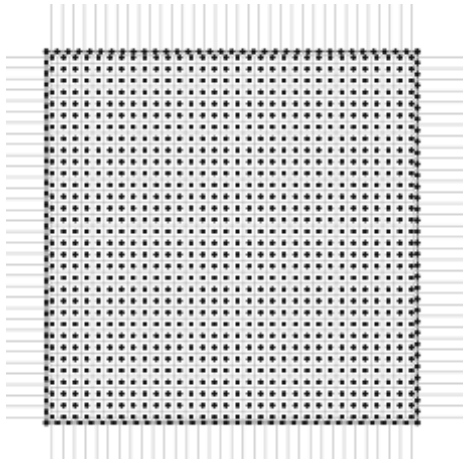
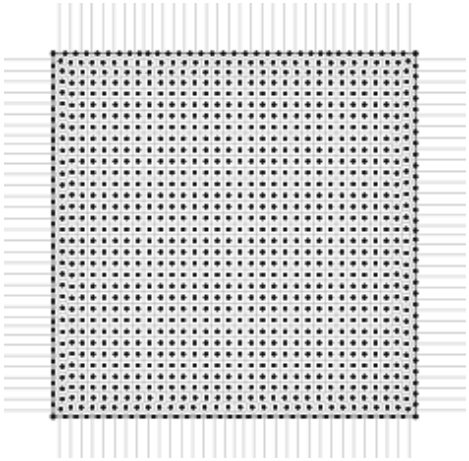


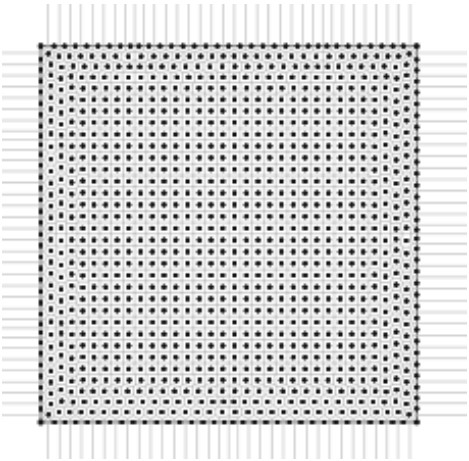
Figura 4.3: Iterações *Lloyd* passo a passo a partir de uma nuvem de pontos gerados segundo o método baseado no refinamento adaptativo (ALG). Note que os deslocamentos dos pontos são bastantes sutis, mais evidentes nas bordas.



(a)



(b)



(c)

Figura 4.4: Iterações *Lloyd* passo a passo a partir de uma nuvem de pontos gerados segundo o método baseado no refinamento adaptativo (ALG), mostrando as células *Voronoi*. (a) refere-se à Figura 4.3a, (b) refere-se à Figura 4.3e e (c) refere-se à Figura 4.3i.

4.2 Tempo de Execução da Geração da Nuvem Inicial de Pontos

Quando as duas soluções foram discutidas, ficou evidente a diferença da complexidade dos dois métodos. A geração de pontos aleatórios é muito mais simples e, como era de se esperar dada sua natureza incremental e custo computacional garantidamente $O(1)$, que realizaria a tarefa mais rapidamente. No entanto, será que impactou o tempo total e a qualidade da distribuição de pontos final?

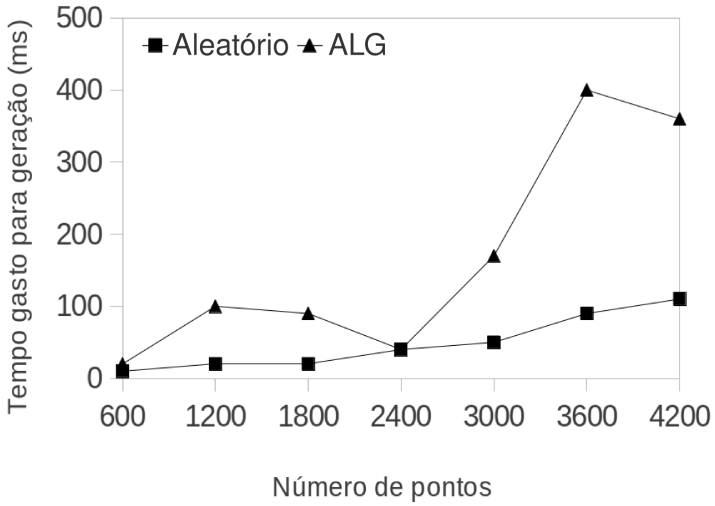


Figura 4.5: Avaliação do custo para geração da nuvem inicial de pontos.

No Gráfico 4.5, podemos analisar a evolução do tempo gasto para a geração de um número cada vez maior de pontos. Percebemos que a linha que representa o processo de geração aleatória cresce linearmente junto com o número de pontos, como era esperado. Já o processo baseado no ALG possui um comportamento não linear. Isto se deve ao fato do algoritmo trabalhar com níveis múltiplos de quatro.

Quando precisamos quebrar um nível, permitindo que somente parte das células seja promovida para o próximo nível, inserimos um processamento especial, mais caro e que acarreta impacto no custo total do algoritmo. Isto explica porque a linha não evolui sempre de forma linear.

Precisamos, então, avaliar se o investimento maior nesta tarefa é justificado, trazendo benefícios nas seguintes.

4.3 Tempo de Execução do Processo de Refinamento e Número de Iterações Necessárias para Atingir a Qualidade Estipulada

Podemos ver no Gráfico 4.6 e 4.7 que o algoritmo mais complexo, dada sua natureza sistemática para inclusão de pontos e, consequentemente, a geração de uma nuvem mais organizada, que tanto o tempo quanto o número de iterações do algoritmo de *Lloyd* necessários para atingir a qualidade foram menores, o que é bastante óbvio, já que desde a criação da nuvem existia a preocupação de que os pontos fossem distribuídos de forma equidistante ao longo da superfície.

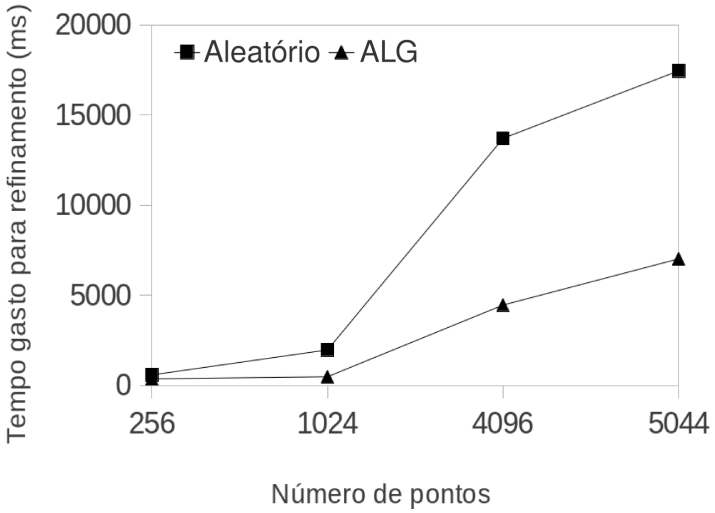


Figura 4.6: Avaliação do custo para refinamento.

Este comportamento acontece mesmo quando o nível do refinamento adaptativo não é o mesmo para todas as suas células. Isto porque, mesmo existindo a variação de densidade, ainda assim a disposição original é mais organizada que a resultante do processo aleatório.

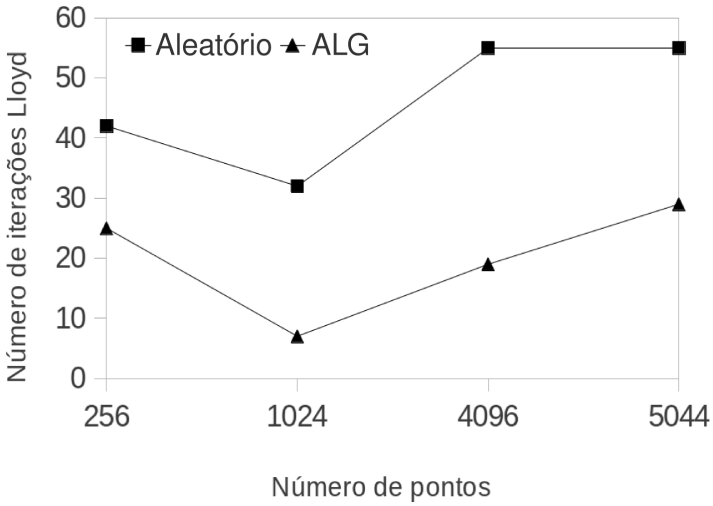


Figura 4.7: Avaliação do número de iterações do algoritmo de *Lloyd* para atingir a qualidade estipulada.

4.4 Análise Visual do Resultado da Geração e Refinamento da Nuvem de Pontos

Nesta Seção vamos nos ater à disposição dos pontos. Usamos a geometria mais simples, um quadrado, para tornar fácil a visualização da distribuição em si, sem influência de eventuais complicações oriundas de uma geometria mais complexa. Analisando a Figura 4.8, podemos ver:

- Em 4.8a a distribuição inicial de pontos segundo a geração aleatória, aproximando-se da densidade esperada mas sem se preocupar com a disposição de cada ponto levando-se em conta o restante da nuvem. Podemos encontrar, como era de se esperar, clarões ou amontoados de pontos, resultado da inserção sem critério;
- Em 4.8b a distribuição inicial de pontos segundo o método baseado no ALG. É evidente, até mesmo devido ao método utilizado, que a malha resultante é organizada mesmo antes do processo de refinamento. No entanto, não é garantido que os pontos mais

próximos ao contorno incidam sobre eles, o que pode provocar uma densidade diferente do restante da superfície nestas regiões;

- Em 4.8c o resultado do refinamento. Neste caso, era esperado um refinamento homogêneo ao longo de toda a superfície e, desta forma, a distância é bem similar para quaisquer dois pontos próximos entre si;

- Em 4.8d o resultado do refinamento que enfatiza as arestas do contorno. Os pontos de nuvens heterogêneas recebem um peso na etapa de refinamento. Neste caso em particular, quanto mais próximo da aresta, menor é o peso do ponto e, consequentemente, menor sua área de influência, fazendo com que os pontos sejam mais próximos uns dos outros nestas regiões;

- Em 4.8e o resultado do refinamento que enfatiza pontos de interesse do domínio que, neste caso, são os dois pontos das extremidades superiores. Quanto mais próximo deles, menor o peso atribuído ao ponto;

- Em 4.8f o resultado do refinamento que enfatiza o segmento de reta que identifica particular interesse no domínio. Veja que existe um segmento de reta que atravessa o quadrado na diagonal. Ela não faz parte do domínio e não segmenta o quadrado. Trata-se, porém, de um elemento imaginário que influencia a disposição dos pontos da nuvem. Quanto mais próximo deles, menor o peso atribuído ao ponto;

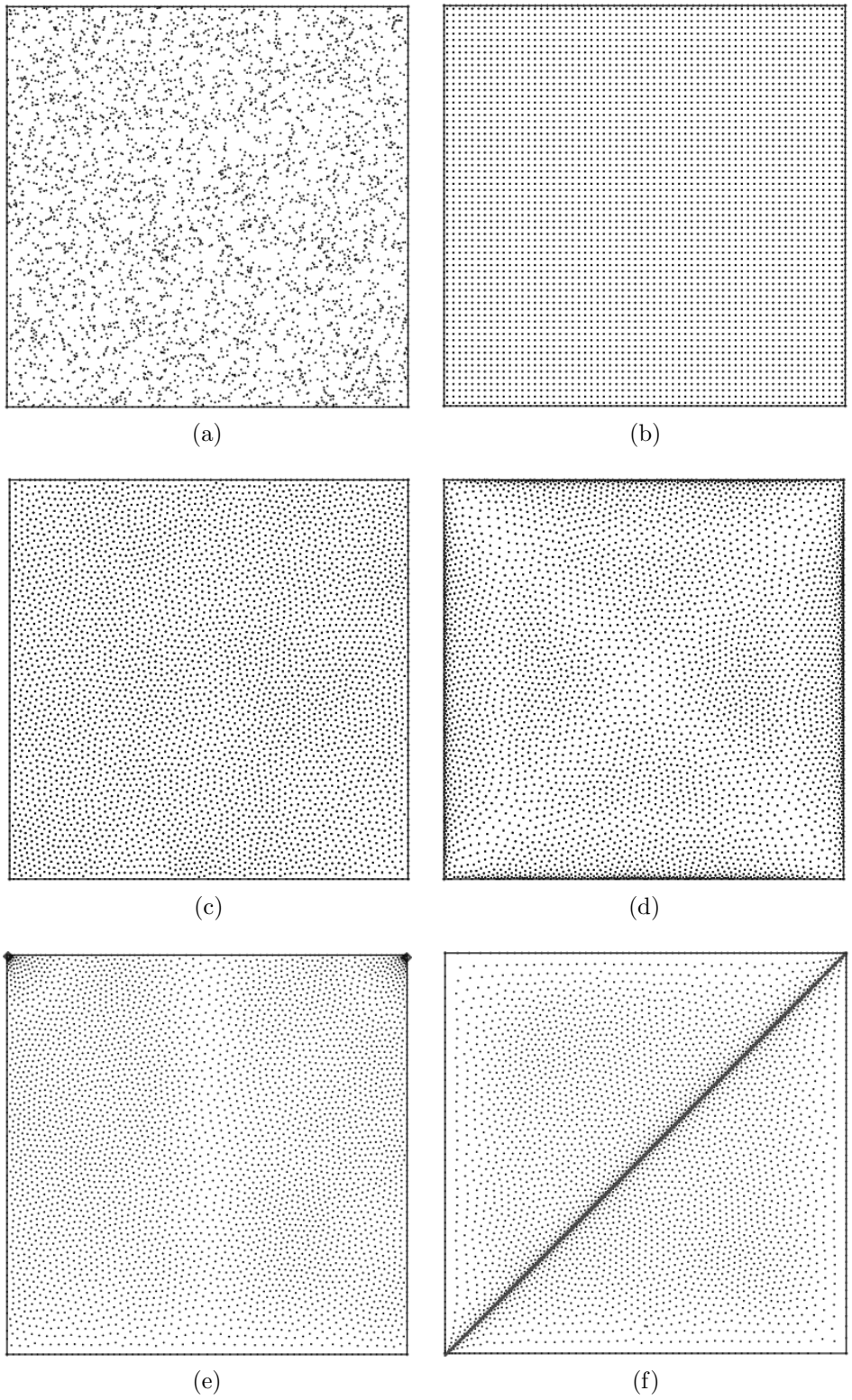


Figura 4.8: Análise visual do resultado da geração e refinamento da nuvem de pontos.

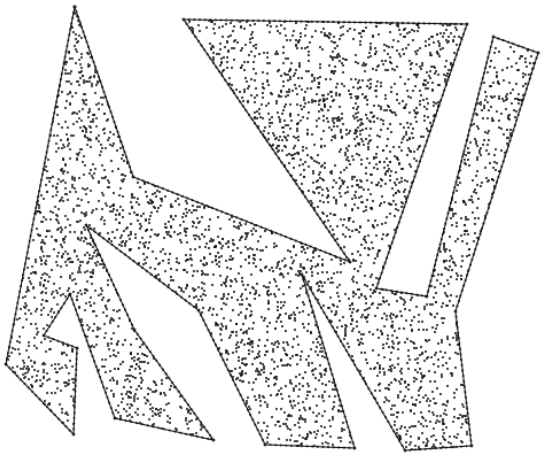
4.5 Análise Visual da Distribuição da Nuvem de Pontos em Domínios Complexos

Nesta seção vamos analisar alguns cenários mais complexos na configuração do(s) domínio(s). O primeiro caso, ilustrado na Figura 4.9 é de um polígono de contorno sinuoso. É importante que a distribuição de pontos considere a superfície e o contorno, e que a configuração de densidade seja garantida em toda a nuvem. Na Figura 4.9a temos o objeto com a distribuição de pontos aleatória inicial, refinada de forma homogênea em 4.9b e enfatizando as bordas (um exemplo de refinamento heterogêneo) em 4.9c.

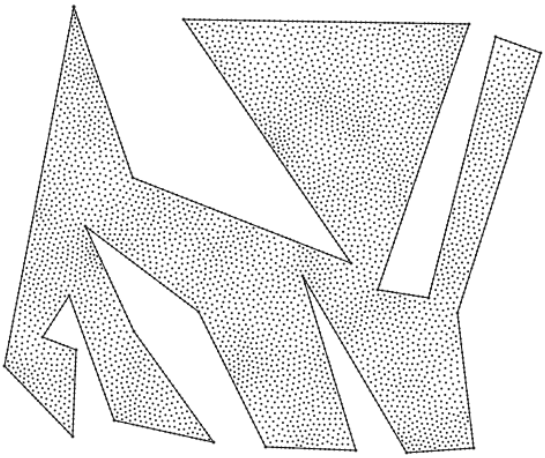
Outro caso bastante comum em problemas reais é um objeto contendo um furo ou outro objeto em seu interior. Neste caso, dependendo do problema, é necessário ou não uma nuvem de pontos no contorno interno, mas como nuvens de pontos diferentes. Na Figura 4.10a ilustramos um objeto furado em que não é necessário distribuir pontos da região do furo. Observe que mesmo não existindo pontos no interior do furo, ainda precisamos distribuir pontos ao longo do contorno interno para representar corretamente esta fronteira.

Nas Figuras 4.10b, 4.10c e 4.10d ilustramos um mesmo exemplo de polígono furado, mas considerando agora que seja necessária a distribuição de pontos na região interna, seja ela um furo ou outro objeto. Em 4.10b atenderíamos a necessidade de enfatizar as bordas do objeto interno, enquanto que em 4.10c e 4.10d, atenderíamos a necessidade de densidades diferentes. Observe que neste caso, a distribuição de pontos ao longo do contorno interno segue o objeto de maior densidade: em 4.10c segue a distribuição do objeto externo e em 4.10d do interno. No entanto, estes pontos sobre a aresta fronteira fazem parte da nuvem dos dois objetos.

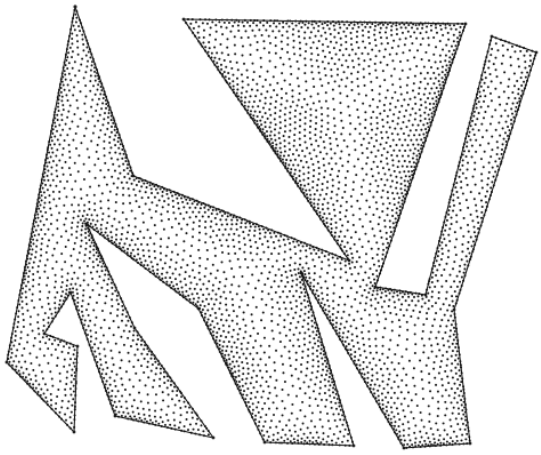
Por fim, uma outra configuração possível é de agrupamento de objetos, que compartilham arestas e vértices como visto na Figura 4.11. Cada objeto possui sua própria nuvem de pontos, podendo ser todas de mesma densidade, como em 4.11a ou de densidades diferentes, como em 4.11b. Da mesma forma que acontece com objetos furados, as arestas compartilhadas seguem a distribuição de pontos do objeto de maior densidade.



(a)



(b)



(c)

Figura 4.9: Análise visual do refinamento da nuvem de pontos em polígono de contorno mais sinuoso.

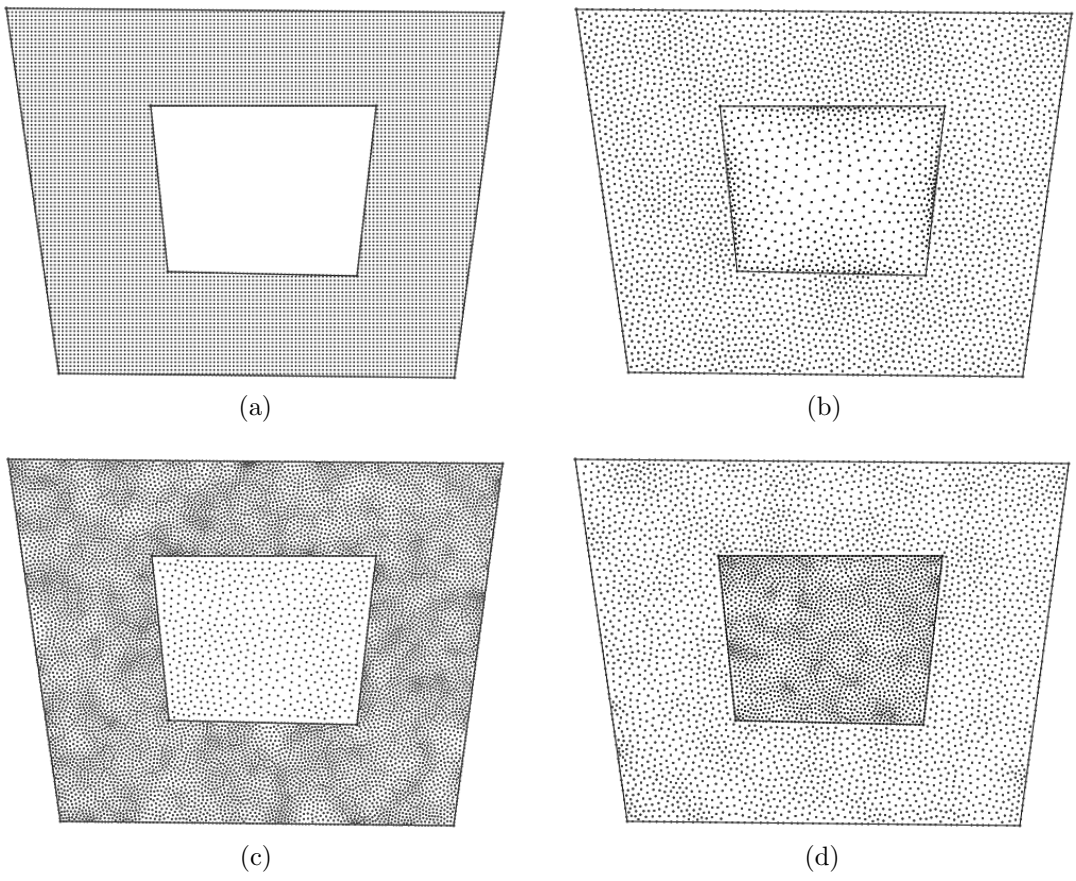
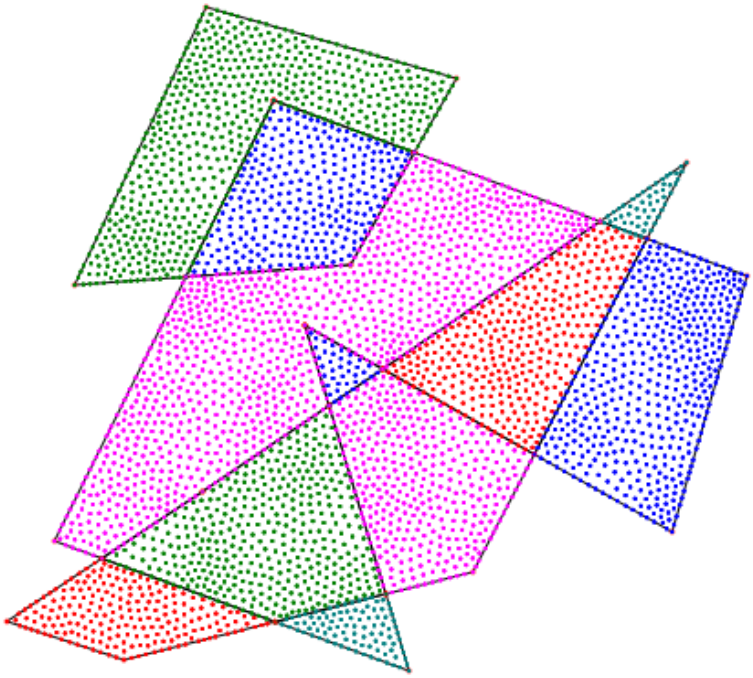
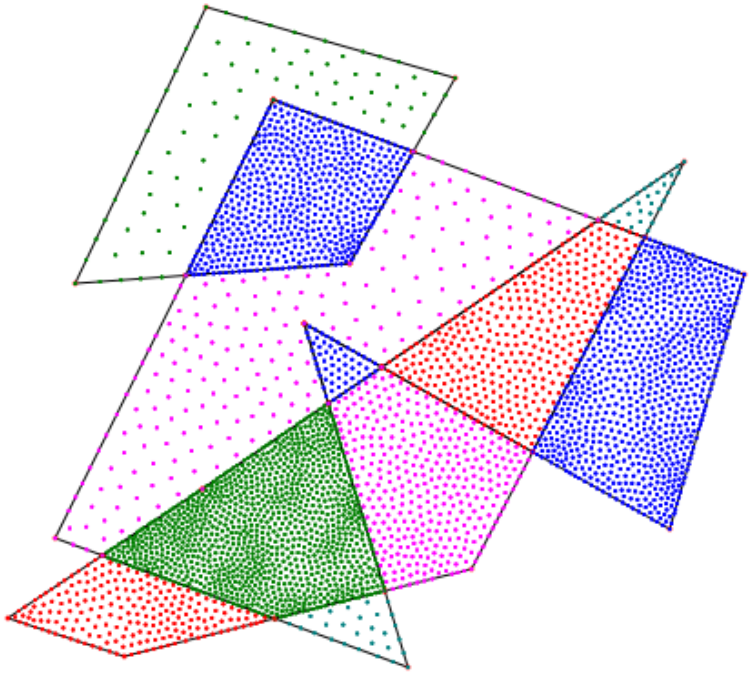


Figura 4.10: Análise visual do refinamento da nuvem de pontos em polígono com furo.



(a)



(b)

Figura 4.11: Análise visual do refinamento da nuvem de pontos em uma coleção de polígonos.

4.6 Avaliação Final Baseada em Testes Práticos

Os processos de geração da nuvem inicial de pontos, independente do método usado para realizar esta tarefa, são executados definindo-se um número de pontos qualquer como objetivo a ser alcançado. No entanto, o que temos definido no problema é a densidade da distribuição e, mesmo assim, com possibilidade dela variar ao longo da superfície caso a distribuição seja heterogênea.

O que acontece, na prática, é que quando a densidade é um requisito rígido (que precisa ser atendido de forma exata), o número de pontos da nuvem pode variar ao longo das iterações de forma a garantir que a densidade seja cumprida. Quando, em determinada iteração, altera-se a organização atual dos pontos pela inserção ou exclusão de pontos, prejudica-se o refinamento conseguido até então, exigindo então que todo ele seja reiniciado. Quando isso acontece, perde-se o benefício da geração organizada da nuvem inicial usando o método baseado no refinamento adaptativo (ALG), bastante complexo, e passa-se a tratar de uma distribuição tão aleatória quanto aquela gerada pelo processo desde o início assim caracterizado e, por sinal, muito mais simples.

Quanto à qualidade da distribuição de pontos da nuvem resultante, independe do processo seguido durante a etapa de geração. É, portanto, um objetivo parametrizado de forma única, verificado periodicamente, permitindo a conclusão do refinamento somente após o cumprimento dos critérios configurados.

Por outro lado, se a densidade da nuvem de pontos for um requisito de valor mínimo, ou seja, qualquer densidade acima daquela determinada é igualmente válida, é possível que ela seja garantida ainda durante a etapa de geração da nuvem de pontos inicial. Neste caso, investir em num processo de geração criteriosa de pontos é justificado, uma vez que o número de iterações de refinamento para atingir a qualidade esperada será menor e, consequentemente, menor também o tempo de execução da solução como um todo.

4.7 Teste Prático: um problema eletrostático

O problema da calha é um problema da eletrostática tratado em várias propostas de métodos numéricos, em particular o proposto por Zoia Lima [30], um *framework* para métodos sem malha escolhido para avaliar as nuvens de pontos geradas pela aplicação que implementa as propostas deste trabalho.

A Figura 4.12 ilustra o problema em duas dimensões. As condições de contorno essenciais são todas constantes. Observe que há um salto no valor do potencial elétrico nas regiões próximas aos vértices $(0, 4)$ e $(4, 4)$, de $10V$ para $100V$. Este salto torna o problema de difícil solução para os métodos numéricos, que não preveem essa descontinuidade.

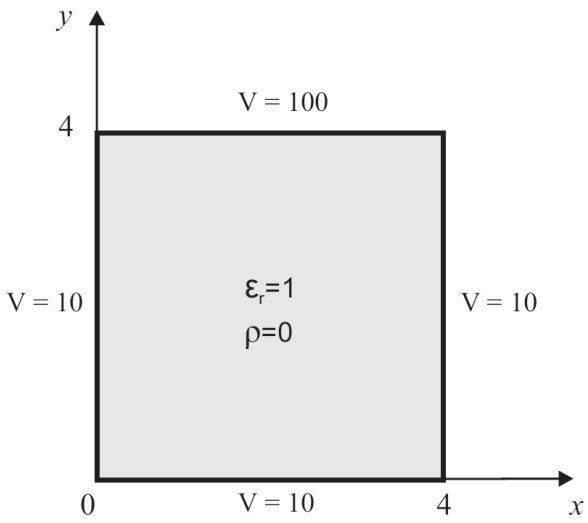


Figura 4.12: Calha em duas dimensões. Não há densidade de carga e a permissividade elétrica relativa do meio é igual a um. As condições de contorno essenciais são todas constantes. O potencial elétrico é dado em $Volts$.

A calha é, no entanto, um problema com solução analítica possível (Figura 4.13), o que torna fácil a avaliação da solução aproximada usando o método numérico escolhido, *Meshless Local Petrov-Galerkin Method* versão 1 [30], para diferentes configurações de distribuição de pontos.

Partindo-se então para a solução numérica, selecionamos quatro nuvens com disposições diferentes para os mesmos 1156 pontos, quantidade arbitrariamente escolhida para este exemplo. A primeira delas é uma nuvem regular, possível de ser gerada devido a geometria simples deste problema: um quadrado. Ela possui, portanto, a disposição mais equidistante possível

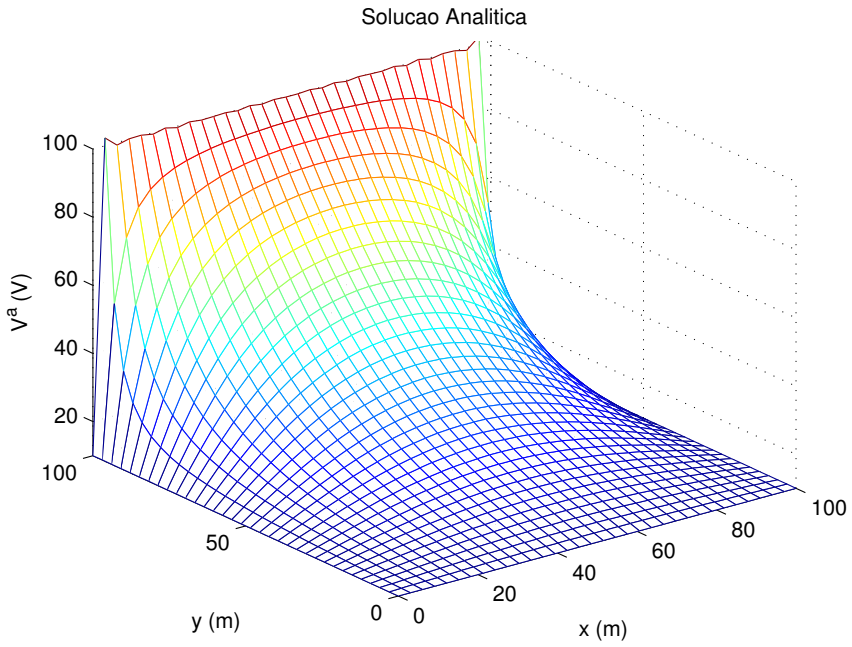


Figura 4.13: Solução analítica para o problema da calha em duas dimensões

dentre as nuvens homogêneas, dispondo os pontos em uma grade 34×34 (Figura 4.14a). As três demais foram geradas automaticamente segundo a solução genérica apresentada neste trabalho, sendo que a 4.14b é uma distribuição homogênea, enquanto que 4.14c e 4.14d são distribuições heterogêneas, ajustadas à configuração do problema tratado e, assim, tentando alcançar um resultado mais próximo da solução analítica.

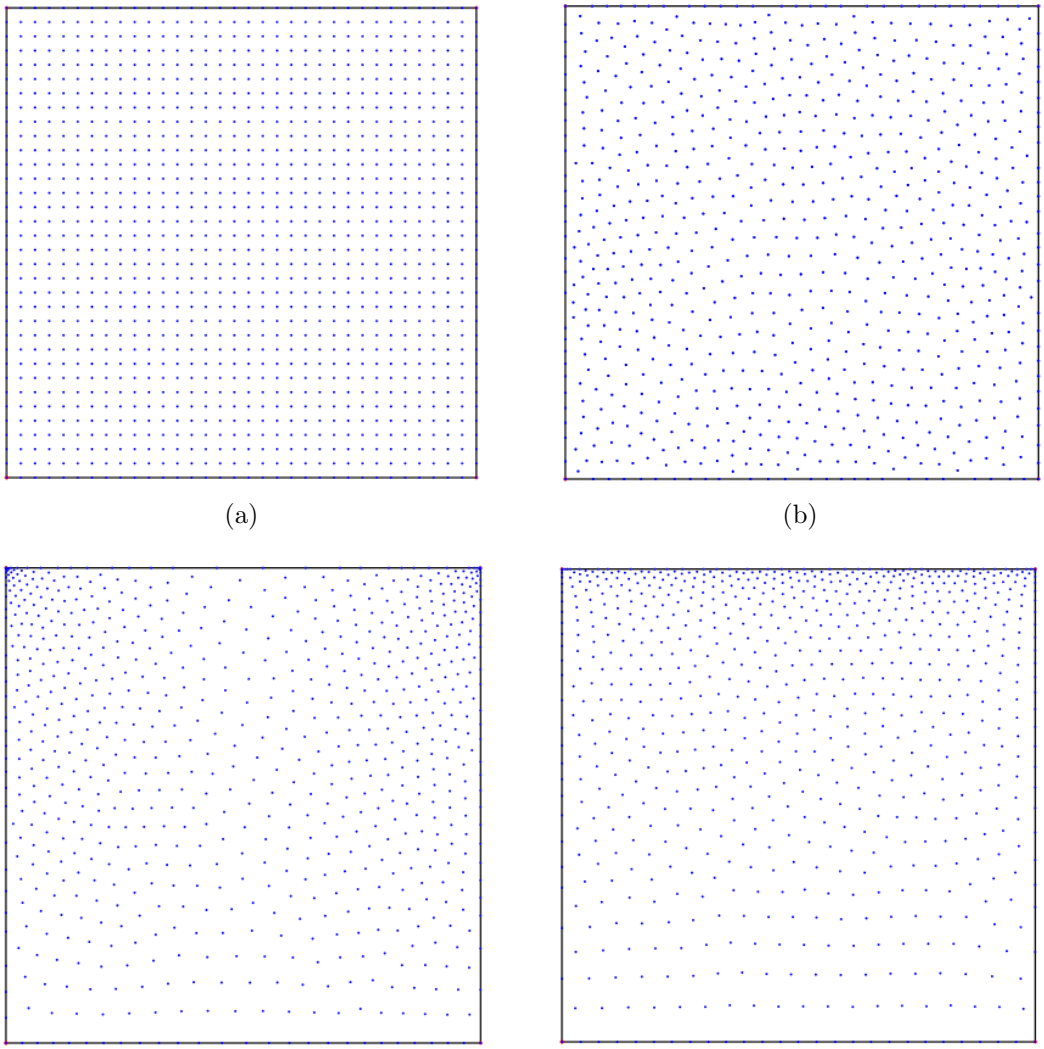


Figura 4.14: Nuvens de pontos usadas para solução do problema da calha em duas dimensões: (b) distribuição regular 34×34 pontos, (b) distribuição homogênea, (c) distribuição heterogênea enfatizando as extremidades superiores do objeto, (d) distribuição heterogênea enfatizando a parte superior do objeto.

A qualidade da solução é dada calculando-se o erro absoluto percentual em cada ponto da nuvem quando comparado com a solução analítica, dado por:

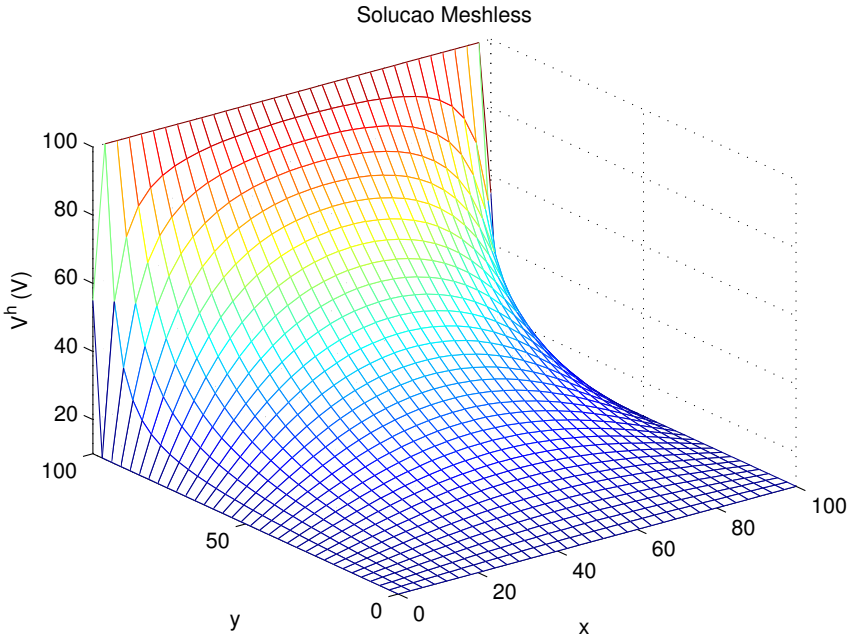
$$erro = 100 \times \left| \frac{V^a - V^h}{V^a} \right| \quad [\%] \tag{4.1}$$

onde V^h é a solução numérica e V^a a analítica. Analisando os resultados do método numérico usando a nuvem regular, ilustrado na Figura 4.15, especificamente quanto ao gráfico de superfície de erro resultante (Figura 4.15b), percebemos que os maiores valores de erros estão concentrados nas extremidades de maiores valores na coordenada y, justamente onde há descontinuidade do potencial elétrico no objeto modelado. É de se esperar um erro mais elevado nesta região, pois o método numérico não consegue reproduzir este comportamento descontínuo na solução.

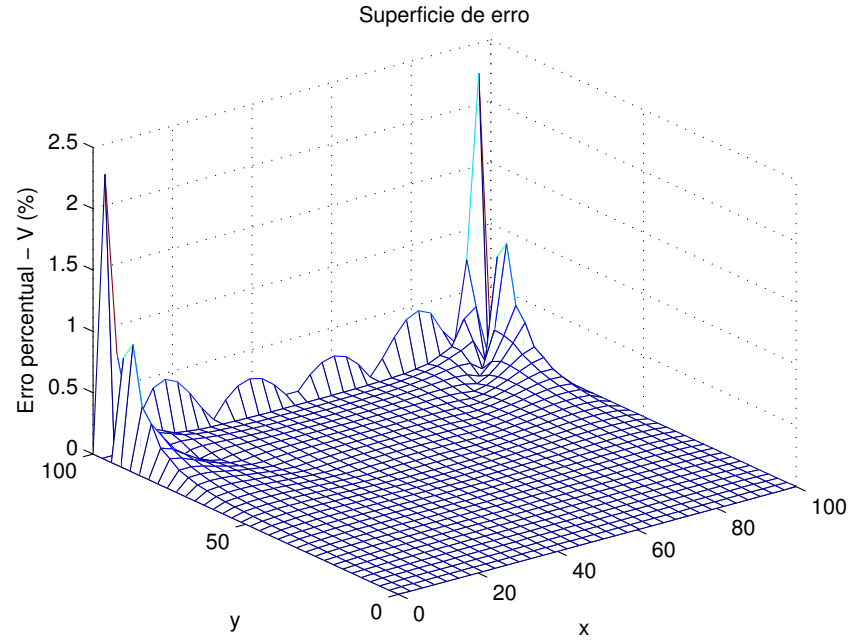
Para minimizar este erro, a nuvem de pontos precisa ser mais densa nas regiões onde a descontinuidade ocorre, como as disposições ilustradas nas Figuras 4.14c e 4.14d. Analisando-se o erro médio total para cada nuvem, dado por:

$$erro = \frac{\sqrt{\int_\Omega (V^h - V^a)^2 d\Omega}}{\sqrt{\int_\Omega (V^a)^2 d\Omega}} \tag{4.2}$$

cujos resultados são mostrados no Gráfico 4.16, concluímos que estas nuvens, como esperado, são mais apropriadas, gerando um erro médio significativamente menor.



(a)



(b)

Figura 4.15: Solução *meshless* para o problema da calha em duas dimensões usando um arranjo estruturado de pontos 34×34: (a) mostra a solução propriamente dita e (b) a superfície de erro.

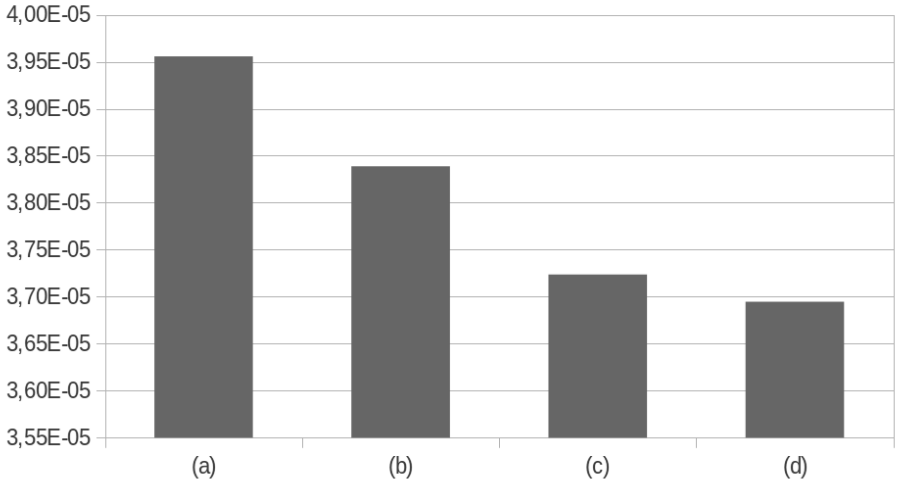


Figura 4.16: Norma do erro da aproximação numérica. (a) na distribuição de pontos regular 34×34, ilustrada na Figura 4.14a, (b) na distribuição de pontos irregular homogênea, ilustrada na Figura 4.14b, (c) na distribuição de pontos irregular heterogênea, enfatizando as quinas de maiores valores para a coordenada y, ilustrada na Figura 4.14c e (d) na distribuição de pontos irregular heterogênea, com maior densidade quanto maior valor para a coordenada y, ilustrada na Figura 4.14d.

Capítulo 5

Conclusão

Neste trabalho desenvolvemos uma aplicação para geração de nuvem de pontos para uso em métodos sem malha. A construção usou a linguagem de programação $C++$, agregando ainda bibliotecas como CGAL (Apêndice A) e GPC (Apêndice B), resultando em um projeto em blocos funcionais bem definidos, estáveis e computacionalmente eficientes.

Inicialmente, revisamos os principais conceitos que seriam utilizados neste trabalho, como algoritmos que realizam operações geométricas, diagrama de *Voronoi*, sobretudo o *Voronoi* centroidal (Seção 2.4.3), e algoritmo de *Lloyd* (Seção 2.4.4). Abordamos também algumas outras abordagens para o problema tratado, como extração de pontos de uma malha pré-gerada, geração de pontos baseada no domínio de influência de círculos espalhados ao longo da área do objeto [19, 16, 17, 18] e extração de pontos de uma malha regular retangular ajustada à geometria do objeto [27]. Nosso objetivo era de propor uma solução computacionalmente viável gerando uma nuvem de pontos de qualidade aceitável segundo duas propostas distintas, posteriormente comparadas.

Nosso primeiro desafio foi identificar a geometria do problema a partir de esboços feitos pelo usuário. A aplicação oferecia, para este fim, um ambiente simplificado para inserção de traços que seriam processados para extração de polígonos. Ainda nesta etapa, identificamos regiões de interesse diferenciado, e recebemos configurações de densidade da distribuição em cada uma delas e dos polígonos identificados. Como resultado, temos o contorno dos objetos que

compõem o problema devidamente modelados e com as devidas configurações, prontos para a distribuição de pontos sobre suas superfícies.

A próxima etapa foi a distribuição de uma nuvem de pontos inicial no interior do objeto sem, necessariamente, atender aos requisitos de qualidade. Fizemos isso de duas formas: segundo uma distribuição aleatória e segundo uma distribuição baseada na divisão recursiva do espaço usando a estrutura *Autonomous Leaves Graphs* (ALG). O objetivo foi comparar o impacto de cada abordagem na tarefa seguinte, de refinamento da nuvem, que detém a maior parcela do custo computacional total da solução do problema.

Complementando a nuvem até então gerada e independente do método usado para isso, distribuimos pontos nos limites externos, fronteiras, limites internos, vértices, e nos pontos e segmentos identificados como regiões de particular interesse.

Finalmente, usando o algoritmo de *Lloyd* para reposicionar os pontos até atingirmos uma distribuição como em um diagrama de *Voronoi* centroidal, realizando o refinamento iterativo até que o qualidade e densidade necessárias fossem conseguidas. O número de iterações necessárias era o que queríamos avaliar quando variamos o método de geração, já que a qualidade, igual para ambos os métodos, não poderia ser parâmetro.

Apresentamos os resultados da solução proposta. Quanto aos dois métodos de geração da nuvem inicial testados, discutimos a contribuição de cada um: uma distribuição aleatória é muito mais simples para ser implementada, mas exige mais iterações de refinamento, enquanto uma geração mais elaborada, que desde o início preza por uma distribuição mais organizada, converge para a solução esperada muito mais facilmente. No entanto, se a aproximação da densidade for muito distante da esperada, a diferença entre os métodos deixa de existir, uma vez que durante as iterações de refinamento uma nuvem originalmente organizada deixa de sê-lo ao se retirar ou inserir pontos durante as iterações.

Apresentamos a distribuição de pontos gerada em diversos objetos, com contornos diferentes, com furos, interfaces, e configuração de densidade diferente. Mostramos a distribuição quando o problema pede uma distribuição homogênea de pontos e quando a distribuição se mostra diferente ao longo da superfície, seja para diferenciar uma região de interesse particular, fronteiras e contornos.

Para permitir integração com outras aplicações, foi implementada uma forma de se importar coordenadas que formam polígonos e de se exportar a nuvem resultante para arquivo. No caso da exportação, os pontos resultantes são identificados através de ids, suas coordenadas e o tipo de ponto que representa, seja um ponto interno, de contorno, de fronteira, de contorno de furo ou vértice. Este arquivo pode ser usado em qualquer aplicação baseada em métodos sem malha.

Ao longo dos trabalhos, identificamos muitas possibilidades de melhorias. Elas não foram implementadas porque, além de exigirem grande esforço e tempo de desenvolvimento, não fazerem parte do escopo deste trabalho. É o caso, por exemplo, da criação de uma interface de desenho mais elaborada, que permita o usuário inserir desenhos conhecidos como retângulos, quadrados e círculos, definição de valor preciso para coordenadas e dimensões, e ações como união e intercessão entre objetos.

A versão atual da aplicação gera nuvens de pontos em duas dimensões, com coordenadas nos eixos x e y. Uma necessidade importante para a solução de problemas práticos é a geração da núvem de pontos com coordenadas nos eixos x, y e z, seja para gerar nuvem sobre a superfície ou ao longo de todo o volume de objetos em três dimensões. É, portanto, um recurso bastante interessante para ser agregado na aplicação em trabalhos futuros, cujos exemplos de aplicações práticas podem ser vistas em Schembri et al. [26].

Por fim, uma outra possibilidade de trabalho futuro que afetaria não a qualidade do resultado ou a capacidade da aplicação, mas sim à melhoria no desempenho nas execuções dos algoritmos mais pesados, especificamente os essencialmente paralelizáveis, é sua execução em GPU (*Graphics Processing Unit* ou Unidade de Processamento Gráfico) ao invés da atual execução em CPU (processador). Tradicionalmente, a diferença fundamental entre CPU e GPU é o fato de que as CPUs são otimizadas para cálculos sequenciais enquanto as GPUs são otimizadas para cálculos massivamente paralelos. Antigamente essa divisão era bem mais nítida, já que as CPUs processavam apenas triângulos, texturas e efeitos simples. Entretanto, com a introdução do uso de shaders, que nada mais são do que pequenos aplicativos destinados a executar tarefas específicas na composição das cenas, elas ganharam a capacidade de também executar código sequencial, assim como os CPUs, possibilitando que ela assuma o processamento das tarefas da geração da nuvem de pontos. Apesar do poder de processamento muito inferior, a GPU possui grande número de núcleos (apesar da variação entre marcas e modelos, é ainda um número muito superior a qualquer CPU de mercado), o que permite que

um algoritmo otimizado para processamento paralelo ter grande ganho comparado com seu equivalente processado da forma tradicional.

Referências Bibliográficas

[1] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars. Computational Geometry - Algorithms and Applications. Springer, 3rd edition, 2008.

[2] D. Burgarelli, M. Kischinhevsky, and R. J Biezuner. A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's. Journal of Computational and Applied Mathematics, 196, pages 115–131, 2006.

[3] Denise Burgarelli. Modelagem Computacional e Simulação Numérica Adaptativa de Equações Diferenciais Parciais Evolutivas Aplicadas a um Problema Termoacústico. PhD thesis, Departamento de Matemática da PUC-RIO, 1998.

[4] Clodoveu Davis. Geometria Computacional para Sistemas de Informação Geográfica. Divisão de Processamento de Imagens (DPI) do Instituto Nacional de Pesquisas Espaciais (INPE), 2001.

[5] La Tecla de Escape. Area de un polígono irregular: producto en cruz. http://latecladeescape.com/con-nombre-propio/area-de-un-poligono-irregular-producto-en-cruz.html, 2011.

[6] Qiang Du, Maria Emelianenko, and Lili Ju. Convergence of the Lloyd algorithm for computing centroidal Voronoi tessellations. SIAM Journal on Numerical Analysis 44, pages 102–119, 2006. doi: DOI:10.1137/040617364.

[7] H. ElGindy, H. Everett, and G. T. Toussaint. Slicing an ear using prune-and-search. Pattern Recognition Letters, 14, pages 719–722, 1993.

[8] Jeff Erickson. Voronoi diagrams and Delaunay triangulations. http://compgeom.cs.uiuc.edu/~jeffe/compgeom/code.html#voronoi, 2011.

[9] Ester Ezra, Eyal Flato, Dan Halperin, Iddo Hanniel, Shai Hirsch, and Ron Wein. CGAL: 2D Arrangements. http://www.ics.uci.edu/~dock/manuals/cgal_manual/Arrangement_2/Chapter_main.html, 2011.

[10] A. R. Fonseca. Algoritmos Eficientes em Métodos sem Malha. PhD thesis, Universidade Federal de Minas Gerais, 2011.

[11] W. Randolph Franklin. PNPOLY - Point Inclusion in Polygon Test. http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html, 2011.

[12] Ryan Holmes. The DCEL Data Structure for 3D Graphics. http://www.holmes3d.net/graphics/dcel/, 2011.

[13] Thomas J. R. Hughes. The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Dover Publications, 2000.

[14] Alan Jeffrey. Applied Partial Differential Equations - An Introduction. Academic Press, Elsevier, 2002.

[15] D. E. Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley, 2th edition, 1973.

[16] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting: Advancing Front meets sphere packing. International Journal of Numerical Methods in Engineering, 1999.

[17] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting Spheres in 3D. Proc. 8th Meshing Roundtable, 1999.

[18] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting Ellipses to Generate Anisotropic Meshes. Proc. 8th Meshing Roundtable, 1999.

[19] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Point placement for meshless methods using Sphere packing and Advancing Front methods. ICCES'00, Los Angeles, USA, 2000.

[20] A. Margalit and G. D. Knott. An Algorithm for Computing the Union, Intersection or Difference of Two Polygons. Computers & Graphics 13(2), pages 167–183, 1989.

[21] Nuno Lopes Martins. Classificação e partição de polígonos simples. PhD thesis, Departamento de Matemática da Universidade de Aveiro, 2005.

[22] G. H. Meisters. Polygons have ears. American Mathematical Monthly 82, pages 648–651, 1975.

[23] Raquel Sofia Rebelo Nunes. Geometria Fractal e Aplicações. PhD thesis, Faculdade de Ciências da Universidade do Porto, 2006.

[24] F. P. Preparata and M. I. Shamos. Computational Geometry: an Introduction. Springer-Verlag, 1988.

[25] Ionildo José Sanches. Compressão sem Perdas de Projeções de Tomografia Computadorizada usando a Transformada Wavelet. http://www.dainf.ct.utfpr.edu.br/~ionildo/wavelet/cap3.htm, 2011.

[26] P. Schembri, D. L. Crane, and J. N. Reddy. A three-dimensional computational procedure for reproducing meshless methods and the finite element method. International Journal for Numerical Methods In Engineering, 61:896–927, 2004.

[27] Kamal Shanazari and Ke Chen. A minimal distance constrained adaptive mesh algorithm with application to the dual reciprocity method. Numerical Algorithms, 32, pages 275–286, 2003.

[28] Kamal Shanazari and Mohammad Hosami. A two dimensional adaptive nodes technique in irregular regions applied to meshless-type methods. 2010.

[29] A. Taflove. Computational Electrodynamics - The Finite-Difference Time-Domain Method. Artech House, Norwood, MA, 2000.

[30] Naísses Zoia Lima. Desenvolvimento de um Framework para Métodos sem Malha. Master's thesis, Universidade Federal de Minas Gerais, 2011.

Apêndices

Apêndice A

CGAL

Criada em 1996, a *Computational Geometry Algorithms Library* (CGAL) é uma biblioteca de software desenvolvida em $C++$ que visa facilitar o acesso aos algoritmos eficientes e confiáveis em geometria computacional. O *software* está disponível sob regime de licenciamento duplo: quando usado para outros *softwares* de código aberto, é disponível sob licenças de código aberto (LGPL ou QPL dependendo do componente). Em outros casos, a licença comercial pode ser adquirida com diferentes opções para os clientes de pesquisa / acadêmico e industrial.

Neste trabalho, utilizamos algoritmos da CGAL na solução de dois problemas:

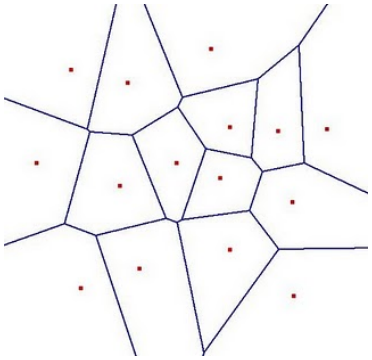


Figura A.1: Diagrama de *Voronoi* gerado pela CGAL.

1. Quando o usuário desenha o polígono no formato desejado, utilizamos os algoritmos de tratamento de arranjos "Arrangement_2", que recolhe o emaranhado de segmentos inseridos e converte em uma *Doubly-Connected Edge List* - DCEL, de onde extraímos cada polígono identificado. Com esta estrutura, podemos caminhar para qualquer sentido nas arestas, ter acesso fácil a faces e a todos pontos que representam os vértices.

2. Para proceder com o refinamento de uma nuvem de pontos, explicado em detalhes na Seção 3.3, precisamos gerar o diagrama de *Voronoi* (Figura A.1). A CGAL oferece os algoritmos para tal tarefa através do "Voronoi_diagram_2", gerando o diagrama de forma incremental, ponto a ponto. Terminada a geração, temos acesso à lista de células geradas na forma, também, de uma DCEL. Assim, podemos realizar as ações necessárias em cada polígono, representantes das células.

Uma vez adotada a CGAL temos acesso a todos os seus recursos vinculados como: iteradores, classes de pontos, pontos com peso, segmentos, polígonos, etc.

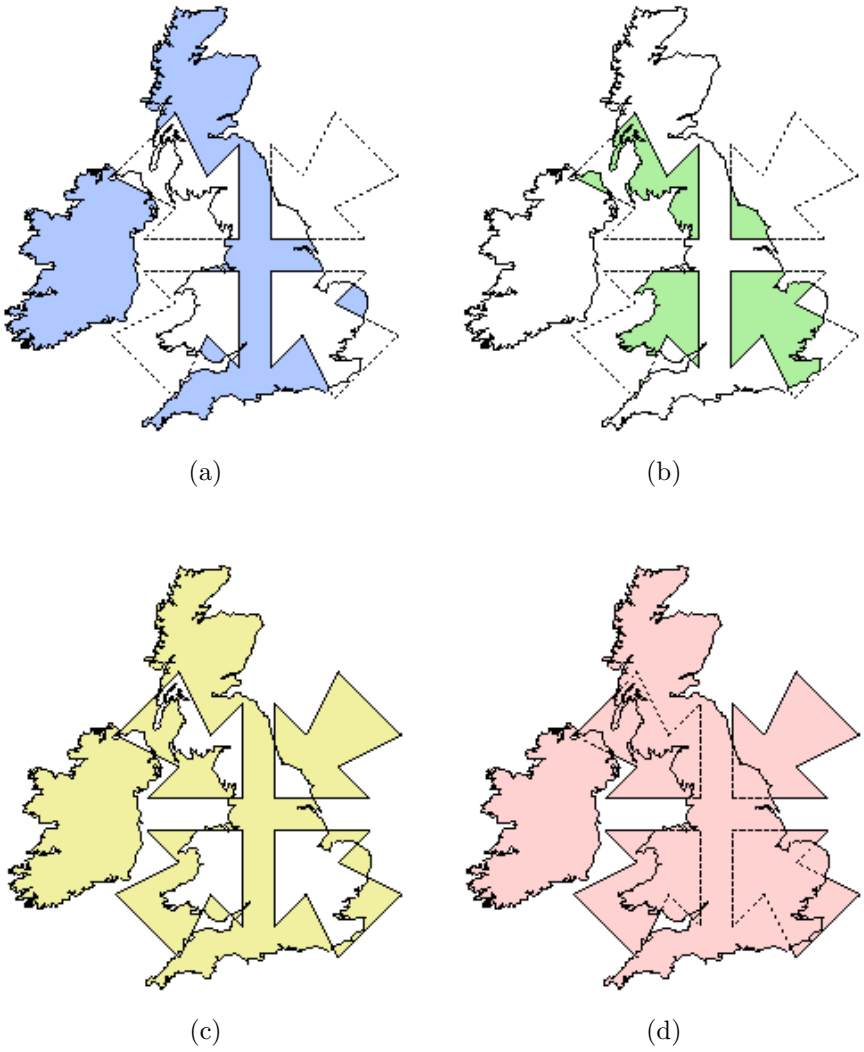
Apêndice B

GPC

O *General Polygon Clipper Library*, GPC, é uma biblioteca desenvolvida pela Universidade de Manchester para realização de operações em polígonos como, por exemplo, intercessão, diferença, ou-exclusivo e união (Figura B.1). Seus algoritmos são flexíveis e altamente robustos, podendo ser utilizados em projetos desenvolvidos usando linguagens como *C*, *C++*, *C#*, *Delphi*, *Java*, *Perl*, *Python*, *Haskell*, *Lua*, *VB.Net* entre outros.

Outras características que colaboraram com a adoção do GPC foi sua capacidade de tratar polígonos aninhados (buracos), arestas coincidentes e, ainda, sua indiferença de orientação de vértices. Este última característica foi particularmente importante, uma vez que em determinadas etapas da execução onde precisávamos da intercessão entre dois polígonos, tínhamos a sequência ordenada de vértices, mas não sabíamos sua orientação.

Apesar da CGAL também ser capaz de realizar tais operações, tivemos problemas na versão da aplicação cuja solução era puramente baseada nela, talvez por permitir diferentes modelos de aproximação numérica e, muitas vezes não suficiente para o nosso problema. A GPC foi capaz de resolver o problema de forma muito mais simples e com desempenho aceitável.



(a) (b)

(c) (d)

Figura B.1: Exemplo de operações em polígonos realizadas usando GPC: (a) diferença, (b) intercessão, (c) ou-exclusivo, (d) união.

Apêndice C

Interface da Aplicação Implementada Neste Trabalho

A aplicação possui uma interface bastante simplificada, com menu que alterna entre as funções de desenho e configuração/geração da nuvem de pontos. A Figura C.1 mostra a tela com o menu de desenho, onde à esquerda o usuário traça os segmentos de reta com o mouse enquanto que a direita ele visualiza a lista de objetos que ele vai criando. Ainda nesta tela, é possível alternar entre desenho de objetos ou de pontos ou segmentos que identificam áreas de particular interesse.

Na tela ilustrada na Figura C.2, podemos ver as configurações possíveis para a geração e refinamento da nuvem de pontos. É também aqui que se configura a densidade para cada objeto.

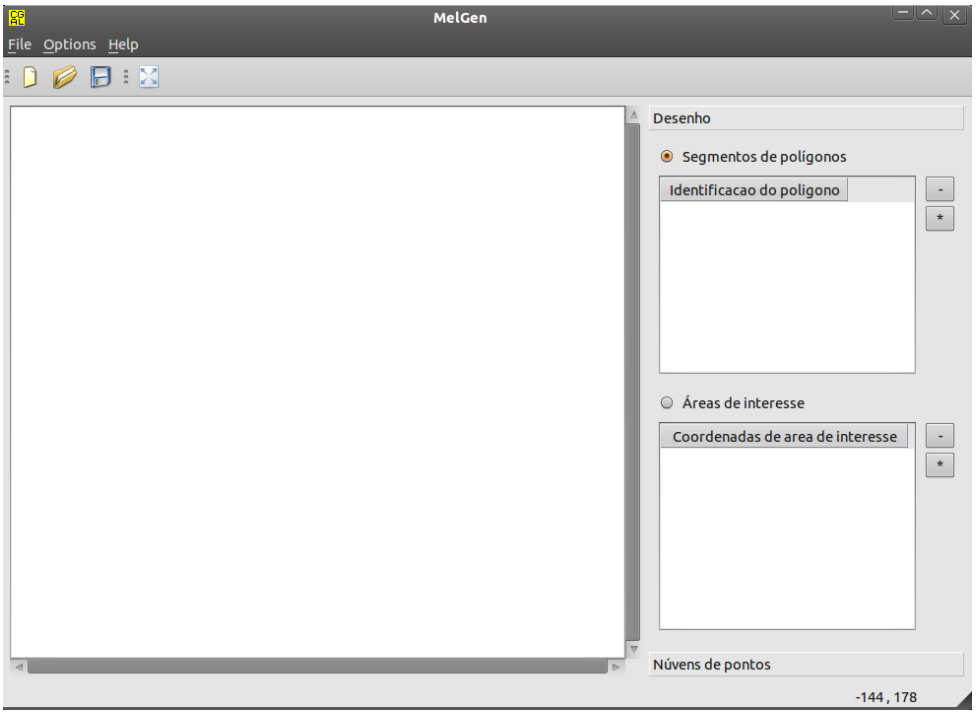


Figura C.1: Tela de desenho livre e marcação de regiões de interesse diferenciado.

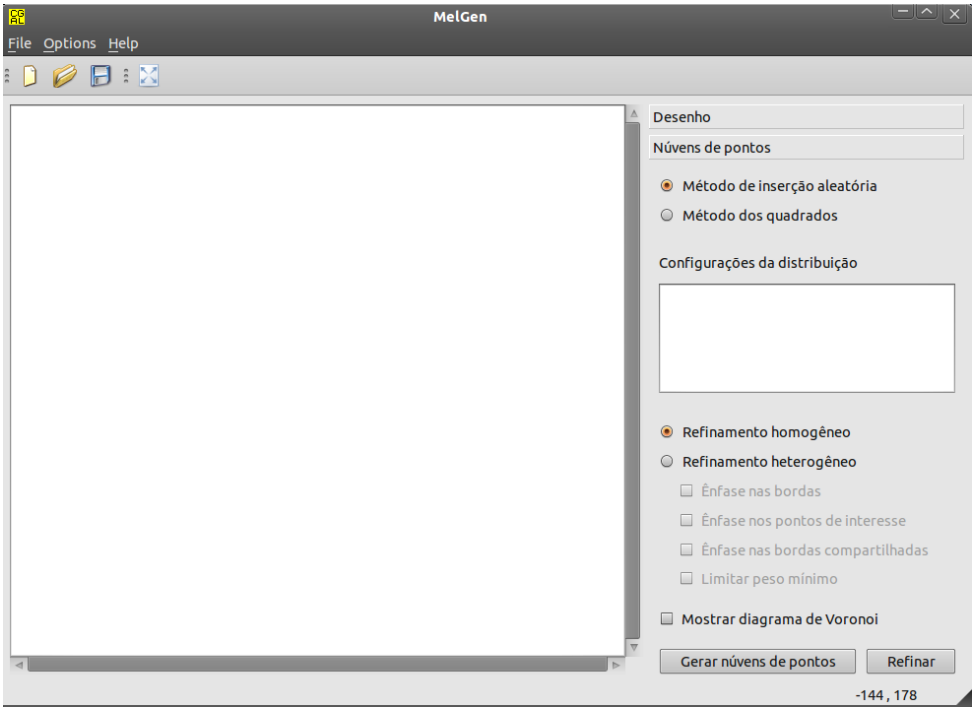


Figura C.2: Tela de geração e refinamento da nuvem.

Apêndice D

Entradas e saídas

Tratamos neste trabalho do problema de geração da nuvem de pontos, que é apenas uma das fazes para a solução de um problema usando método sem malha. Por se tratar de uma fase dentre muitas, é mandatório efetuar a integração com as demais fases.

Para isto, precisamos definir uma forma de entrar e sair com dados na aplicação. Como nossa proposta é a geração de uma nuvem de pontos, precisamos definir uma forma de se carregar uma geometria já existente para a definição dos polígonos, e também para salvar em arquivo o resultado da geração, listando as geometrias, pontos e respectivas coordenadas.

Para carregar as geometrias dos objetos modelados do mundo real, podemos tanto desenhar a partir da própria aplicação, utilizando o mouse em uma área destinada a este fim, quanto selecionar um arquivo texto. Se for escolhido o segundo modo, neste arquivo basta que existam as coordenadas dos vértices (pontos) do polígono ordenados segundo à ordem em que foram desenhados, seja no sentido horário ou anti-horário. Exemplificando dom um quadrado, teríamos:

*<x> <y> <polygonId>
0 0 0
100 0 0
100 100 0

0 100 0
0 0 0

Como se trata apenas de um polígono, o "id" foi "0" para todos os pontos. Os próximo objetos seriam de "id" "1", "2", "3" sucessivamente. As coordenadas são valores do tipo flutuante, *float* na linguagem *C++*. O caractere especial "*" informa que a linha deve ser descartada, sendo tratada apenas como um comentário para tornar o arquivo inteligível ao usuário. O caractere "espaço" ou "tab" é usado para separar valores. O último ponto possui coordenadas iguais ao primeiro, o que caracteriza o fechamento do caminho que contorna o polígono.

Ao fim da carga, na tela é exibido o objeto formado. A qualquer momento após a carga, é possível excluir ou incluir novos polígonos a partir dos recursos de desenho, afetando aqueles que foram carregados de arquivo.

Já o arquivo de saída, que pode ser gerado após o refinamento, possui o formato:

*DOMAIN_BEGIN

*POINTS_NUM
n_p n_p = número de pontos da geometria

*POINTS_COORD
id_p x y id_p = id do ponto da geometria
... x = coordenada x
y = coordenada y

*CURVES_NUM
n_c n_c = número de segmentos da geometria

*CURVES_DESC
id_c id_ps id_pt id_c = id do segmento (sentido horário)
id_ps = id do ponto inicial na geometria
id_pt = id do ponto final na geometria

*REGIONS_NUM
n_r n_r = número de regiões

*REGIONS_DESC
id_r id_m id_c id_r = id da região
... id_m = id do material da região
id_c = id de um segmento desta região

*INTERFACES_NUM
n_i n_i = número de interfaces

*INTERFACES_DESC
t_i id_c id_rl id_rr t_i = id da interface entre materiais
id_c = id do segmento desta interface
id_rl = id da região à esquerda
id_rr = id da região à direita

*NODES_NUM
n_n n_n = número de nós

*NODES_COORD
id_n x y id_n = id do nó
... x = coordenada
y = coordenada y

*NODES_REGIONS
id_n n_rn id_r1 id_r2 ... id_n = id do nó
... n_rn = número de regiões às quais ele pertence
id_r1 = id da região 1
id_r2 = id da região 2 (se houver)

*DOMAIN_END

Mais uma vez utilizando nosso exemplo do quadrado, a saída seria:

*DOMAIN_BEGIN

```
*POINTS_NUM
4

*POINTS_COORD
0 0 0
1 0 100
2 100 100
3 100 0

*CURVES_NUM
4

*CURVES_DESC
0 0 1
1 1 2
2 2 3
3 3 0

*REGIONS_NUM
1

*REGIONS_DESC
0 0 0

*INTERFACES_NUM
0

*INTERFACES_DESC

*NODES_NUM
262

*NODES_COORD
200 0 0
203 0 100
```

260 0 14.5919
259 0 20.7769
...

*NODES_REGIONS
0 1 0
1 1 0
2 1 0
3 1 0
...

Os campos selecionados para composição do arquivo de saída estão, conforme necessário para ser usado como entrada no trabalho de solução de problemas usando método sem malha de Zoia Lima [30].