Teorema da galeria de arte e triangularização de polígonos

Lucas Piva Rocha Corrêa[1] e
Carlos Eduardo Ferreira (orientador)[2]

[1] *Universidade de São Paulo (USP), Brazil*
piva@linux.ime.usp.br
[2] *Universidade de São Paulo (USP), Brazil*
cef@ime.usp.br

1. Introdução

Geometria Computacional é o estudo de 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 *Teorema da Galeria de Arte*, que motivou o projeto de Iniciação Científica, por nos levar à necessidade do estudo de estruturas de dados e algoritmos interessantes para sua resolução.

Na primeira seção, apresentamos o problema estudado, definições básicas e alguns resultados e provas que precisaremos para o desenvolvimento da teoria de triangularização. Na segunda seção, exploramos melhor a teoria da triangularização e algumas de suas propriedades. Por fim, apresentamos e analisamos algoritmos de triangularização de polígonos.

2. O problema

O problema da Galeria de Arte foi proposto por Victor Klee em 1973[2] 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 polígno. Veja Figura 1.

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.

Definido o que é uma cobertura, precisamos agora definir precisamente o que queremos. No problema,

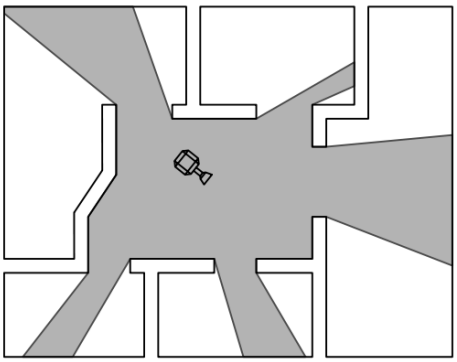

Figura 1. Uma galeria de arte modelada como um polígono no plano. A região sombreada representa a visibilidade da câmera (Fonte: Computational Geometry: Algorithms and Applications [1])

queremos encontrar o máximo sobre todos os polígonos de n vértices, do mínimo número de câmeras necessárias para cobrir o polígno, 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

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

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

especial, $n/3 \leq G(n)$, e assim achamos um limitante inferior para $G(n)$.

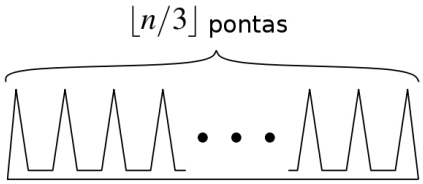

Figura 2. Pente de Chvátal. Precisamos de um guarda para cada ponta.

Vamos apresentar agora a Prova de Suficiência de Fisk de 1978[4].

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 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 diferentes triangularizações. Vamos assumir que uma triangularização sempre existe (deixamos a prova para a seção 3).

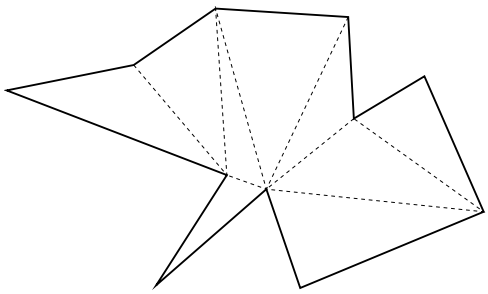

Figura 3. Um polígono simples, e uma possível triangularização, representada pelas diagonais tracejadas.

Agora, seja $G = (V, E)$ o grafo associado à 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 3.

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.

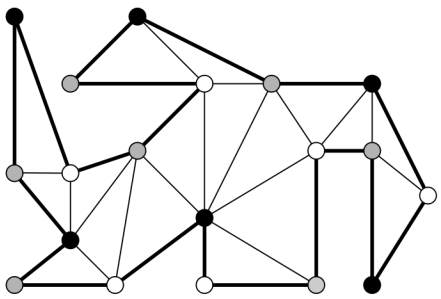

Figura 4. Um polígono e uma *3-coloração* do grafo associado à uma triangularização (Fonte: Computational Geometry: Algorithms and Applications [1])

Para concluir a prova, usamos o "Princípio da casa dos pombos" ("Pigeonhole Principle" em inglês). O princípio diz que se colocarmos n pombos em k casas de pombo, nenhuma casa pode conter mais que n/k pombos. Aqui, os nós do grafo fazem o papel dos pombos, e as cores das casas. Pelo princípio, nenhuma cor pode ser usada mais que $n/3$ vezes. Como n é um inteiro, umas das cores não pode ser usada mais que $\lfloor n/3 \rfloor$ vezes. Podemos portanto 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, pelo resto do documento, para o assunto de triangularização de polígonos, apresentando, na seção 4 algoritmos para resolver o problema de forma eficiente.

3. Triangularização: teoria

Antes de olhar para os algoritmos, precisamos estudar um pouco o campo da 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 3.1. Todo polígono possue pelo menos um vértice estritamente convexo.

Prova. 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 3.2. *(Meisters)* Todo polígono de $n \geq 4$ vértices possue uma diagonal.

Prova. Seja v um vértice estritamente convexo, cuja existência é garantida pelo Lema 3.1 e sejam a e b os vértice adjacentes a v. Se 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 vv' não pode intersectar nenhuma aresta do polígono, pois isso implicaria na existência de um vértice dentro do triângulo Δavb, mais distante de ab do que v', contrariando a escolha desse vértice. Portanto, vv' é uma diagonal.

Teorema 3.3. *(Triangularização)* Todo polígono P de n vértices pode ser particionado em triângulos pela adição de (zero ou mais) diagonais.

Prova. A prova é feita por indução. Se $n = 3$, o polígono é um triângulo, e o teorema vale trivialmente. Seja $n \geq 4$. Seja $d = ab$ uma diagonal de P, que existe, segundo o Lema 3.2. A diagonal divide o polígono em dois subpolígonos P_1 e P_2 não vazios. Como o número de vértices de cada um dos subpolígonos é menor que o número de vértices de P, aplicamos a indução nos dois subpolígonos e temos que P admite uma triangularização.

Lema 3.4. Toda triangularização de um polígono P de n vértices usa $n-3$ diagonais e consiste de $n-2$ triângulos.

Prova. 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 subpolígonos 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 subpolígonos. Aplicando a hipótese de indução para os subpolígonos, temos $(n_1 - 3) + (n_2 - 3) + 1 = n - 3$ diagonais e $(n_1 - 2) + (n_2 - 2) = n - 2$ triângulos.

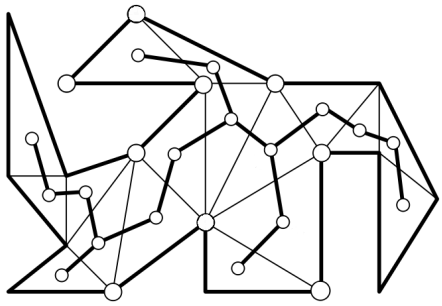

Figura 5. O grafo dual associado a uma triangularização de P (Fonte: Computational Geometry: Algorithms and Applications [1])

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

Prova. 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.

Com esse arsenal de resultados à disposição, podemos facilmente provar o Teorema das Duas Orelhas de Meisters de 1975[5] e concluir a seção com a prova da *3-coloração*. Dizemos que três vértices consecutivos a, b, c formam uma *orelha* se 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 3.6. *(Teorema das duas orelhas de Meisters)* Todo polígono com $n \geq 4$ vértices possui pelo menos duas orelhas disjuntas.

Prova. Uma folha no grafo dual corresponde a uma orelha. Pelo resultado do Lema 3.4 a árvore possui pelo menos dois nós, e toda árvore de dois ou mais nós possui pelo menos duas folhas.

Teorema 3.7. *(3-coloração)* O grafo da triangularização de um polígono possui uma *3-coloração*

Prova. Provamos por indução no número de vértices. Um triângulo pode ser trivialmente colorido

[3]Grafo conexo acíclico

com 3 cores. Seja $n \geq 4$. Pelo Teorema 3.6, 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.

4. 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 subpolígonos monótonos em $O(n \log n)$, resultado fundamental para obtenção de um algoritmo de triangularização de polígonos em tempo $O(n \log n)$. Analisaremos mais a fundo apenas os dois últimos.

4.1 Triangularização por adição de diagonais

O primeiro algoritmo segue a prova da existência de uma triangularização (Teorema 3.3): 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 3.4, para $n-3$ diagonais, o que nos leva a concluir que o algoritmo roda em $O(n^4)$.

4.2 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 3.6). 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.

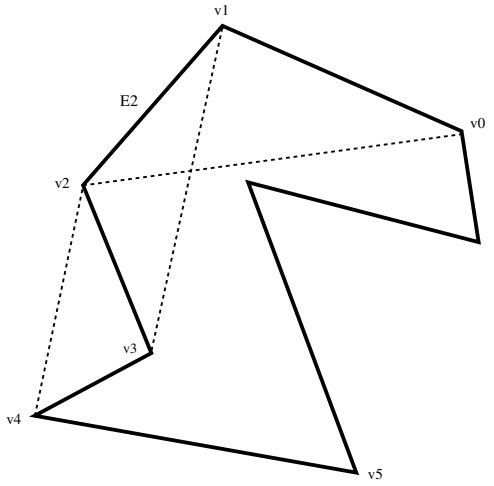

Figura 6. Removendo a orelha $E_2 = \Delta(v_1, v_2, v_3)$. O vértice v_1 deixa de ser uma ponta da orelha

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
```

4.3 Monotonização

Nessa seção, vamos precisar de algumas definições. O conceito de monotonicidade foi introduzido por Lee e Preparata, 1977 [2].

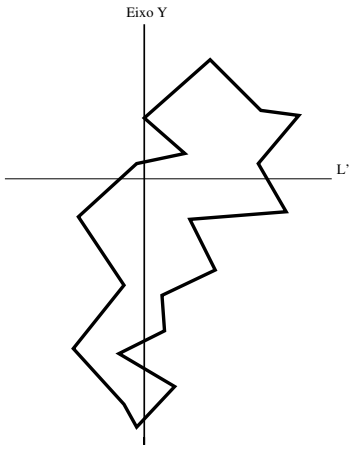

Figura 7. Um polígono y-monótono

*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 dividí-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.

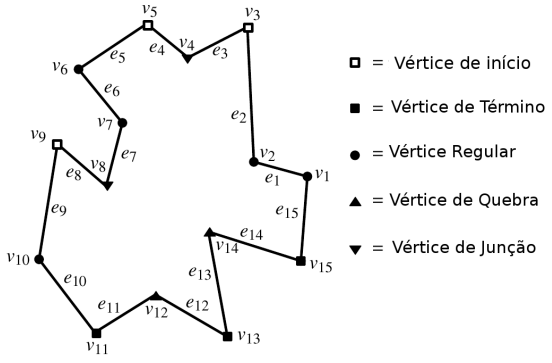

Figura 8. Tipos de vértices de um polígono (Fonte: Computational Geometry: Algorithms and Applications [1])

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

O Lema 4.1 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 *plane sweep*. O algoritmo move um linha horizontal (*sweep line*) 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 *sweep line*. 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 sweep line, tal que o segmento horizontal do vértice à 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, mas ao contrário. 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. 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 substituido 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 *sweep line*, 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 sweep line se desloca verticalmente no plano, novas arestas começam a intersectar a linha, e arestas deixam de intersectar essa linha.

Podemos esboçar 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 $T \leftarrow \{\}$ ▷ 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 Use o procedimento adequado, dependendo do seu tipo.

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

Se v_i for um vértice de início, basta adicionar e_i à T e atribuir v_i ao ajudante de e_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.

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.

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 para de intersectar a sweep line, 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.

Agora 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, mudamos $ajudante(e_j)$ para 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 subpolígonos monótonos.

5. Conclusão

Nesses primeiros meses da Iniciação Científica, o estudo foi bastante focado na construção de uma base teórica sólida no campo de Geometria Computacional. Após estudar o problema da Galeria de Arte, começamos a estudar e implementar algoritmos de triangularização de polígonos. Depois de concluir esse tópico, pretendemos extender o conceito de triangularização de polígonos e estudar o problema de triangularização de pontos no plano, ou seja, queremos encontrar uma subdivisão do interior do fecho convexo do conjunto de pontos por segmentos que não se interceptam de forma que todas as regiões formadas sejam triangulares.

Referências

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer, New York, 2005.

[2] Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1998.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, MIT Press, 2001.

[4] S. Fisk, *A short proof of Chvátal's watchmen theorem*, Journal of Combinatorial Theory B (1978).

[5] G. H. Meisters, *Polygons have ears*, American Mathematical Monthly (1975).