Instituto de Matemática e Estatística
Universidade de São Paulo

Trabalho de Formatura Supervisionado

Teorema da Galeria de Arte e Triangularização de Polígonos e Pontos no Plano

Lucas Piva Rocha Corrêa
Orientador: Carlos Eduardo Ferreira

Sumário

Parte I
Introdução

1 Introdução

Geometria Computacional é a área da computação que estuda algoritmos para resolução de problemas de natureza geométrica no computador, ou seja, em que os dados envolvem pontos (no plano ou no espaço), retas, segmentos de retas, polígonos, poliedros, etc...

Um desses problemas é o problema conhecido como *Problema da Galeria de Arte*, que motivou o projeto, por nos levar à necessidade do estudo de estruturas de dados e algoritmos interessantes para sua resolução. Nesse problema, queremos vigiar uma Galeria de Arte usando, por exemplo, câmeras de segurança.

Desse problema, surge naturalmente a necessidade de triangularizar polígonos, devido à prova do Teorema da Galeria de Arte dada por Fisk [8]. O tópico da triangularização por si só já apresenta inúmeros problemas e algoritmos.

Estendemos o conceito de triangularização para que a nossa entrada seja um conjunto de pontos no plano, e estudamos propriedades de uma triangularização particular, a *triangularização de Delaunay*, que possui diversas aplicações tanto para geometria computacional quanto computação gráfica. Também estudamos brevemente uma variação em que podemos inserir pontos artificialmente - chamados *pontos de Steiner* - no nosso conjunto original de pontos, visando melhorar a qualidade dos triângulos gerados, produzindo *triangularizações de Steiner*, muito usadas para geração de malhas.

Parte II
Parte Técnica

2 Triangularização de Polígonos

2.1 O Problema da Galeria de Arte

O problema da Galeria de Arte foi proposto por Victor Klee em 1973 [9] e pode ser formulado da seguinte maneira: imagine uma galeria de arte, que pode ser modelada como um polígono simples[1] de n vértices no plano. Quantas câmeras são necessárias para que toda a galeria seja monitorada? Dizemos que uma câmera fixada no ponto x monitora um ponto y se e somente se o segmento xy não passa por nenhum ponto exterior ao polígono. Veja figura 1.

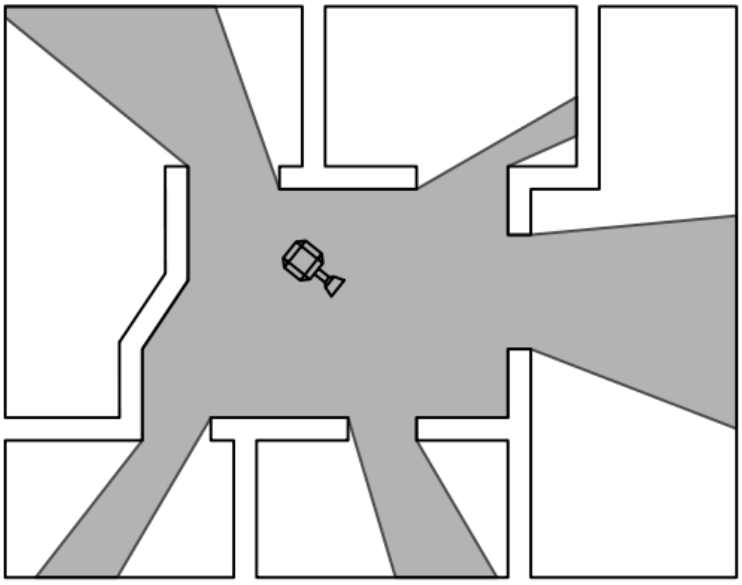


Figura 1: Uma galeria de arte modelada como um polígono no plano. A região sombreada representa a visibilidade da câmera (Fonte: Computational Geometry: Algorithms and Applications [2]).

Uma câmera é um ponto. Um conjunto de câmeras cobre o polígono se todo ponto do polígono é monitorado por alguma câmera. Uma variante interessante do problema seria exigir que apenas a fronteira do polígono seja coberta, onde geralmente ficam as obras de arte.

[1] Região do plano determinada por uma cadeia poligonal fechada que não se intersecta. Polígonos com buracos não são permitidos.

Definido o que é uma cobertura, precisamos agora definir precisamente o que queremos. No problema, queremos encontrar o máximo sobre todos os polígonos de n vértices, do mínimo número de câmeras necessárias para cobrir o polígono, ou seja, queremos como uma função de n, o menor número de câmeras suficientes para cobrir qualquer polígono de n vértices. Vamos definir formalmente o problema.

Seja $g(P)$ o menor número necessário para cobrir o polígono P: $g(P) = min_S|\{S : S \text{ cobre } P\}|$, onde S é um conjunto de pontos e $|S|$ é a cardinalidade[2] de S. Seja P_n um polígono com n vértices. $G(n)$ é o máximo de $g(P_n)$ sobre todos os polígonos de n vértices: $G(n) = max_{P_n} g(P_n)$. Queremos calcular $G(n)$. Dizemos que esse número é necessário e suficiente: Necessário no sentido de que precisamos de pelo menos esse número de câmeras para pelo menos um polígono e suficiente no sentido de que esse número é suficiente para qualquer polígono de n vértices.

Vamos mostrar, através de um exemplo, que $\lfloor n/3 \rfloor$ câmeras sempre são necessárias para um tipo de polígono especial. O polígono em forma de "pente" da figura 2 possui k pontas. Cada ponta possui duas arestas e duas pontas consecutivas são separadas por uma aresta. Se associarmos cada ponta com a aresta separadora da direita, e a última ponta com a aresta de baixo, vemos que o polígono possui $n = 3k$ arestas e conseqüentemente, vértices. Também podemos observar que uma câmera consegue enxergar no máximo uma dessas pontas, então para esse polígono especial, $n/3 \leq G(n)$, e assim achamos um limitante inferior para $G(n)$.

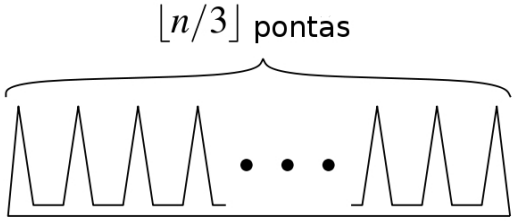


Figura 2: Pente de Chvátal. Precisamos de um guarda para cada ponta.

Vamos apresentar agora a Prova de Suficiência de Fisk de 1978 [8].

A prova depende em quebrar um polígono em triângulos pela adição de diagonais. Podemos decompor um polígono em triângulos pela adição de (zero ou mais) *diagonais*. Uma *diagonal* é um segmento aberto que liga dois vértices de um polígono, não intersecta nenhuma aresta, e está no interior desse polígono. A decomposição de um polígono em triângulos pela adição de um conjunto maximal de diagonais que não se intersectam é chamada de *triangularização* do polígono. Um polígono pode admitir

[2]A cardinalidade de um conjunto é o seu número de elementos

diferentes triangularizações. Vamos assumir que uma triangularização sempre existe (deixamos a prova para a seção 2.2).

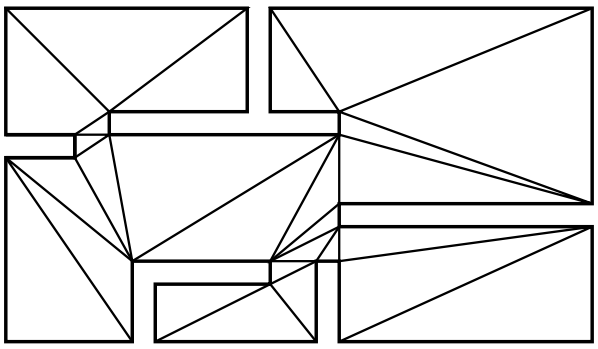


Figura 3: Um polígono simples, e uma possível triangularização.

Agora, seja $G = (V, E)$ o grafo associado a uma triangularização, onde os nós do grafo são precisamente os vértices do polígono, e os arcos de G são as arestas do polígono, mais as diagonais adicionadas na triangularização. Definimos uma *3-coloração* do grafo como uma atribuição de 3 cores aos nós do grafo, de forma que nenhum nó adjacente receba uma mesma cor. Veja a Figura 4. Novamente, assumimos que uma *3-coloração* sempre existe para o grafo associado a uma triangularização e deixamos a prova para a seção 2.2.

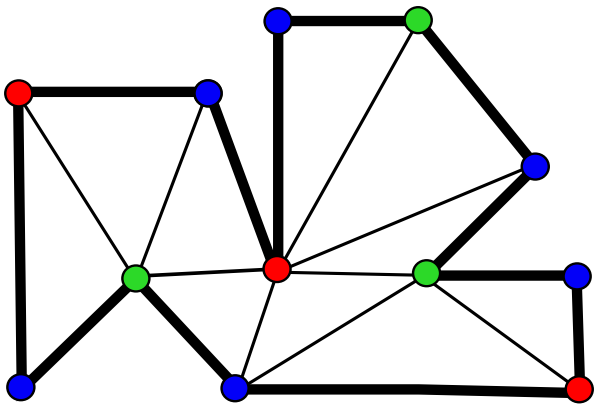


Figura 4: Um polígono e uma *3-coloração* do grafo associado à uma triangularização.

Seja $K = \{1, 2, 3\}$ as cores atribuídas. Cada triângulo possui um nó com cada uma das cores, caso contrário, estaríamos violando a propriedade da coloração. Notamos que colocar uma câmera em um dos vértices de um triângulo é suficiente para que ele seja coberto. Portanto, podemos arbitrariamente escolher a cor 1 e colocar uma câmera em todos os nós de cor 1. Dessa forma, conseguimos cobrir o polígono com $n/3$ câmeras.

Para concluir a prova, notamos que pelo menos uma das cores é usada no máximo $\lfloor n/3 \rfloor$ vezes, caso contrário, teríamos mais que n vértices. Portanto, podemos escolher essa cor para posicionar as câmeras.

Assim, com o limite inferior estabelecido pelo uso do "Pente de Chvátal" e pela prova de Suficiência de Fisk, concluímos nossa análise do problema da Galeria de Arte, com o resultado que $G(n) = \lfloor n/3 \rfloor$. Mas, como podemos triangularizar um polígono para posicionar as câmeras? Com esse interesse em vista, mudamos o foco para o assunto de triangularização de polígonos, apresentando, na seção 2.3 algoritmos para resolver o problema de forma eficiente.

2.2 Triangularização: Teoria

Antes de olhar para os algoritmos, precisamos estudar triangularização de polígonos. Vamos começar a seção provando que todo polígono admite uma triangularização.

Um dos pontos chave da prova é mostrar que todo polígono possui uma diagonal. Para essa prova, precisamos que todo polígono possua pelo menos um vértice estritamente convexo.

Lema 2.1. *Todo polígono possui pelo menos um vértice estritamente convexo.*

Demonstração. Seja v o vértice com menor coordenada y do polígono. Se houverem vários desses vértices, escolhemos o que possuir maior coordenada x. Assuma que nenhum dos vértices adjacentes está estritamente acima de v. Então, os dois vértices estão à esquerda de v, pelo jeito que escolhemos v, e possuem mesma coordenada y. Isso contraria a hipótese do polígono ser simples pois teríamos duas arestas sobrepostas. Assim, pelo menos um dos vértices adjacentes à v está estritamente acima dele e portanto o ângulo interior do vértice v é estritamente menor que π. □

Lema 2.2. *(Meisters) Todo polígono de $n \geq 4$ vértices possui uma diagonal.*

Demonstração. Seja v um vértice estritamente convexo, cuja existência é garantida pelo Lema 2.1 e sejam a e b os vértice adjacentes a v. Se $\overline{ab}$ for um segmento interno ao polígono, achamos uma diagonal. Se não, existe um ou mais vértices dentro do triângulo Δavb. Chame de v' o vértice mais distante de ab. O segmento $\overline{vv'}$ não pode intersectar nenhuma aresta do polígono, pois isso implicaria na existência de um vértice dentro do triângulo Δavb, mais distante de $\overline{ab}$ do que v', contrariando a escolha desse vértice. Portanto, $\overline{vv'}$ é uma diagonal. □

Teorema 2.3. *(Triangularização) Todo polígono P de n vértices pode ser particionado em triângulos pela adição de (zero ou mais) diagonais.*

Demonstração. A prova é feita por indução. Se $n = 3$, o polígono é um triângulo, e o teorema vale trivialmente. Seja $n \geq 4$. Seja $d = \overline{ab}$ uma diagonal de P, que existe, segundo o Lema 2.2. A diagonal divide o polígono em dois sub-polígonos P_1 e P_2 não vazios. Como o número de vértices de cada um dos sub-polígonos é menor que o número de vértices de P, aplicamos a indução nos dois sub-polígonos e juntamos P_1 e P_2 na diagonal d. Assim, temos que P admite uma triangularização. □

Lema 2.4. *Toda triangularização de um polígono P de n vértices usa $n-3$ diagonais e consiste de $n-2$ triângulos.*

Demonstração. Faremos a prova por indução. Para $n = 3$ as duas afirmações valem trivialmente. Seja $n \geq 4$. Toda diagonal d divide o polígono em dois sub-polígonos P_1 e P_2. Seja n_1 e n_2 o número de vértices de P_1 e P_2, respectivamente. Temos que $n_1 + n_2 = n + 2$ pois os vértices da diagonal são contados duas vezes, pois pertencem aos dois sub-polígonos. Aplicando a hipótese de indução para os sub-polígonos, temos $(n_1 - 3) + (n_2 - 3) + 1 = n - 3$ diagonais e $(n_1 - 2) + (n_2 - 2) = n - 2$ triângulos. □

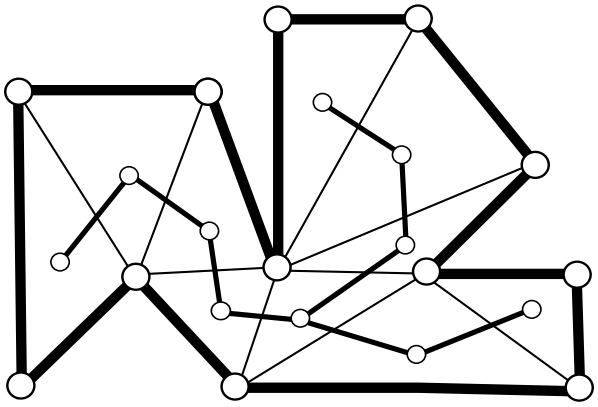


Figura 5: O grafo dual associado a uma triangularização de P.

Para provarmos que todo grafo associado a uma triangularização admite uma *3-coloração* definimos o *grafo dual* $G(T_p)$ como sendo um grafo onde temos um nó para cada triângulo e, para cada dois nós cujo triângulo associado divide uma diagonal, colocamos uma aresta.

Lema 2.5. *O dual $G(T_p)$ de uma triangularização é uma árvore*[3]*.*

Demonstração. Os arcos do grafo dual são justamente as diagonais adicionadas na triangularização. Como o polígono é simples, uma diagonal de P divide o polígono em dois componentes, portanto a remoção de um arco divide o grafo dual em dois componentes. Assim, $G(T_p)$ é uma árvore. □

[3]Grafo conexo acíclico

Com esse arsenal de resultados à disposição, podemos facilmente provar o Teorema das Duas Orelhas de Meisters de 1975 [10] e concluir a seção com a prova da *3-coloração*. Dizemos que três vértices consecutivos a, b, c formam uma orelha se o segmento $\overline{ac}$ é uma diagonal. Além disso, chamamos b de *ponta da orelha*. Duas orelhas são disjuntas se a intersecção do interior de seus triângulos for vazia.

Teorema 2.6. *(Teorema das duas orelhas de Meisters) Todo polígono com $n \geq 4$ vértices possui pelo menos duas orelhas disjuntas.*

Demonstração. Uma folha no grafo dual corresponde a uma orelha. Pelo resultado do Lema 2.4 a árvore possui mais que dois nós, e toda árvore de dois ou mais nós possui pelo menos duas folhas. □

Teorema 2.7. *(3-coloração) O grafo da triangularização de um polígono possui uma* 3-coloração

Demonstração. Provamos por indução no número de vértices. Um triângulo pode ser trivialmente colorido com 3 cores. Seja $n \geq 4$. Pelo Teorema 2.6, P possui uma orelha abc. Remova a orelha de P, ou seja, remova b, e aplique a hipótese de indução para o polígono de $n-1$ vértices resultante. Colocamos o vértice b de volta e atribuímos a ele a cor que não foi atribuída nem para a nem para c. □

2.3 Algoritmos

A seguir, apresentamos algoritmos para triangularização de polígonos. O primeiro, e mais intuitivo, roda em $O(n^4)$. Podemos facilmente usar o Teorema das duas orelhas de Meister's para reduzir a complexidade da triangularização para $O(n^3)$, e posteriormente para $O(n^2)$. Por último, iremos estudar o conceito de monotonicidade e apresentar um algoritmo para dividir um polígono em sub-polígonos monótonos em $O(n \log n)$ que, juntamente com o algoritmo para triangularização de polígonos monótonos apresentado na última seção, nos fornece um algoritmo para triangularização de polígonos em tempo $O(n \log n)$.

2.3.1 Triangularização por Adição de Diagonais

O primeiro algoritmo segue a prova da existência de uma triangularização (Teorema 2.3): Achamos uma diagonal qualquer, dividimos o polígono em duas partes, e resolvemos recursivamente para cada uma das partes.

Temos $\binom{n}{2} = O(n^2)$ candidatos a diagonais, e para cada uma delas, gastamos $O(n)$ para verificarmos se é uma diagonal válida. Faremos isso, segundo o Teorema 2.4, para $n-3$ diagonais, o que nos leva a concluir que o algoritmo roda em $O(n^4)$.

2.3.2 Triangularização por Remoção de Orelhas

Podemos, sem dificuldade, melhorar a complexidade do algoritmo da seção anterior usando o Teorema das Duas Orelhas de Meisters (Teorema 2.6). Sabemos que existe uma diagonal que separa uma orelha. Temos exatamente $n = O(n)$ candidatas para serem tais diagonais: (v_i, v_{i+2}) para $i = 0, ..., n-1$. Além disso, sempre dividimos o polígono em um triângulo (a orelha) e um polígono de $n-1$ vértices, precisando chamar a recursão apenas para o último. Assim, temos uma complexidade de $O(n^3)$.

Podemos explorar ainda mais a idéia da remoção de orelhas. Toda vez que adicionamos uma diagonal e removemos uma orelha do polígono, podemos notar que o polígono não muda muito. De fato, ao removermos uma orelha $E_2 = \Delta(v_1, v_2, v_3)$ os únicos vértices cujo estado de *ponta da orelha* podem mudar são os vértices v_1 e v_3. Vamos analisar cuidadosamente essa situação a seguir.

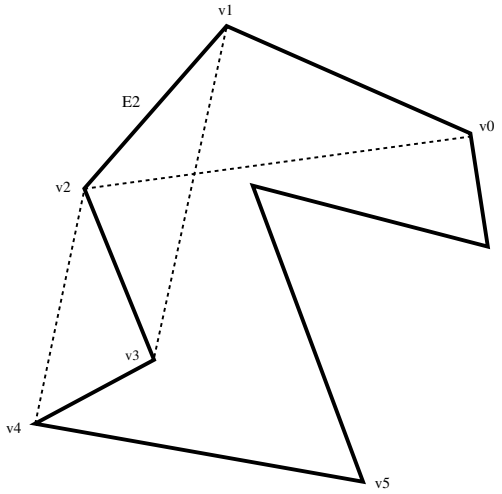


Figura 6: Removendo a orelha $E_2 = \Delta(v_1, v_2, v_3)$. O vértice v_1 deixa de ser uma ponta da orelha

Veja a figura 6. O status de ponta da orelha do vértice v_4 depende apenas da diagonal v_3v_5. A remoção da orelha $E2$ não altera os vértices v_3 ou v_5. Assim, os únicos vértices cujo status pode mudar, são os vértices v_1 e v_3. Portanto, ao removermos uma orelha, basta atualizarmos o status de dois vértices, com um número constante de operações.

Assim, temos o seguinte algoritmo para triangularização de um polígono:

TRIANGULARIZAÇÃO(P)

```
1  D ← ∅ ▷ Conjunto de diagonais adicionadas
2  Inicialize o status de ponta de orelha
   para cada vértice em P
3  enquanto n > 3
4      faça
5          Localize uma ponta de uma orelha v2
6          Adicione v1v3 ao conjunto de diagonais D
7          Remova v2 de P
8          Atualize o status de orelha dos vértices v1 e v3
9
10 devolva D
```

2.3.3 Monotonização

Nessa seção, vamos precisar de algumas definições. O conceito de monotonicidade foi introduzido por Lee e Preparata, 1977 [9].

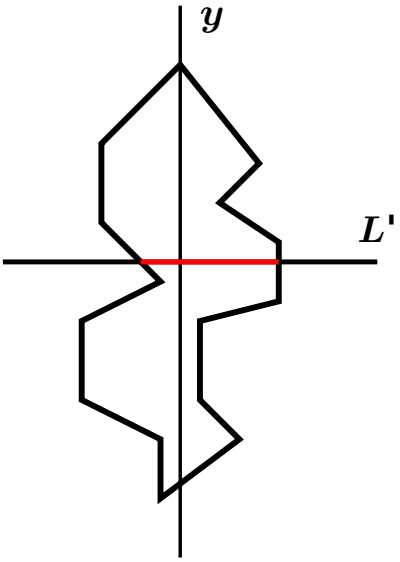


Figura 7: Um polígono y-monótono

*Definição (*Monotonicidade*):* Um polígono P é dito monótono em relação a uma linha l se, para toda linha l' perpendicular a l, a intersecção de l' com P é conexa, ou seja, é um ponto, um segmento de reta, ou vazio.

Estaremos interessados em polígonos monótonos em relação ao eixo y, que chamaremos de *y-monótonos*. Uma propriedade interessante desses polígonos é que se andarmos do vértice mais alto (com maior coordenada y) para o vértice mais baixo (menor coordenada y), pela fronteira esquerda (ou direita), sempre estaremos andando para baixo, ou horizontalmente, mas nunca para cima. Nossa estratégia para triangularizar

um polígono será primeiro dividi-lo em polígonos menores, *y-monótonos* e depois triangularizar cada um desses polígonos.

Vamos analisar e classificar os tipos de vértices de um polígono qualquer em 5 categorias, como visto na Figura 8.

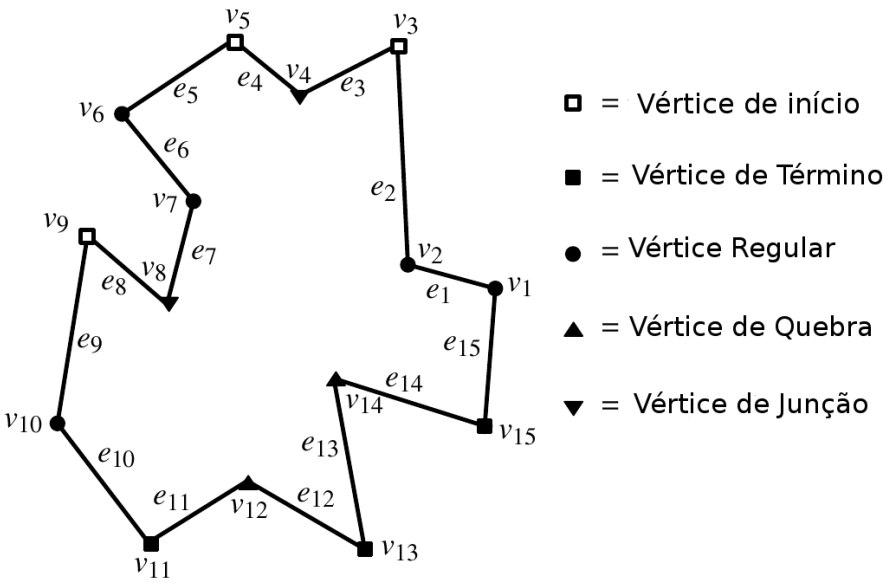


Figura 8: Tipos de vértices de um polígono (Fonte: Computational Geometry: Algorithms and Applications [2])

Um vértice v pertence a um dos tipos definidos a seguir:

1. Vértice de Início - Os dois vértices adjacentes estão abaixo de v e o ângulo interno no vértice é menor que π.

2. Vértice de Quebra - Os dois vértices adjacentes estão abaixo de v e o ângulo interno no vértice é maior que π.

3. Vértice de Término - Os dois vértices adjacentes estão acima de v e o ângulo interno no vértice é menor que π.

4. Vértice de Junção - Os dois vértices adjacentes estão acima de v e o ângulo interno no vértice é maior que π.

5. Vértice Regular - Vértice que não é um vértice de início, quebra, término ou junção.

Os vértices de quebra e junção são fontes de não-monotonicidade local. Vamos enunciar isso formalmente em um lema:

Lema 2.8. *Um polígono é* y-monótono *se não possui nenhum vértice de quebra, e nenhum vértice de junção.*

Demonstração. Vamos supor que P é não *y-monótono* e provar que ele possui ou um vértice de quebra ou um vértice de junção.

Pela definição de monotonicidade, se P não é monótono, existe uma linha horizontal r que cruza o polígono em duas componentes distintas. Sem perda de generalidade, assumimos que a componente mais a esquerda que intersecta r é um segmento. Seja p o ponto mais à esquerda e q o ponto mais à direita. À partir de q, andamos pela borda de P de forma que o interior do polígono esteja sempre à esquerda. Da forma que escolhemos r, p e q, esse passeio é para cima. Em algum ponto, chamemos de s, temos que cruzar novamente com r. Temos dois casos: No primeiro, $s \neq p$ e o vértice mais alto que encontramos no passeio é um vértice de quebra, como podemos ver na Figura 9(a).

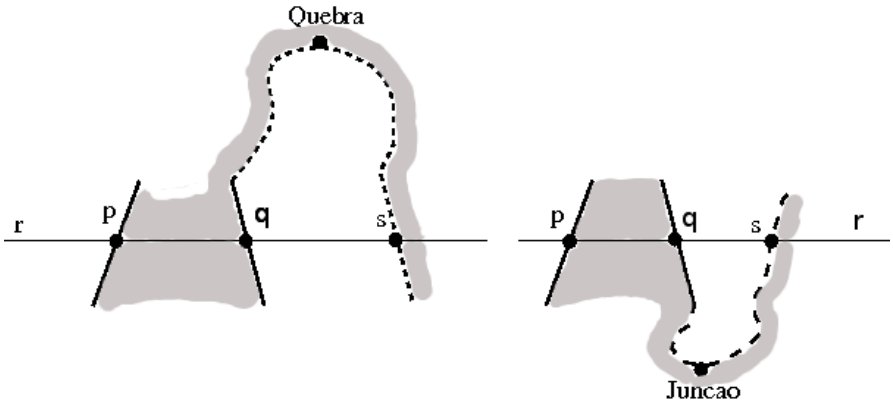


Figura 9: Dois casos na prova do Lema 2.8

No caso em que $s = p$, novamente percorremos a borda de P a partir de q, mas no outro sentido. Novamente, temos que cruzar com r em algum ponto s'. Mas $s' \neq p$, caso contrário estaríamos contradizendo a não-monotonicidade do polígono e a escolha de r. Agora, o ponto mais baixo que encontramos no passeio é um vértice de junção. Como na Figura 9(b). □

O Lema 2.8 implica que para monotonizarmos um polígono, basta nos livrarmos dos vértices de quebra e de junção.

Para nos livrarmos dos vértices de quebra e de junção, vamos utilizar um algoritmo de linha de varredura. O algoritmo move um linha horizontal que vai varrendo o plano, de cima para baixo, parando em eventos importantes. No nosso caso, esses eventos ocorrem quando a varredura encontrar um vértice novo. Podemos pré-processar o nosso polígono e construir uma fila Q onde os pontos sejam ordenados pela coordenada y. O ponto de maior coordenada y será o primeiro evento, e, analogamente, o ponto de menor coordenada y será o último evento.

Vamos ver o que fazer quando encontramos um vértice v_i de quebra. Evidentemente, queremos adicionar uma diagonal ligando v_i à um vértice

acima. Seja e_j a aresta imediatamente à esquerda de v_i e e_k a aresta imediatamente à direita de v_i na linha de varredura. Podemos conectar o vértice de quebra com o vértice mais próximo, que está entre e_j e e_k. Se não houver tal vértice, podemos ligar v_i com o ponto da extremidade de e_j ou e_k. Definimos o *ajudante(e_j)* como sendo o vértice mais abaixo acima da linha de varredura, tal que o segmento horizontal do vértice a e_j está dentro do polígono.

Agora, temos uma intuição de como lidar com os vértices de junção, afinal, eles são bem parecidos com os vértices de quebra. Quando chegamos em um vértice v_i de junção, ainda não sabemos com qual vértice devemos ligá-lo, mas, nesse momento, v_i se torna o ajudante de e_j, a aresta imediatamente à esquerda de v_i. Queremos então achar o vértice mais alto abaixo de v_i, que esteja entre as arestas e_j e e_k, as arestas à esquerda e à direita de v_i, respectivamente. Esse vértice será justamente o vértice que vai substituir v_i como ajudante de e_j. Então, sempre que substituirmos o ajudante de uma aresta e o ajudante anterior for um vértice de junção, adicionamos uma diagonal. É possível que o ajudante não seja substituído e nesse caso, ligamos v_i com a extremidade inferior de e_j.

Para o algoritmo, precisamos achar rapidamente a aresta à esquerda de um vértice. Assim, podemos construir uma árvore de busca binária dinâmica que guarda, a cada evento, as arestas que cruzam a linha de varredura, ordenadamente (nas folhas, da esquerda para direita, por exemplo). Com cada aresta, guardamos também seu ajudante. A estrutura da árvore, assim como os ajudantes, mudam conforme a linha de varredura se desloca verticalmente no plano, novas arestas começam a intersectar a linha, e arestas deixam de intersectar essa linha.

Podemos então escrever um algoritmo que será explicado com um pouco mais de detalhe.

```
MONOTONIZAÇÃO(P)
 1  Construa uma fila de prioridade Q com os
    vértices de P, usando sua coordenada y.
 2  T ← ∅ ▷ Árvore de Busca começa vazia
 3  enquanto Q não está vazia
 4      faça
 5          Remova o vértice v_i de maior
            prioridade de Q
 6          se v_i é vértice de início
 7            então VÉRTICEINÍCIO(P, T, v_i)
 8            senão se v_i é vértice de término
 9                    então VÉRTICETÉRMINO(P, T, v_i)
10            senão se v_i é vértice de quebra
11                    então VÉRTICEQUEBRA(P, T, v_i)
12            senão se v_i é vértice de junção
13                    então VÉRTICEJUNÇÃO(P, T, v_i)
14            senão se v_i é vértice regular
15                    então VÉRTICEREGULAR(P, T, v_i)
```

Vamos ver em detalhe o que fazer para cada tipo de vértice.

Se v_i for um vértice de início, basta adicionar e_i a T marcar v_i como ajudante de e_i.

```
VÉRTICEINÍCIO(P, T, v_i)
1  T ← T ∪ {e_i}
2  ajudante(e_i) ← v_i
```

Se v_i for um vértice de término precisamos verificar se o ajudante da aresta e_{i-1} com extremidade inferior v_i era um vértice de junção, e adicionar uma diagonal ligando v_i ao vértice de junção, se for o caso. Por fim, removemos a aresta de T.

```
VÉRTICETÉRMINO(P, T, v_i)
1  se ajudante(e_{i-1}) é um vértice de junção
2      então
3            Adicione uma diagonal de v_i para ajudante(e_{i-1})
4  T ← T − {e_{i-1}}
```

Se v_i for um vértice de quebra, precisamos achar a aresta e_j diretamente à esquerda de v_i. Inserimos a diagonal ligando v_i ao $ajudante(e_j)$. Depois, dizemos que o ajudante de e_j agora é v_i, inserimos e_i em T sendo v_i seu ajudante.

VÉRTICEQUEBRA(P, T, v_i)

1 Seja e_j a aresta diretamente à esquerda de v_i
2 Adicione uma diagonal de v_i para $ajudante(e_j)$
3 $ajudante(e_j) \leftarrow v_i$
4 $T \leftarrow T \cup \{e_i\}$
5 $ajudante(e_i) \leftarrow v_i$

Se v_i for um vértice de junção, primeiro precisamos verificar se o ajudante anterior da aresta adjacente e_{i-1} era um vértice de junção. Nesse caso, adicionamos uma diagonal de v_i para $ajudante(e_{i-1})$. Removemos e_{i-1} de T pois a aresta pára de intersectar a linha de varredura, e procuramos em T por e_j, a aresta diretamente à esquerda de v_i. Se o ajudante de e_j for novamente um vértice de junção, adicionamos outra diagonal entre eles. Por fim, trocamos o $ajudante(e_j)$ para v_i.

VÉRTICEJUNÇÃO(P, T, v_i)

1 **se** $ajudante(e_{i-1})$ é um vértice de junção
2 **então**
3 Adicione uma diagonal de v_i para $ajudante(e_{i-1})$
4 $T \leftarrow T - \{e_{i-1}\}$
5 Seja e_j a aresta diretamente à esquerda de v_i
6 **se** $ajudante(e_j)$ é um vértice de junção
7 **então**
8 Adicione uma diagonal de v_i para $ajudante(e_j)$
9 $ajudante(e_j) \leftarrow v_i$

Por fim, falta apenas tratar o caso de v_i ser um vértice regular. Diferenciamos dois casos. No primeiro, o polígono P está à direita de v_i. Nesse caso, verificamos se $ajudante(e_{i-1})$ é um vértice de junção e adicionamos uma diagonal caso ele seja. Depois deletamos e_{i-1} de T, e inserimos e_i com v_i como seu ajudante. Se o polígono está à esquerda de v_i, procuramos em T pela aresta diretamente à esquerda de v_i. Como esperado, verificamos se $ajudante(e_j)$ é um vértice de junção e adicionamos uma diagonal se for. Por último, marcamos v_i como o ajudante de e_j.

VÉRTICEREGULAR(P, T, v_i)

```
1  se P está a direita de v_i
2     então
3           se ajudante(e_{i-1}) é um vértice de junção
4              então
5                    Adicione uma diagonal de v_i para ajudante(e_{i-1})
6           T ← T − {e_{i-1}}
7           T ← T ∪ {e_i}
8           ajudante(e_i) ← v_i
9     senão
10          Seja e_j a aresta diretamente à esquerda de v_i
11          se ajudante(e_j) é um vértice de junção
12             então
13                   Adicione uma diagonal de v_i para ajudante(e_j)
14          ajudante(e_j) ← v_i
```

Assim, sempre adicionamos uma diagonal quando encontramos um vértice de quebra, e sempre que atualizamos o ajudante de uma aresta cujo ajudante anterior era um vértice de junção. O algoritmo descrito adiciona um conjunto de diagonais que não se intersectam e particionam o polígono P em sub-polígonos monótonos.

Lema 2.9. *O algoritmo* MONOTONIZAÇÃO *remove todos os vértices de quebra e todos os vértices de junção pela adição de diagonais.*

Demonstração. Primeiro, observamos que se adicionarmos uma diagonal de um vértice de quebra para um vértice de maior y-coordenada, estaremos dividindo o vértice de quebra em dois ângulos menores que π. O mesmo vale se adicionarmos uma diagonal de um vértice de junção para um vértice com menor y-coordenada.

Para os vértices de quebra, basta notar que sempre adicionais uma diagonal para um vértice com maior coordenada y quando processamos um vértice de quebra, já que o ajudante da aresta à esquerda necessariamente está acima.

Vamos mostrar que sempre adicionamos pelo menos uma diagonal de um vértice de junção para um vértice de menor coordenada y. Assim que olhamos um vértice de junção v_i, ele se torna o ajudante da aresta e_j imediatamente à esquerda. Considere agora o momento em que atualizamos o ajudante de e_j. Podemos trocar v_i por um vértice regular, de quebra ou de junção. Nos três casos, verificamos se o ajudante anterior era um vértice de junção e adicionamos uma diagonal. No entanto, pode ser que o ajudante de e_j não seja atualizado. Nesse caso, quando tratamos o vértice inferior da aresta e_j, que só pode ser um vértice de junção, regular ou término, verificamos se o ajudante da aresta que está saindo da linha de varredura era um vértice de junção e adicionamos uma diagonal. □

Lema 2.10. *O algoritmo* MONOTONIZAÇÃO *adiciona um conjunto de diagonais não intersectantes.*

Demonstração. As rotinas VÉRTICETÉRMINO, VÉRTICEQUEBRA, VÉRTICEJUNÇÃO e VÉRTICEREGULAR podem adicionar diagonais. Vamos mostrar que as diagonais adicionadas pela rotina VÉRTICEQUEBRA não intersectam nenhuma outra diagonal, e nenhuma aresta do polígono. A prova para as outras rotinas é análoga.

Seja $\overline{v_i v_p}$ o segmento adicionado quando processamos o vértice de quebra v_i e seja e_i e e_j a aresta imediatamente à esquerda e direita de v_i, respectivamente, como na figura 10.

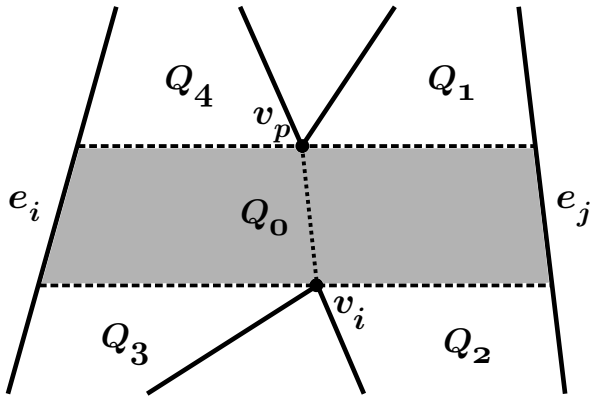


Figura 10: Prova do lema 2.10

Considere a região Q_0 como sendo a região fechada pelas arestas e_i e e_j e pelas arestas horizontais que passam por v_i e v_p. Como adicionamos o segmento $\overline{v_i v_p}$, v_p necessariamente é o *ajudante* de e_i e não existe nenhum ponto na região Q_0, caso contrário ele seria o *ajudante* de e_i.

Primeiro, mostramos que o segmento adicionado não intersecta nenhuma aresta do polígono. Suponha que exista uma aresta e do polígono que intersecte com $\overline{v_i v_p}$. Como não há nenhum ponto em Q_0, os vértices de e teriam que estar um em Q_1 e outro em Q_3 ou um em Q_2 e outro em Q_4. Nos dois casos, estaríamos violando a propriedade de e_i ou e_j como aresta imediatamente à esquerda e direita de v_i.

Considere agora uma diagonal d adicionada anteriormente. Como nosso algoritmo adiciona diagonais apenas entre vértices já processados, os dois vértices dessa diagonal possuem maior y coordenada que v_i. Novamente lembramos que Q_0 não possui nenhum vértice, então os dois vértices de d estão acima de v_p e portanto não podem intersectar o segmento $\overline{v_i v_p}$.

□

Teorema 2.11. *O algoritmo* MONOTONIZAÇÃO *particiona um polígono simples de n vértices em componentes monótonas pela adição de diagonais não intersectantes, em tempo $O(n \log n)$ e armazenamento $O(n)$.*

Demonstração. Com os lemas 2.9 e 2.10, precisamos apenas mostrar que ele consome tempo a $O(n \log n)$ e armazenamento $O(n)$. A estrutura Q pode ser inicializada em tempo $O(n \log n)$ pela ordenação dos vértices. O laço da linha 3 consome $O(n)$, pois nenhum vértice é inserido em Q e removemos um vértice por iteração. A cada iteração, realizamos uma operação na estrutura Q, adicionamos no máximo duas diagonais e fazemos um número constante de operações de inserção, remoção e busca na estrutura T. As operações na fila Q e a adição de diagonais são realizadas em $O(1)$. Usando uma estrutura como uma árvore balanceada para representar a ordenação das arestas que cruzam a linha de varredura, podemos realizar todas as operações em T em tempo $O(\log n)$. Assim, o algoritmo consome tempo $O(n \log n)$.

Quanto ao armazenamento, cada vértice aparece exatamente uma vez na fila Q e cada aresta aparece no máximo uma vez na estrutura T. Claramente, ocupamos espaço proporcional a $O(n)$. □

2.3.4 Triangularização de polígonos y-monótonos

Agora que sabemos como monotonizar polígonos, queremos uma forma eficiente de triangularizá-los. Inicialmente, assumimos que os polígonos são estritamente y-monótonos, ou seja, não existem arestas paralelas ao eixo x. Dessa forma, um passeio do vértice com maior coordenada y ao vértice com menor coordenada y, pela fronteira esquerda ou direita, sempre andará para baixo, nunca para o lado. Isso nos permite ordenar os vértices pela coordenada y sem que haja empate entre dois vértices na mesma fronteira.

Processamos os vértices em ordem decrescente em relação à coordenada y. Se houver empate, escolhemos tratar o de menor coordenada x primeiro. Usamos uma pilha S para armazenar os vértices do polígono P que já foram processados mas talvez ainda precisem de diagonais. Conforme processamos os vértices, as partes já triangularizadas de P vão sendo removidas de S. O vértice no topo da pilha é o vértice sendo analisado no momento, de menor coordenada y, o segundo vértice da pilha é o de segunda menor coordenada y e assim por diante. Quando processamos um vértice, tentamos adicionar todas as diagonais que conseguimos para os vértices que estão na pilha.

Com essa estratégia, a parte não triangularizada de P que está em S sempre apresenta um formato particular, de um funil ao contrário. O último vértice da pilha (de maior coordenada y) é um vértice convexo (ângulo interno menor que 180^o), um dos lados do funil é uma aresta e o outro é uma seqüência de vértices de reflexo (ângulo interno maior ou igual a 180^o), conforme a figura 11. Vamos ver em mais detalhes como adicionar essas diagonais. Separamos dois casos, o primeiro quando v_j, o próximo vértice a ser analisado, encontra-se na cadeia oposta do vértice do topo da pilha, e o segundo quando v_j encontra-se na mesma cadeia

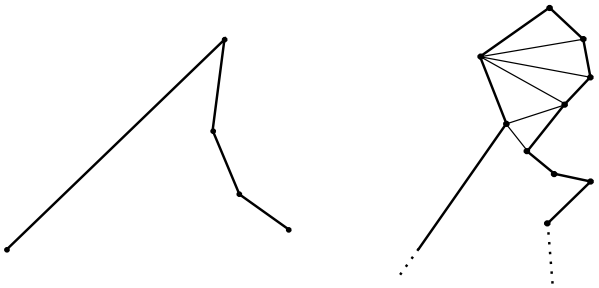


Figura 11: Na esquerda, o formato invariante do funil. A figura da direita mostra uma triangularização parcial, onde os triângulos produzidos são "jogados fora".

do vértice no topo da pilha.

No primeiro caso, v_j é o ponto inferior da aresta do funil. Devido ao formato do funil, podemos adicionar diagonais de v_j para todos os vértices na pilha, com exceção do último, que já está ligado a v_j por uma aresta do polígono. Todos esses vértices são desempilhados. A parte ainda não triangularizada possui todos os vértices não analisados, v_j e o vértice que estava no topo da pilha e é limitada pela diagonal que liga esses dois vértices preservando o formato do funil, e por isso, colocamos eles de volta na pilha, como indicado na figura 12.

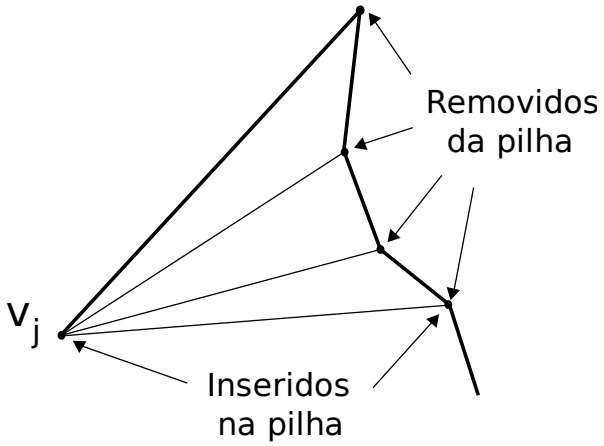


Figura 12: Vértice sendo analisado encontra-se na cadeia oposta

No segundo caso, v_j está na mesma cadeia que o vértice anteriormente no topo da pilha. Diferente do primeiro caso, nem sempre conseguimos adicionar diagonais para todos os vértices da pilha, mas para os vértices que conseguimos, eles são todos consecutivos. Basta remover o vértice do topo da pilha (que já é um vizinho de v_j), e ir adicionando diagonais para os vértices da pilha até não conseguirmos mais. Podemos verificar se uma aresta de v_j para um vértice v_k da pilha é uma diagonal válida olhando apenas para esses dois vértices e o último vértice removido da pilha.

Colocamos o último vértice que conseguimos adicionar uma diagonal de volta na pilha (ou o vizinho de v_j caso nenhuma diagonal tenha sido adicionada). Colocamo v_j na pilha e assim preservamos o invariante: um dos lados do funil continua igual, a aresta. O outro consiste apenas de vértices de reflexo, caso contrário teríamos adicionado mais uma diagonal no processo. Veja a figura 13.

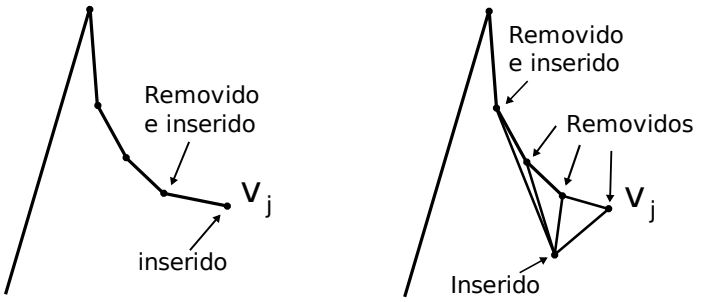


Figura 13: Dois casos quando vértice sendo analisado encontra-se na mesma cadeia

Isso nos leva a um algoritmo para triangularização de polígonos monótonos.

TriangularizaPolígonoMonótono(P)

```
1  Faça um merge dos vértices da cadeia esquerda
   com os da cadeia direita, ordenados decrescentemente
   pela coordenada y.
   Seja u_1, u_2, ..., u_n essa sequência de vértices.
2  Inicialize uma pilha S vazia e coloque u_1 e u_2 em S
3  para j ← 3 até n − 1
4      faça
5          se u_j e o vértice no topo de S estão em cadeias diferentes
6             então
7                 Remova todos os vértices de S.
8                 Insira uma diagonal de u_j para cada vértice
                  removido de S, a menos do último.
9                 Coloque u_{j−1} e u_j em S
10            senão
11                Remova um vértice de S
12                Remova os outros vértices de S enquanto é
                  possível traçar uma diagonal do vértice para
                  u_j e insira cada uma dessas diagonais.
13                Coloque o último vértice removido de volta em
                  S e também u_j.
14 Adicione diagonais de u_n para todos os vértices da pilha,
   com exceção do primeiro e último.
```

Vamos agora analisar quanto o algoritmo consome. O primeiro e segundo passo consomem, respectivamente, tempo linear e tempo constante. O laço do terceiro passo é executado $n-3$ vezes e cada execução pode realizar $O(n)$ operações. No entanto, a cada iteração do laço, no máximo dois vértices são colocados na pilha. Assim, o número de inserções na pilha, incluindo os do segundo passo, está limitado por $2n-4$. Como o número de remoções não pode ultrapassar o número de inserções, o tempo total de execução do laço é linear. No total, o algoritmo consome tempo $O(n)$.

3 Triangularizações de Pontos no Plano

Primeiramente, vamos definir uma triangularização de um conjunto de pontos no plano: Seja $P = p_1, p_2, ..., p_n$ um conjunto de pontos no plano. Uma subdivisão planar maximal é uma subdivisão S tal que a adição de uma aresta quebra a propriedade da planaridade. Ou seja, toda aresta que não está em S intersecta alguma aresta em S. Então uma triangularização de P é justamente uma subdivisão planar maximal cujo conjunto de vértices é precisamente P. O nosso interesse é estudar essa triangularização.

Uma aplicação é a seguinte: Superfícies terrestres podem ser modeladas como *terrenos*. Podemos definir um terreno como um conjunto de pontos em um plano R^2 e uma função *altura* $f : R^2 \rightarrow R$ que mapeia para cada ponto uma altura. Intuitivamente, não possuímos informação sobre a altura de todos os pontos de uma determinada superfície, ou seja, nós temos o valor da função f para um conjunto finito $P \subset R^2$ de pontos. Com essa amostra, queremos aproximar a altura de todos os pontos do plano de alguma maneira.

Uma das formas mais simples de fazer tal aproximação é associar a cada ponto a altura do ponto mais próximo da amostra. Usando essa abordagem teremos um terreno discreto, que geralmente não corresponde à superfície real. Uma abordagem mais razoável é triangularizar o conjunto de pontos da amostra e elevar cada ponto à sua altura dada pela função f , mapeando assim cada ponto a um triângulo. Veja figura 14.

Ainda sim, não sabemos como triangularizar o conjunto de pontos. Além disso, o conjunto de pontos pode ser triangularizado de várias maneiras, e delas, precisamos decidir qual serve melhor o nosso propósito, o de aproximar um terreno. Apesar de todas as triangularizações mapearem corretamente os pontos da amostra e parecerem iguais, podemos ter uma intuição boa ao olhar para as duas triangularizações do mesmo conjunto de pontos da figura 15. Pela altura dos pontos, o terreno parece representar o cume de uma montanha. Isso parece se confirmar pela triangularização (a), da esquerda. No entanto, se olharmos para a triangularização (b), da direita, temos um vale entre o cume de duas

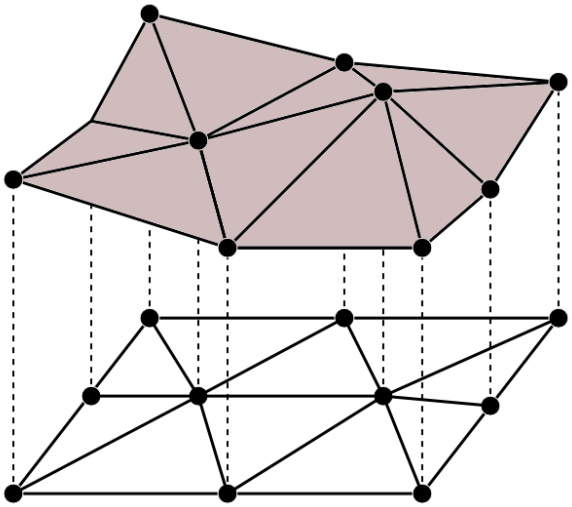


Figura 14: Um terreno poliédrico.

montanhas. Isso parece representar incorretamente o terreno. Se analisarmos a triangularização (b) podemos ver que isso acontece pois a altura do ponto q é determinada por pontos que estão relativamente distantes. Isso acontece pois q está em uma aresta de dois triângulos finos e compridos. Intuitivamente, uma triangularização que possui triângulos finos, i.e. com ângulos muito pequenos, parece ser pior.

Podemos então definir a triangularização procurada da seguinte forma: De todas as triangularizações possíveis do conjunto de pontos, queremos a que maximiza o menor ângulo dentre todos os ângulos de todos os triângulos. Se duas triangularizações distintas possuem o menor ângulo igual, olhamos para o segundo menor ângulo, e assim por diante.

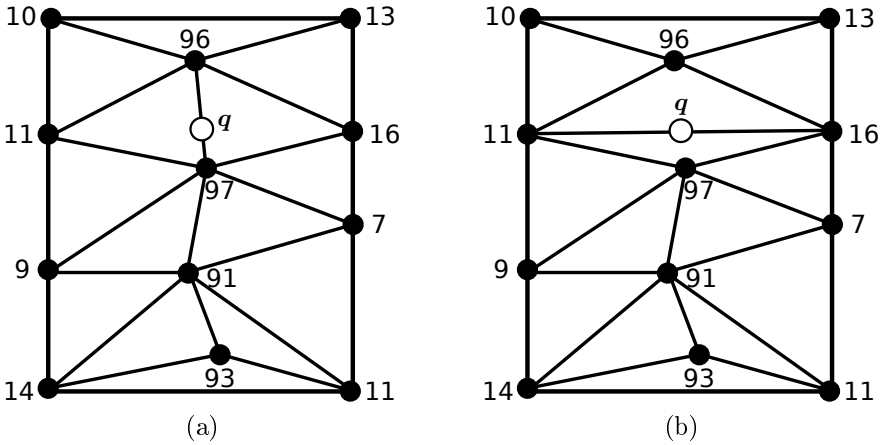


Figura 15: A diferença entre uma troca

Formalmente, podemos definir a triangularização procurada da seguinte forma: Seja T uma triangularização do conjunto de pontos P, e suponha que possua m triângulos. Agora, ordenamos os $3m$ ângulos de T em ordem crescente. Seja $A(T) := (\alpha_1, \alpha_2, ..., \alpha_{3m})$ a sequência ordenada, $\alpha_i \leq \alpha_j$ para todo $i < j$. Chamamos $A(T)$ o vetor-ângulo de T. Seja T' outra triangularização de P com vetor-ângulo $A(T') := (\alpha'_1, \alpha'_2, ...\alpha'_{3m})$. Dizemos que o vetor-ângulo de T é maior que o vetor-ângulo de T' se para algum índice i com $1 \leq i \leq 3m$, temos

$$\alpha_j = \alpha'_j \text{ para todo } j < i, \text{ e } \alpha_i > \alpha' i.$$

Em outras palavras, no primeiro índice i em que α_i e α'_i diferem, $\alpha_i > \alpha'_i$. Escrevemos $A(T) > A(T')$. Uma triangularização T é chamada *ângulo-ótima* se $A(T) \geq A(T')$ para toda triangularização T' de P.

Agora que temos o nosso problema em mente, podemos olhar para algumas propriedades de uma triangularização. Uma observação importante é que assumimos que os pontos nunca são todos colineares.

Lema 3.1. *Uma triangularização de pontos no plano sempre existe e consiste apenas de triângulos.*

Demonstração. Pela forma como definimos uma triangularização de pontos no plano, é trivial ver que ela sempre existe. Suponha por contradição que ela não seja formada apenas por triângulos. Então temos um polígono que possui uma diagonal, e então poderíamos adicionar essa diagonal sem quebrar a propriedade da planaridade. □

Vamos mostrar que o número de triângulos de uma triangularização depende do número de vértices do conjunto e do número de vértices no fecho convexo. O fecho convexo de um conjunto de pontos P é um polígono convexo cujos vértices são pontos de P e que contém todos os pontos de P.

Teorema 3.2. *Seja P um conjunto de pontos no plano com mais de 2 pontos e seja k o número de pontos que estão no fecho convexo de P. Então toda triangularização de P contém exatamente $2n-2-k$ triângulos e $3n-3-k$ arestas.*

Demonstração. Vamos fazer a prova por indução em n. Para $n = 3$, o fecho convexo é um triângulo, então $k = 3$ e $2n - 2 - k = 1$. Para $n > 3$, removemos um ponto p do interior do fecho convexo, e pela hipótese de indução, podemos triangularizar $P - p$ com $2(n-1) - 2 - k$ triângulos. Colocamos p de volta, e podemos triangularizar P pela adição de novas arestas. Se p estiver no interior de um triângulo, adicionamos 3 arestas de p aos vértices do triângulo. Se p estiver contido no lado de dois triângulos, adicionamos uma aresta de p para o vértice oposto de cada

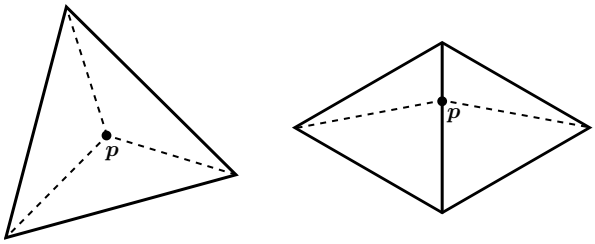


Figura 16: Em ambos os casos, adicionamos dois novos triângulos.

triângulo, conforme a figura 16. Nos dois casos, adicionamos 2 triângulos, então o número de triângulos é dado por $2(n-1)-2-k+2 = 2n-2-k$.

No entanto, pode ser que todos os pontos encontrem-se no fecho convexo. Nesse caso, removemos um ponto p do fecho convexo e pela hipótese de indução, podemos triangularizar o conjunto resultante com $2(n-1)-2-(k-1)$ triângulos, pois não havia pontos dentro do fecho. Colocando o ponto de volta, adicionamos um triângulo (uma orelha) trivialmente como na figura 17, e temos $2(n-1)-2-(k-1)+1 = 2n-2-k$ triângulos.

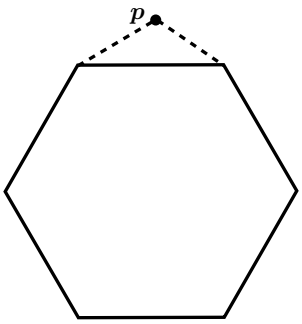


Figura 17: Colocamos o vértice de volta e adicionamos as arestas tracejadas, formando um novo triângulo.

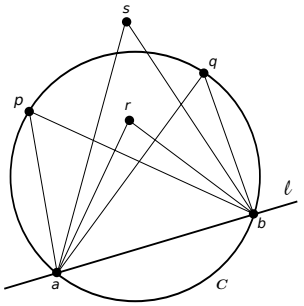
Como temos $2n-2-k$ triângulos, e cada triângulo possui exatamente 3 arestas, $3(2n-2-k)$ conta cada aresta de um triângulo duas vezes, pois uma aresta é sempre compartilhada por dois triângulos, a menos das arestas do fecho convexo, que são contadas apenas uma vez. Como o número de arestas no fecho convexo é igual ao número de vértices no fecho convexo, então o número de arestas é dado por

$$\frac{3(2n-2-k)+k}{2} = \frac{6n-6-3k+k}{2} = 3n-3-k$$

□

A seguir, vamos estudar quando uma triangularização é ângulo-ótima. O seguinte resultado de geometria elementar será útil.

Teorema 3.3. *Seja C um círculo, l uma reta que intersecta C nos pontos a e b. Sejam p, q, r e s pontos distintos do mesmo lado da reta, tal que p e q estejam na borda de C, r esteja fora de C e s dentro de C, como na figura 3.3.*



Então

$$a\hat{r}b > a\hat{p}b = a\hat{q}b > a\hat{s}b.$$

Considere uma aresta $e = \overline{p_i p_j}$ de uma triangularização T de P. Se e não for uma aresta da fronteira, então ela é compartilhada por dois triângulos, $\Delta p_i p_j p_k$ e $\Delta p_i p_j p_l$. Além disso, se os dois triângulos compartilhados formam um quadrilátero convexo, podemos trocar a aresta $\overline{p_i p_j}$ pela aresta $\overline{p_k p_l}$, obtendo assim uma nova triangularização. Chamamos essa operação de *troca de aresta.*

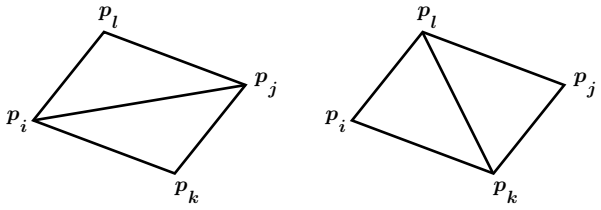


Figura 18: As duas diferentes triangularizações obtidas através de uma troca de aresta

Seja α_i o menor ângulo dos triângulos $\Delta p_i p_j p_k$ e $\Delta p_i p_j p_l$ e seja α_i' o menor ângulo dos triângulos $\Delta p_k p_l p_i$ e $\Delta p_k p_l p_j$. Dizemos que $\overline{p_i p_j}$ é uma *aresta ilegal* se $\alpha_i < \alpha_i'$. Analogamente, $\overline{p_k p_l}$ é uma *aresta ilegal* se $\alpha_i' < \alpha_i$.

Em outras palavras, uma aresta é ilegal se podemos aumentar localmente o menor ângulo através de uma troca de aresta. Evidentemente, se temos uma triangularização T com uma aresta ilegal, podemos obter uma nova triangularização T' através de uma troca de aresta, tal que $A(T') > A(T)$. Através do seguinte lema, que segue do teorema 3.3, podemos facilmente verificar se uma aresta é ilegal.

Lema 3.4. *Seja $\overline{p_ip_j}$ uma aresta dos triângulos $\Delta p_ip_jp_k$ e $\Delta p_ip_jp_l$ e C o círculo que passa por p_i, p_j e p_k. A aresta $\overline{p_ip_j}$ é ilegal se e somente se o ponto p_l está no interior do círculo C. Além disso, se os pontos p_i, p_j, p_k e p_l formam um quadrilátero convexo e não estão todos no mesmo círculo, então exatamente uma entre $\overline{p_ip_j}$ e $\overline{p_kp_l}$ é uma aresta ilegal.*

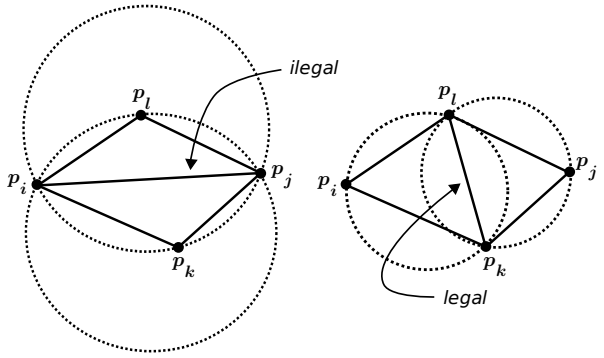


Figura 19: Trocando uma aresta ilegal por uma legal.

Note que o lema é simétrico. O ponto p_l está no interior do círculo C se e somente se o ponto p_k está no interior do triângulo $\Delta p_ip_jp_l$. Se todos os quatro pontos estiverem na fronteira do mesmo círculo, tanto $\overline{p_ip_j}$ quanto $\overline{p_kp_l}$ são arestas legais.

Definimos uma *triangularização legal* como sendo uma triangularização que não contém nenhuma aresta ilegal. Toda triangularização ângulo-ótima é uma triangularização legal. Dada uma triangularização arbitrária inicial, podemos desenvolver um algoritmo simples para computar uma triangularização legal. Basta realizarmos trocas de aresta enquanto houverem arestas ilegais.

TriangularizaçãoLegal(T)

1 **enquanto** T contém uma aresta ilegal $\overline{p_ip_j}$
2 **faça**
3 Sejam $\Delta p_ip_jp_k$ e $\Delta p_ip_jp_l$ os triângulos adjacentes a $\overline{p_ip_j}$
4 $T \leftarrow T - \overline{p_ip_j}$
5 $T \leftarrow T \cup \overline{p_kp_l}$
6 **devolva** T

Lema 3.5. *O algoritmo* TriangularizaçãoLegal *sempre termina e produz uma triangularização legal.*

Demonstração. Como removemos todas as arestas ilegais, evidentemente o algoritmo produz uma triangularização legal.

Sempre que realizamos uma troca de aresta, o vetor ângulo da triangularização aumenta. Como temos um número finito de triangularizações do conjunto P, o algoritmo sempre termina. □

3.1 Triangularizações de Delaunay

Uma *triangularização de Delaunay* é uma triangularização de um conjunto de pontos no plano com características especiais. Uma caracterização comum envolve o uso do *Diagrama de Voronoi*. Como esse conceito não pertence ao escopo desse trabalho, usaremos uma definição direta.

Seja P um conjunto de pontos no plano e seja T uma triangularização de P. Dizemos que T é uma *triangularização de Delaunay* se e somente se a circunferência circunscrita a qualquer triângulo de T não contenha nenhum ponto em seu interior.

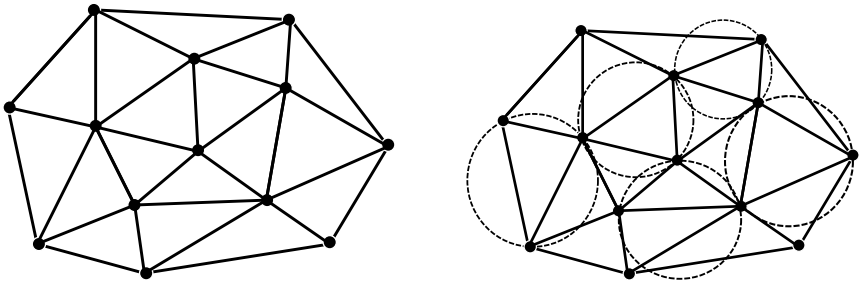


Figura 20: Uma triangularização de Delaunay. Nenhuma circunferência circunscrita a um triângulo contém um ponto.

Note que, à primeira vista, uma triangularização de Delaunay difere de uma triangularização legal. Na verdade, o seguinte resultado mostra que as duas são de fato equivalentes.

Teorema 3.6. *Seja P um conjunto de pontos no plano. Uma triangularização T de P é uma triangularização legal se e somente se é uma triangularização de Delaunay.*

Demonstração. Trivialmente, pela definição dada, uma triangularização de Delaunay é uma triangularização legal.

Definimos $C(a, b, c)$ como sendo a circunferência determinada pelos pontos a, b e c.

Suponha por contradição que T seja uma triangularização legal e que não seja uma triangularização de Delaunay. Como T não é uma triangularização de Delaunay, existe um ponto p_l no interior da circunferência circunscrita a algum triângulo, digamos $\Delta p_i p_j p_k$. Como a triangularização é legal, os pontos p_l, p_i e p_j não podem formar um triângulo. Então, existe um ponto p_m fora de $C(p_i, p_j, p_k)$ que forma o triângulo $\Delta p_i p_j p_m$. Notamos que a parte de $C(p_i, p_j, p_k)$ separada pela aresta $p_i p_j$, no semiplano que contém p_m, está contida em $C(p_i, p_j, p_m)$, ou seja, p_l também está em $C(p_i, p_j, p_m)$. Observamos também que p_l não está no interior do triângulo $\Delta p_i p_j p_m$, caso contrário a aresta $\overline{p_i p_j}$ seria ilegal. Veja figura 21a.

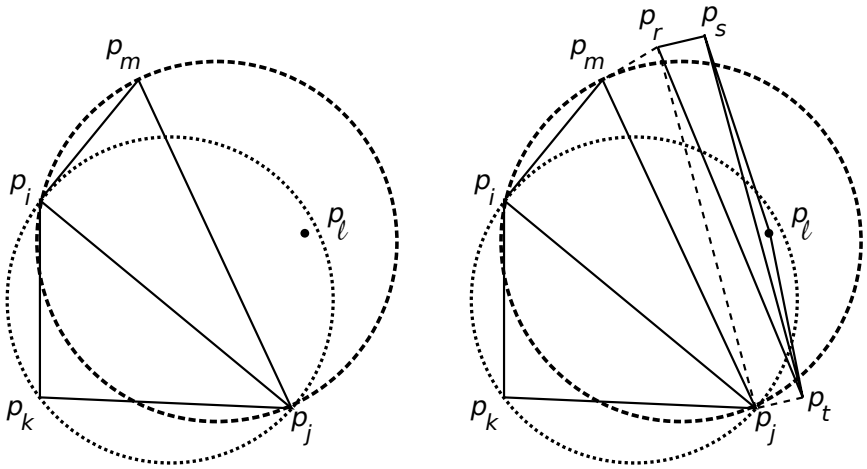


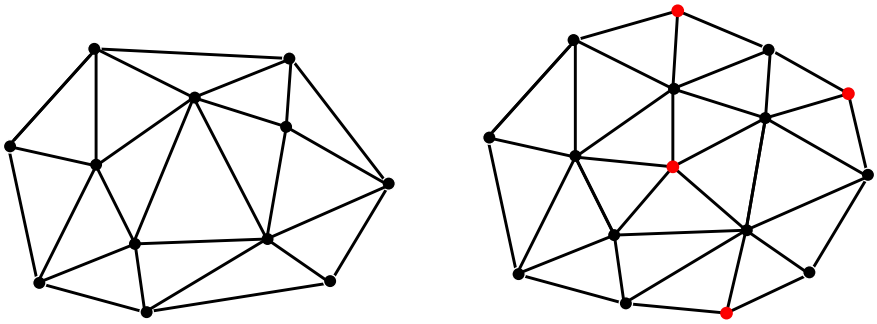
Figura 21: Uma triangularização legal é uma triangularização de Delaunay

Seja o $\Delta p_s p_t p_l$ o triângulo tal que $\overline{p_s p_t}$ intersecta $C(p_i, p_j, p_k)$ em dois pontos distintos e deixe p_k e p_l em semi-planos distintos, como na figura 21b. Novamente, a parte de $C(p_i, p_j, p_k)$ separada pela aresta $p_s p_t$, no semi-plano que contém p_l, está contida em $C(p_r, p_s, p_t)$, e portanto, p_l está contido em $C(p_r, p_s, p_t)$. Concluímos que $\overline{p_r p_t}$ é uma aresta ilegal, contrariando a triangularização T.

□

3.2 Triangularizações de Steiner

Mesmo com boas triangularizações, uma triangularização de Delaunay ainda pode apresentar triângulos com características indesejáveis. Uma forma de melhorar a qualidade da triangularização é inserir pontos artificialmente no nosso conjunto. Chamamos esses pontos de *pontos de Steiner* e a triangularização resultante do novo conjunto de *triangularização de Steiner*. Intuitivamente, não queremos adicionar muitos pontos, já que a adição de muitos triângulos pode prejudicar o propósito da aplicação.



(a) Triangularização de Delaunay (b) Triangularização de Steiner

Figura 22: Adicionando pontos de Steiner para melhorar a triangularização.

4 Resultados e produtos obtidos

Como principal produto do projeto, podemos citar a produção desse texto. Tentamos abordar o tema da triangularização de forma objetiva, mostrando resultados e algoritmos importantes na área. Acreditamos que a monografia seja um ótimo texto introdutório para o assunto.

Como subproduto do projeto, implementamos alguns dos algoritmos estudados, juntamente com uma interface gráfica que visa demonstrar os passos executados pelos algoritmos. Implementamos os algoritmos de triangularização por remoção de orelhas, o algoritmo de monotonização, de triangularização de polígonos monótonos e um algoritmo de triangularização arbitrária de um conjunto de pontos no plano. Além disso, implementamos um algoritmo força bruta e um algoritmo de divisão e conquista para o problema de achar o par de pontos mais próximo, a pedido da Professora Cristina Gomes Fernandes.

A implementação foi feita na linguagem de programação Java. O código e as instruções de uso encontram-se disponíveis na página do projeto. Inicialmente, o objetivo dessa implementação era apenas facilitar o entendimento do algoritmo, mas durante o desenvolvimento, foram implementadas bibliotecas extensas de geometria, de tal forma que o código pode ser usado como um *framework* para a implementação gráfica de algoritmos.

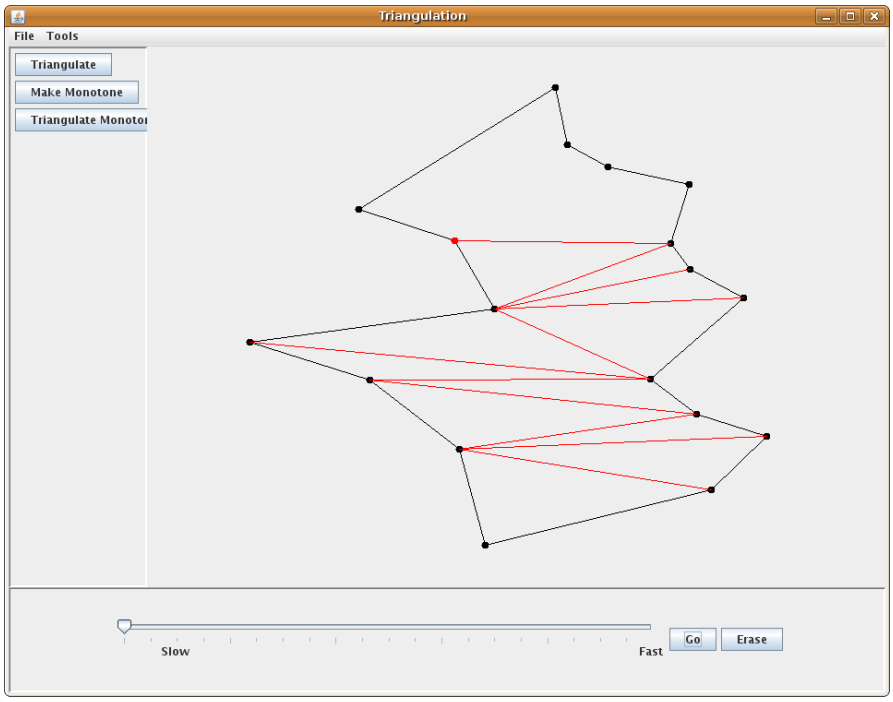


Figura 23: Interface do programa desenvolvido, rodando o algoritmo de triangularização de polígonos monótonos.

5 Conclusão

Geometria Computacional é um campo relativamente novo, mas sua importância cresceu de forma acentuada nas últimas décadas, principalmente devido às diversas aplicações que possui. A dificuldade de problemas aparentemente simples e a beleza das soluções e algoritmos atraem cada vez mais pesquisadores e trabalhos na área.

O problema inicial que escolhemos, da Galeria de Arte, é um problema antigo e bastante estudado. O resultado que mostramos diz respeito apenas a um número necessário e sempre suficiente para cobrir um polígono em função do seu número de vértices. O problema de achar o número mínimo de câmeras necessárias para cobrir um polígono simples é NP-completo [1]. A questão sobre a existência de um algoritmo linear para triangularização de polígonos ficou em aberto até 1991, quando Chazelle apresentou um algoritmo linear para o problema [4].

É pertinente falar que tratamos apenas de subdivisões triangulares. Na literatura, diversas subdivisões são estudadas. Podemos particionar polígonos em subpolígonos convexos, em polígonos com mais do que três vértices, ou outros polígonos com características especiais.

Por mais restrito que tenha sido nosso estudo, a riqueza do material proporcionou um excelente aprendizado. A complexidade dos algoritmos implementados serviram como um ótimo estudo de estruturas de dado avançadas, técnicas de construção de análise e prova de algoritmos. Finalmente, a produção desse texto trabalhou habilidades menos comuns na área da computação, tanto na formalização de conceitos matemáticos e rigor da apresentação dos resultados, quanto na escrita em uma linguagem menos formal, visando facilitar o entendimento para pessoas com menos familiaridade no assunto.

Parte III
Parte Subjetiva

6 Desafios e frustrações

O desafio inicial foi escolher um tema para o projeto. No terceiro ano do curso, decidi que gostaria de estudar algum assunto de computação de forma mais aprofundada. Procurei o Professor Carlos Eduardo Ferreira, que me mostrou diversos problemas na área de Geometria Computacional. Após pesquisar um pouco, decidimos estudar o Problema da Galeria de Arte. Esse estudo deu início a um projeto de iniciação científica que estendemos para o trabalho de conclusão de curso.

Inicialmente, tive bastante dificuldade em estudar o tema proposto. Meu último contato com conceitos de geometria plana tinha sido no colégio, e eu não possuía nenhuma familiaridade com a área de Geometria Computacional. Com a ajuda do orientador, consegui assimilar novos conceitos e resolver dúvidas pertinentes tanto da parte teórica (provas de teoremas, definições, complexidade), quanto da implementação de algoritmos.

Ainda sim, algoritmos de geometria computacional podem ser muito complicados. É importante citar as principais dificuldades que tive para implementá-los.

1. Algoritmos usam estruturas de dados avançadas. Raramente o material estudado discursava sobre a estrutura usada, indicando apenas a complexidade esperada para as operações necessárias.

2. Casos degenerados. Muitos algoritmos de geometria computacional funcionam sem problemas quando fazemos algumas suposições, como a não existência de pontos colineares, coincidentes, entre outros. No entanto, precisamos fazer modificações não triviais para eliminar essas suposições.

3. Para poder implementar uma interface gráfica, decidi usar a linguagem de programação Java. Não tinha muita experiência com a linguagem e muitas vezes isso acabou sendo um empecilho para o desenvolvimento do projeto.

Outra dificuldade enfrentada foi a de escrever a monografia. A falta de prática com a escrita, especialmente de textos científicos, fazia com que a produção do texto fosse, muitas vezes, mais devagar e cansativa. Felizmente, o Professor Carlinhos sugeria correções e dicas de como escrever trechos do trabalho, o que contribuiu muito com o desenvolvimento tanto da monografia, como da minha habilidade pessoal de escrita.

7 Disciplinas Relevantes

Durante o desenvolvimento do trabalho, tive o prazer de utilizar conhecimentos adquiridos em diversas disciplinas do curso. A maioria das matérias tiveram alguma relevância para o trabalho, direta ou indiretamente. Nesta seção, listarei apenas as matérias mais importantes.

- Geometria Computacional - Evidentemente, a matéria mais relevante para o projeto. Durante o curso, estudamos assuntos que foram cobertos nesse trabalho. O estudo proporcionou um melhor entendimento de partes delicadas do tema, além de mostrar diversos problemas de Geometria Computacional relacionados com os problemas estudados.

- Análise de Algoritmos - Qualquer trabalho de teor mais teórico conta com uma base de análise de algoritmos. Durante o trabalho, vários algoritmos foram implementados e analisados do ponto de vista de complexidade. O curso de Análise de Algoritmos foi essencial em vários momentos.

- Estrutura de Dados - Os algoritmos de Geometria Computacional são famosos pelo uso de Estruturas de Dados avançadas. Os algoritmos estudados não são exceção. Nessa matéria, aprendi diversas estruturas que foram usadas na implementação dos algoritmos, e proporcionaram uma base sólida para o aprendizado de novas estruturas.

- Desafios de Programação - Essa matéria contribuiu para o trabalho de forma indireta. Com ela, aprendi a resolver problemas, aplicar técnicas de programação que não são cobertas em outros cursos e programar de forma mais eficiente.

- Introdução à Teoria Dos Grafos - Acho importante citar essa matéria, cujo conteúdo tem pouca relação com o conteúdo estudado no trabalho. Essa disciplina foi ministrada pelo Professor Paulo Feofiloff. Nela, aprendi a entender e construir provas formais e rigorosas, habilidade muito importante para o desenvolvimento do projeto.

8 No futuro...

O Teorema da Galeria de Arte possui diversas variações. Nesse trabalho, estudamos apenas o teorema de Chvátal, sobre o menor número suficiente e necessário para cobrir um polígono de n vértices. Uma variação mencionada é que podemos restringir nossa cobertura desejada

apenas às arestas do polígono, onde as obras de arte são geralmente colocadas. Outro problema bastante interessante é o da *Rota Ótima do Vigia* [5,6]. Nele, queremos achar uma rota curta para um vigia que cobre todo o polígono, ou seja, queremos descobrir um caminho de comprimento mínimo no polígono, tal que todo ponto interior do polígono é enxergado por algum ponto deste caminho.

Quanto à triangularização de polígonos, temos algoritmos melhores que os estudados, incluindo um algoritmo linear no número de vértices de um polígono [4]. Tais algoritmos são extremamente complexos e não foram estudados.

Finalmente, estudamos apenas um tipo de triangularização de pontos no plano e uma variante, a triangularização de Delaunay e a extensão para a triangularização de Steiner. A triangularização de Delaunay é uma das mais estudadas na literatura, mas variantes que visam melhorar a aproximação de terrenos também possuem diversos resultados interessantes. Infelizmente, tivemos pouco tempo para estudar as triangularizações de Steiner, deixando espaço para um aprofundamento futuro.

Referências

[1] A. Aggarwal, *The art gallery theorem: its variations, applications and algorithmic aspects* (1984).

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational geometry: algorithms and applications*, Springer Verlag, 2000.

[3] M. Bern, D. Eppstein, and J. Gilbert, *Provably good mesh generation*, Foundations of computer science, 1990. proceedings., 31st annual symposium on, 1990.

[4] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete and Computational Geometry **6** (1991), no. 1, 485–524.

[5] W. Chin and S. Ntafos, *Optimum watchman routes*, Proceedings of the second annual symposium on Computational geometry (1986), 24–33.

[6] W.P. Chin and S. Ntafos, *Shortest watchman routes in simple polygons*, Discrete and Computational Geometry **6** (1991), no. 1, 9–31.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, MIT press Cambridge, MA, USA, 2001.

[8] S. Fisk, *A short proof of Chvatal's watchman theorem*, J. Combin. Theory Ser. B **24** (1978), no. 3, 374.

[9] O.R. Joseph, *Computational geometry in C*, Cambridge University Press, 1998.

[10] GH Meisters, *Polygons have ears*, American Mathematical Monthly (1975), 648–651.