Algoritmos 2
Trabalho Prático 1

Eduardo Augusto Militão Fernandes
2019006540

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

eduardo1410@ufmg.br

1. Introdução

O objetivo deste trabalho é implementar um programa que, dado um conjunto de pontos que forma um polígono, resolver o problema da galeria de arte para o ambiente representado pela figura geométrica. Tal problema envolve implementar o algoritmo de Ear-Clipping e de 3-coloração de grafos, como descrito em [Vimieiro 2020].

Além disso, toda a execução do programa deve possuir uma representação gráfica interativa, a fim de permitir observar como cada passo do algoritmo funciona.

Para executar o programa, basta executar as células do notebook, alterando o valor do parâmetro *INPUT_TEST* na segunda célula para o caminho até o arquivo de teste desejado, como no seguinte exemplo:

```
INPUT_PATH = "Inputs/input2.txt"
```

2. Implementação

2.1. Primitivas Geométricas

A fim permitir a execução precisa e eficiente de todas as etapas do programa, duas funções primitivas foram implementadas: *direction(origin, a, b)* e *inside_triangle(pt, t1, t2, t3)*.

A primeira função recebe como entrada 3 pontos, sendo um deles o de origem. Assim, ela retorna se o caminho da origem até o ponto *b*, passando pelo ponto *a*, faz uma mudança de direção no sentido horário ou anti-horário. Tal processo é realizado utilizando o produto vetorial de *origin-a* e *origin-b* e, assim, não sofre com os possíveis problemas acarretados por precisão de ponto flutuante.

A segunda função informa se um ponto *pt* está dentro do triângulo formado pelos pontos *t1, t2, t3*. Para isso, ela simplesmente observa se a direção do ponto em relação às arestas do triângulo é sempre a mesma. Logo, ela também não faz uso de ponto flutuante, evitando erros de precisão.

2.2. Ear Clipping

O algoritmo de Ear-Clipping recebe como entrada um polígono e gera como saída um conjunto de triângulos disjuntos que cobrem todo o polígono e cujos vértices são todos vértices do polígono original.

O algoritmo funciona da seguinte maneira: seja um ponto P_i arbitrário de um polígono em sentido anti-horário. Se o caminho $P_{i-1} - P_i - P_{i+1}$ muda de direção no

sentido também anti-horário, e nenhum outro ponto do polígono está dentro do triângulo $T = P_{i-1}P_iP_{i+1}$, então P_i é uma ponta de orelha. Assim, o triângulo T é adicionado à lista de triângulos e P_i é removido do polígono. Este procedimento é repetido até que não haja mais de três pontos restantes.

Para a implementação do algoritmo descrito acima, foi utilizada uma lista de duplas para representar o polígono de entrada, e uma lista de lista de 3 duplas, representando a lista de triângulos gerados ao longo da execução. Cada dupla de valores representa um ponto do espaço 2D. Além disso, as primitivas *direction* e *inside_triangle* foram utilizadas para verificar o sentido dos pontos analisados e se há um ponto dentro do triângulo em questão, respectivamente.

2.3. 3-coloração

O próximo passo do algoritmo consiste em, a partir da lista de triângulos obtidos no passo anterior, fazer uma 3-coloração dos vértices do polígono original.

Assim, a primeira etapa necessária é construir o grafo dual da triangulação, onde cada vértice representa um triângulo e dois vértices estão ligados se seus respectivos triângulos compartilham uma aresta. Logo, para representar o grafo, foi utilizada uma lista de adjacências e, para verificar se dois triângulos compartilham uma aresta, foi verificado se estes possuem dois vértices em comum.

Com o grafo dual da triangulação montado e armazenado na lista de adjacências, uma busca em profundidade (DFS) é realizada neste a fim de gerar a 3-coloração. Para tal, primeiramente os vértices do primeiro triângulo são coloridos com 3 cores diferentes, representadas pelos números 1, 2, e 3. Em seguida, a DFS é realizada, onde, para cada nó, é verificado qual vértice de seu triângulo ainda não foi colorido e, então, uma cor é atribuída a ele, seguindo a fórmula:

$$color_3 = 3 - (color_1 - 1) - (color_2 - 1) + 1 \tag{1}$$

onde $color_1$ e $color_2$ são as cores dos outros vértices do mesmo triângulo.

As cores atribuídas a cada vértice são armazenadas em um dicionário de dupla (ponto no espaço 2D) para valor.

2.4. Resultado Final

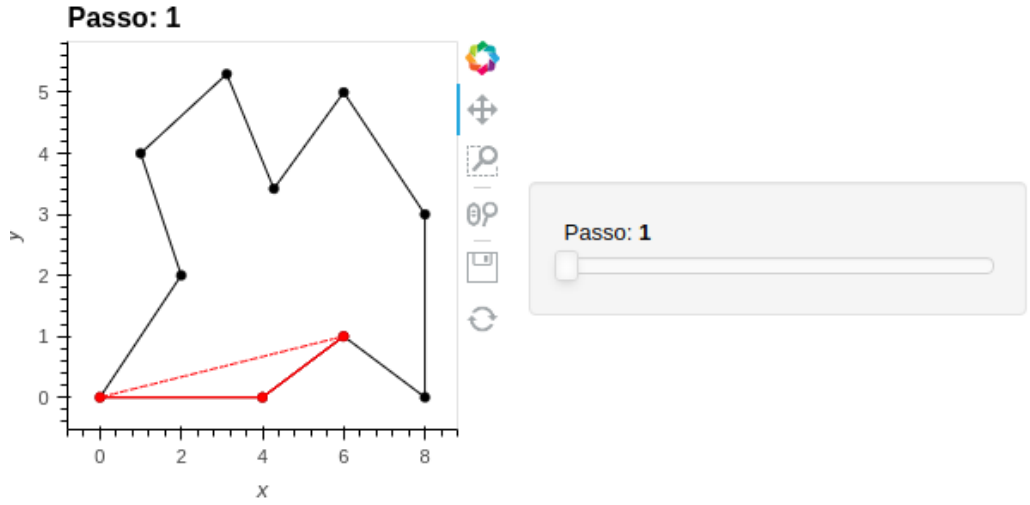
Finalmente, a fim de se descobrir com quantas câmeras é possível observar todo o recinto definido pelo polígono inicial, deve-se descobrir qual a cor com o menor número de vértices atribuídos a ela.

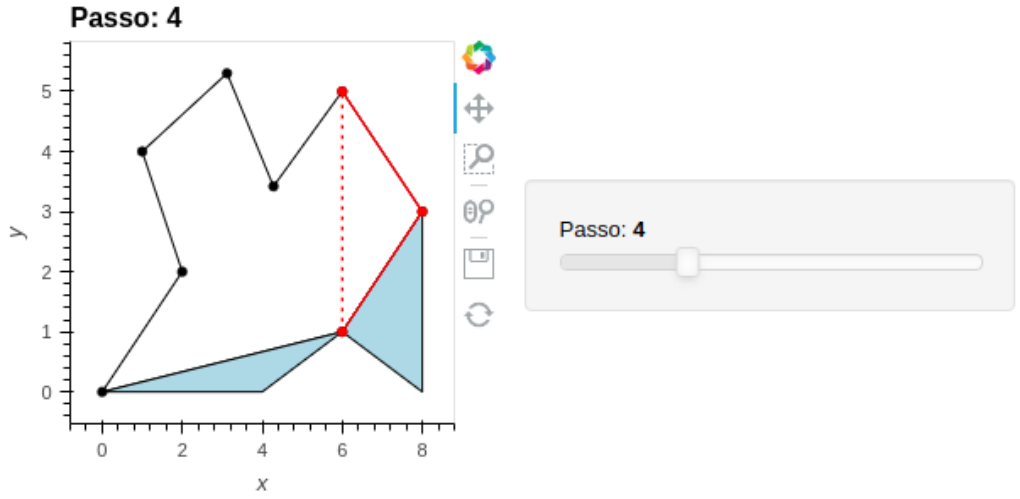
Logo, 3 novas listas de duplas são geradas, cada uma contendo os pontos de cada cor. Em seguida, o tamanho das listas é comparado, selecionando aquela de menor tamanho. Por fim, os pontos na lista escolhida são os que representam as posições das câmeras na galeria de arte.

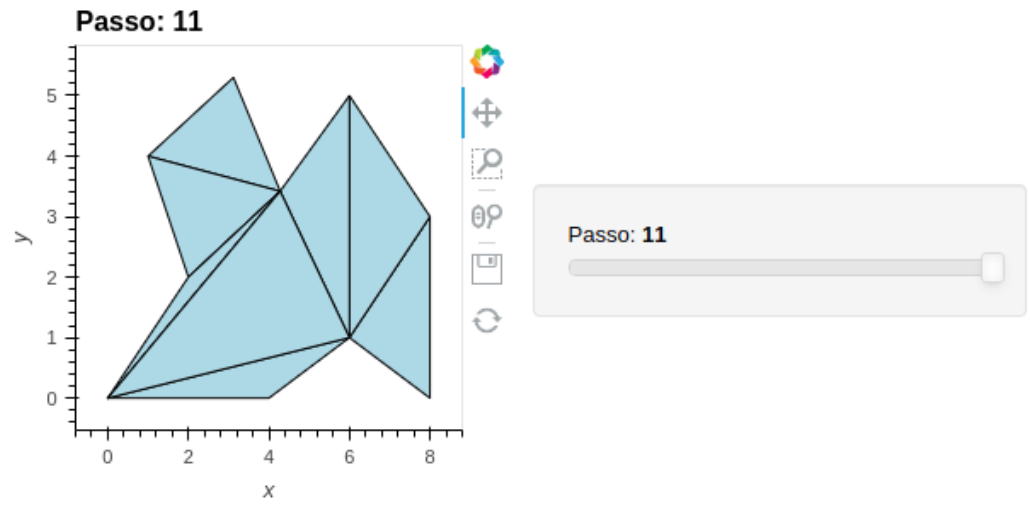
2.5. Plots interativos

Durante a execução do programa, diversos plots são gerados de maneira a permitir a visualização de cada etapa do processo. A seguir estão alguns exemplos:

2.5.1. Ear-Clipping

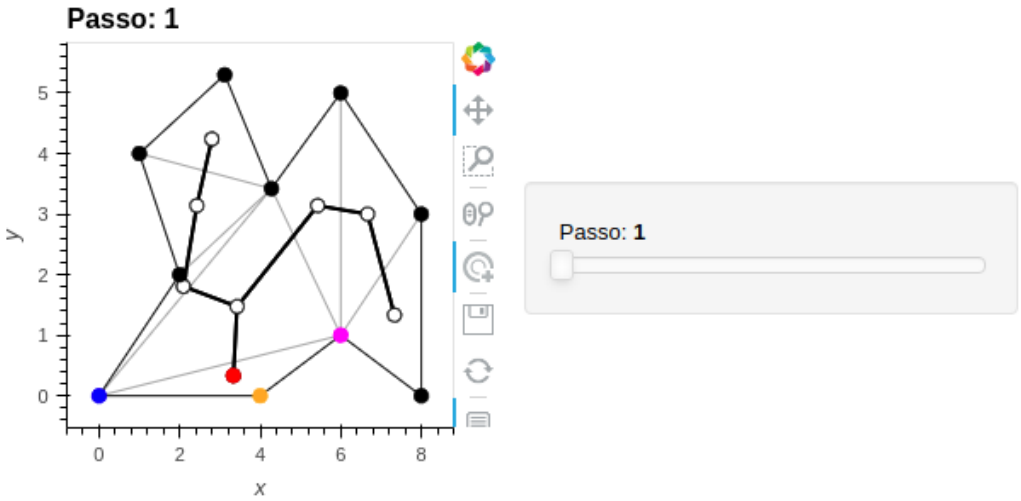


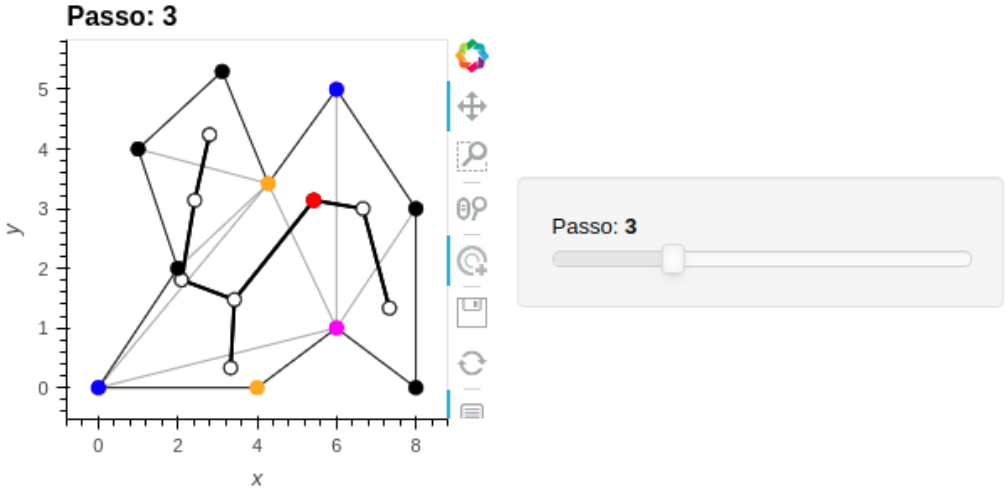


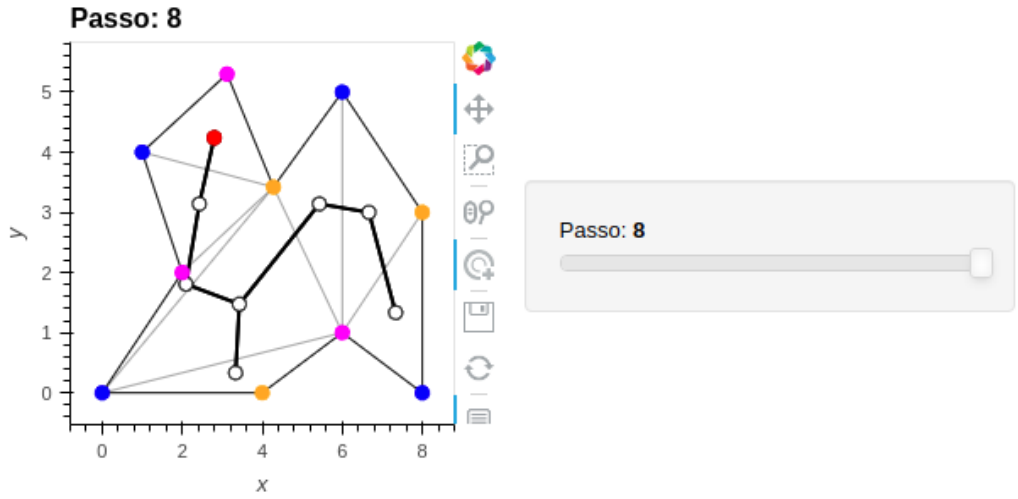


2.5.2. 3-coloração

Nos plots dessa etapa, é possível observar tanto o polígono original quanto o grafo dual da triangulação. A posição dos vértices do grafo é calculada pelo centroide de seus respectivos triângulos.

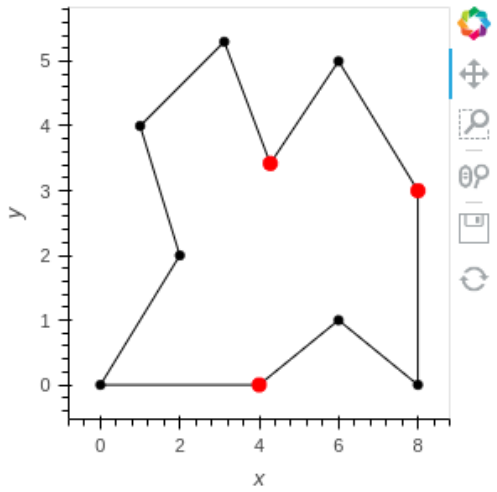






2.5.3. Saída Final

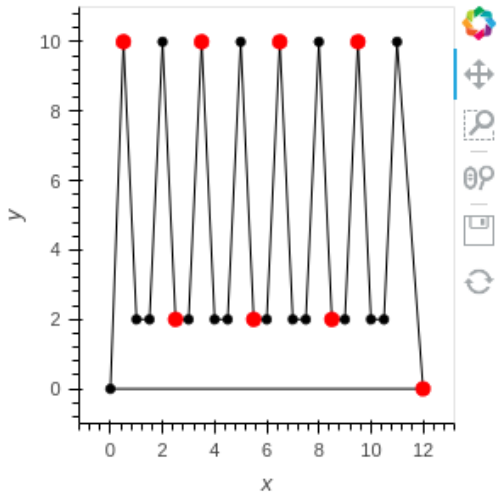
```
Pontos selecionados: [(4.0, 0.0), (8.0, 3.0), (4.28, 3.42)]
```



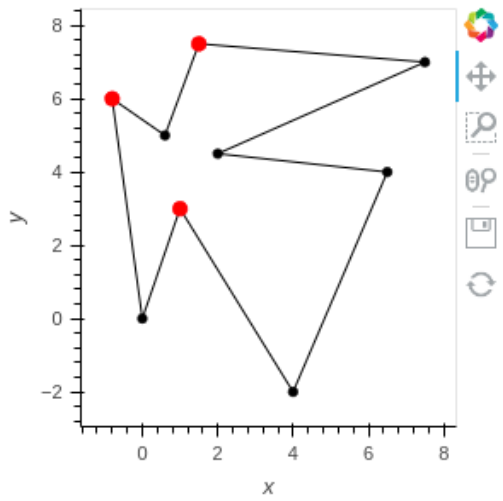
3. Testes

Muitos testes foram realizados, tanto durante a implementação do programa quanto após a finalização desta. Primeiramente, as funções primitivas foram testadas em cada um dos necessários que podem ocorrer. Em seguida, os procedimentos de Ear-Clipping e 3-coloração foram testados em várias entradas diferentes, contendo polígonos em ordem horária e anti-horária e com pontos colineares. A seguir é possível observar a saída de alguns destes testes.

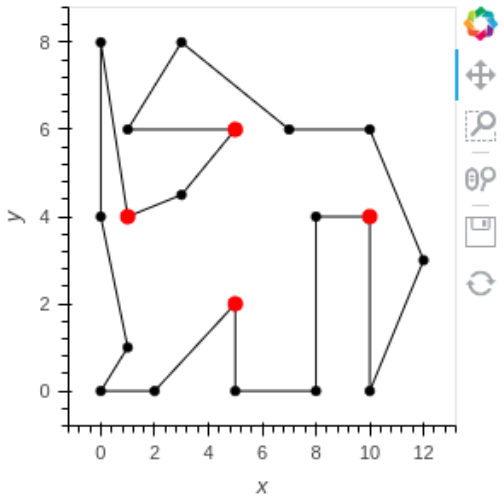
```
Pontos selecionados: [(12.0, 0.0), (9.5, 10.0), (8.5, 2.0), (6.5, 10.
0), (5.5, 2.0), (3.5, 10.0), (2.5, 2.0), (0.5, 10.0)]
```




```
Pontos selecionados: [(1.0, 3.0), (1.5, 7.5), (-0.8, 6.0)]
```



```
Pontos selecionados: [(5.0, 2.0), (10.0, 4.0), (5.0, 6.0), (1.0, 4.0)]
```



Estes e outros casos de teste estão disponíveis no repositório do projeto.

4. Conclusão

Neste trabalho foi desenvolvido um programa capaz de resolver o problema proposto de forma eficiente e elegante. Para tal, foram utilizados algoritmos clássicos de geometria computacional, a fim de se evitar ao máximo problemas de precisão por ponto flutuante. Dessa forma, foi possível fixar os conceitos vistos em aula [Vimieiro 2020], além de praticar a capacidade de abstração de problemas.

Referências

Vimieiro, R. (2020). *Slides de Algoritmos II*. Departamento de Ciência da Computação - Universidade Federal de Minhas Gerais.