Trabalho Prático 1
Problema da Galeria de Arte

Marcos Vinicius Caldeira Pacheco

Matrícula: 2019006957

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

1. Introdução

O objetivo deste trabalho prático era resolver o problema da Galeria de Arte, que consiste em descobrir o número mínimo de câmeras para cobrir toda área de uma galeria de arte representada por um polígono, utilizando triangulação Ear-Clipping e 3-coloração. Também era necessário apresentar cada passo da execução do programa de forma gráfica usando a biblioteca HoloViews em Python.

2. Implementação

O programa foi desenvolvido na linguagem Python, utilizando a biblioteca HoloViews para plotar os gráficos com o funcionamento dos algoritmos. A implementação foi feita em um Notebook de Jupyter, e está disponível no repositório do GitHub:

https://github.com/MVCP1/TP1-Algoritmos1

2.1. Entrada e Saída

Para a execução do programa, é necessário chamar a função 'problemaGaleria(points)', com o parâmetro 'points' que consiste em uma lista de tuplas, em que cada tupla são as coordenadas dos pontos do polígono que representa a galeria, em ordem anti-horária, e o primeiro ponto também aparece no final da lista. Ou seja, o menor polígono, que seria um triângulo, seria representado por uma lista de 4 tuplas.

Para inserir a entrada, basta escrever a lista dos pontos na variável global 'points' e em seguida chamar a função 'problemaGaleria(points)'.

A partir dessa entrada, a solução é computada, mas é necessário também plotar os gráficos que representam os passos do algoritmo. Para isso, é necessário chamar duas variáveis globais: a variável 'holomap' para plotar o algoritmo de triangulação, e a variável 'holomapColor', que plota o algoritmo de 3-coloração.

Com isso, a saída apresentada será a quantidade de câmeras que foi obtida, a cor que representa tais câmeras, e uma lista com as coordenadas desses pontos, bem como o plot do funcionamento dos dois algoritmos, que pode ser observado utilizando um slider que mostra passo a passo. Vale ressaltar que apesar do algoritmo conseguir resolver o problema de maneira rápida, pode demorar alguns segundos para que o plot dos processos seja exibido.

2.2. Funções Principais

Para a implementação da solução, foram criadas várias funções que auxiliam na execução do algoritmo. As principais são as seguintes:

- vector(a, b)
    - Retorna uma tupla que representa o vetor resultante do segmento de reta que vai do ponto ‘a’ até o ponto ‘b’
- prodVetorial(a, b)
    - Retorna o valor do produto vetorial entre os pontos ‘a’ e ‘b’
- inTriangle(p, q, r, s)
    - Retorna verdadeiro caso o ponto ‘p’ esteja dentro do triângulo ‘qrs’, em que ‘qrs’ foi dado em sentido anti-horário, e falso caso contrário. Vale ressaltar que foi considerado que caso o ponto ‘p’ esteja em alguma das arestas do triângulo, então ele está dentro.
- validTriangle(u, v, w)
    - Retorna verdadeiro caso o triângulo ‘uvw’ represente um triângulo válido, isto é, se é um potencial triângulo que pode ser montado durante a triangulação. Isso é obtido verificando se o produto vetorial entre os vetores ‘uv’ e ‘vw’ é maior que zero.
- isEar(point, pp)
    - Retorna verdadeiro se o triângulo formado por ‘point’, seu sucessor e seu antecessor na lista de pontos do polígono representado por ‘pp’ é uma orelha. Para ser uma orelha ele deve ser um triângulo válido e não possuir nenhum outro ponto do polígono dentro dele.
- EarClipping(points)
    - Retorna uma lista de todos os triângulos resultantes da triangulação do polígono representado por ‘points’ usando o algoritmo de ‘EarClipping’. Cada triângulo é uma lista de três tuplas.
- adjacente(tri1, tri2)
    - Retorna verdadeiro caso o triângulo ‘tri1’ e o ‘tri2’ compartilham uma aresta, ou seja, são adjacentes.
- pinta(tri, points, colors, pointsColor)
    - Essa função determina quais cores o triângulo ‘tri’ deve ser pintado, baseado no conjunto de pontos ‘points’ e na lista de suas respectivas cores ‘colors’. Ele também salva esses pontos na lista ‘pointsColor’ dependendo das cores obtidas, para que isso possa ser plotado posteriormente.
- Coloring(points, triangles)
    - Essa função retorna uma lista com as cores de todos os pontos de ‘points’, em ordem, de acordo com os triângulos ‘triangles’, aplicando 3-coloração com busca em largura (BFS).
- problemaGaleria(points)
    - Chama as funções ‘EarClipping(points)’ e ‘Coloring(points, triangles)’ e utiliza seus resultados para retornar uma lista com a cor que representa a solução, o valor mínimo de câmeras obtido, e as suas coordenadas correspondentes.

2.3. Ear-Clipping

Ao chamar a função ‘problemaGaleria(points)’, que resolve o problema, duas funções são chamadas em sequência: ‘EarClipping(points)’, e ‘Coloring(points, triangles)’. A primeira consiste na execução do algoritmo de Ear-Clipping. Essa função recebe os pontos do polígono e retorna uma lista de todos os triângulos resultantes da triangulação, além de alterar a variável global ‘holomap’ para que o processo possa ser plotado.

Primeiramente, essa função analisa todos os pontos que representam triângulos que são orelhas e os armazena em uma lista. Esses pontos são representados no plot como sendo os vértices amarelos.

Após ter encontrado todos as orelhas, a função entra em um loop até que reste apenas 3 vértices no polígono original. Dentro desse loop, o primeiro ponto da lista de orelhas é marcado de vermelho no plot e é removido do polígono original. O triângulo que ele representa, que consiste em seu antecessor, ele e seu sucessor, é adicionado na lista que será retornada ao final da execução do algoritmo. Como esse ponto foi removido, adiciona-se uma linha em vermelho entre os outros dois pontos restantes, e é reavaliado se eles se agora eles são orelhas ou não, atualizando-se assim a lista de orelhas do polígono. O loop se repete até que o polígono resultante seja apenas três pontos que são orelhas, ou seja, três pontos amarelos.

Com isso, é retornado uma lista com V-2 triângulos, em que V é a quantidade de vértices do polígono original.

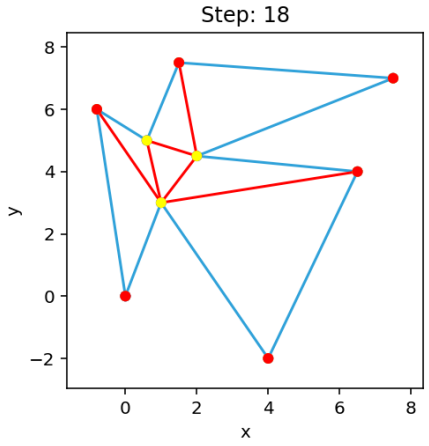


Figura 1. Exemplo do plot final de um polígono após o Ear-Clipping

2.4. 3-Coloração

Ao chamar a função ‘Coloring(points, triangles)’, é retornado uma lista com as cores de todos os pontos de ‘points’, em ordem, de acordo com os triângulos ‘triangles’, aplicando 3-coloração com busca em largura (BFS). Essa função também é responsável por alterar a variável global ‘holomapColor’ para que o processo de coloração possa ser plotado. As cores utilizadas são vermelho, azul e verde (‘red’, ‘blue’, ‘green’).

Primeiramente a função pinta os três pontos de um dos triângulos, e o remove da lista de triângulos restantes, e o adiciona na lista de triângulos para análise. Após isso, a função entra em um loop que para apenas quando não resta nenhum triângulo para pintar. Dentro do loop, todos os triângulos adjacentes aos triângulos que estão sendo analisados são removidos da lista de triângulos restantes, são pintados e adicionados na lista de triângulos vizinhos. Ao final de uma iteração, a lista de vizinhos se torna a lista de analisados. O processo se repete até todos os pontos serem pintados.

Esse processo é equivalente a fazer uma BFS em um grafo G que cada triângulo é um vértice e cada aresta representa que dois triângulos são adjascentes.

Com isso, uma lista com V cores, uma para cada ponto em ordem é retornada.

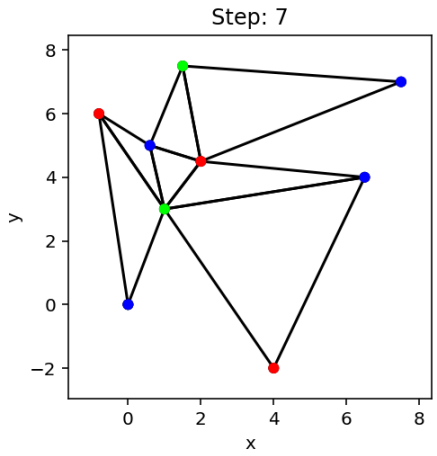
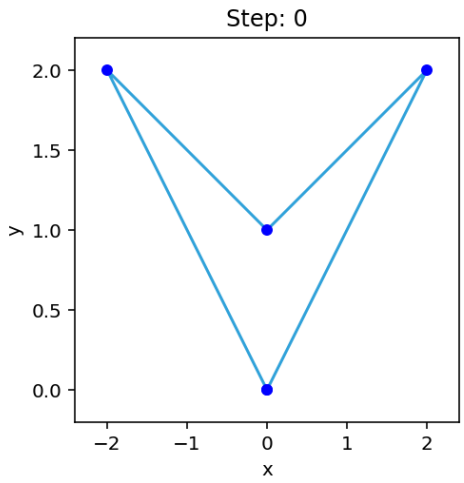
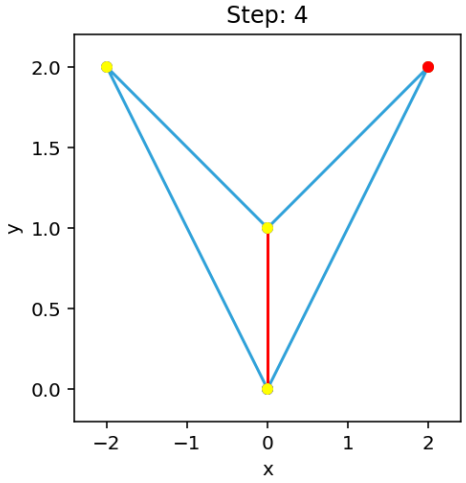


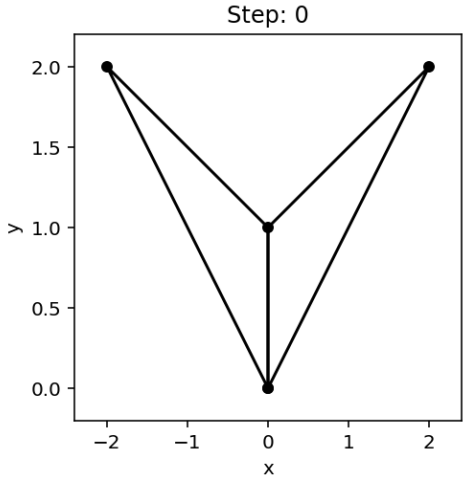
Figura 2. Exemplo do plot final de um polígono após a 3-coloração

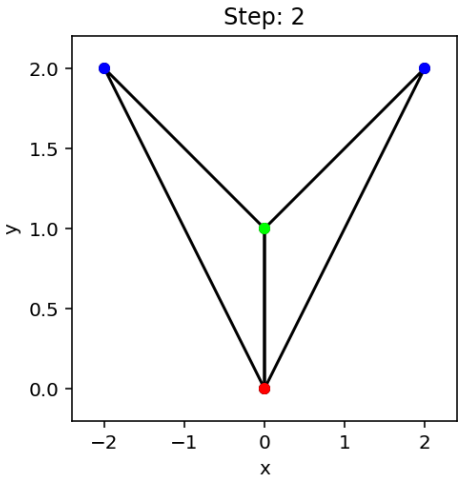
3. Casos Teste

Abaixo pode-se ver o resultado de três polígonos de teste diferentes:

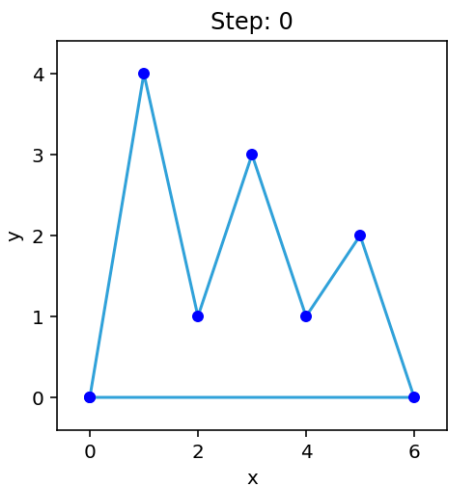


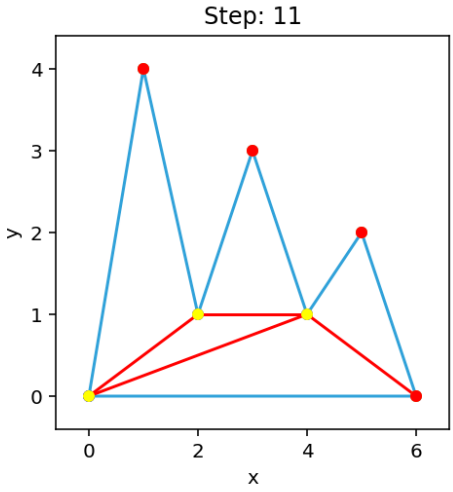


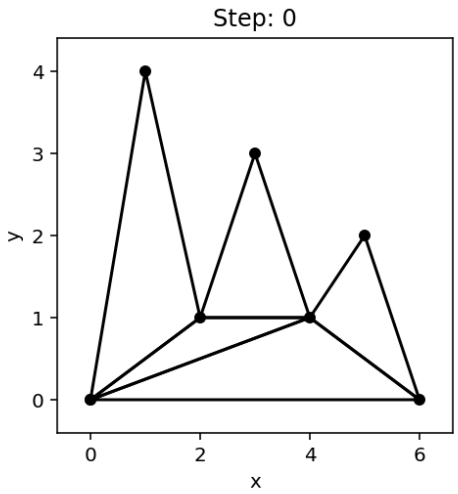


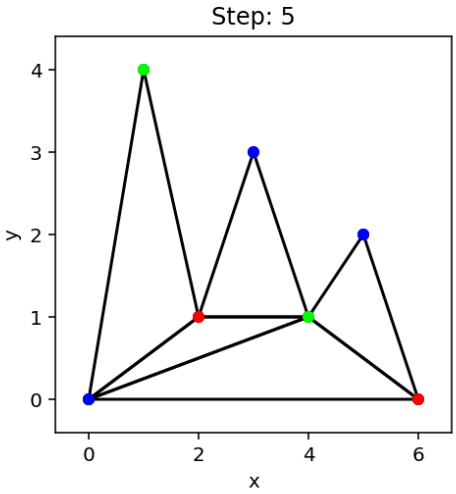


Resultado = ['red', 1, [(0, 0)]]

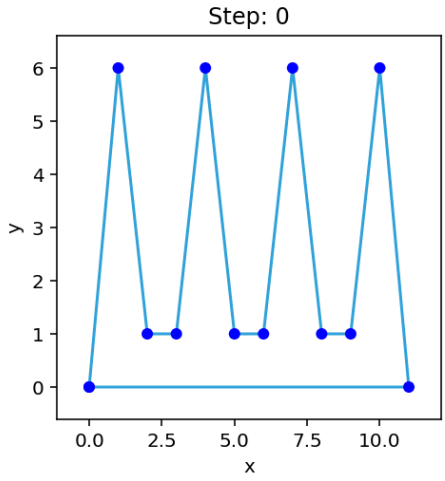


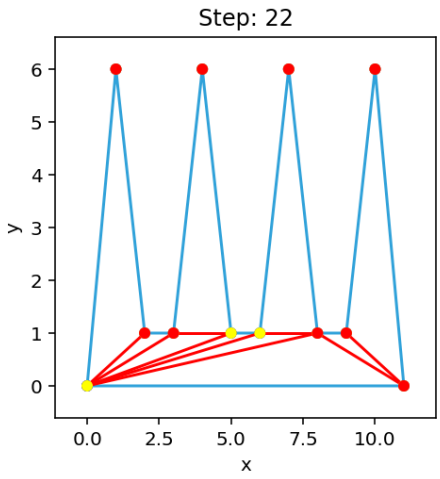


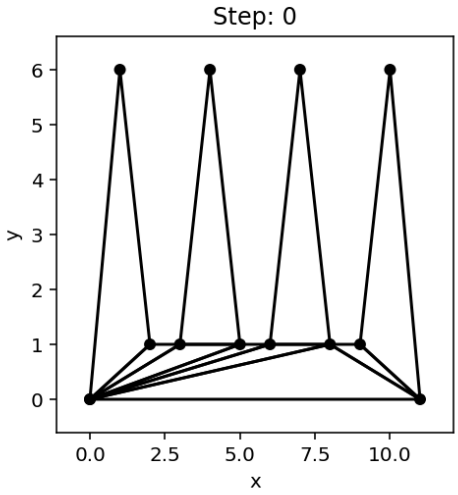


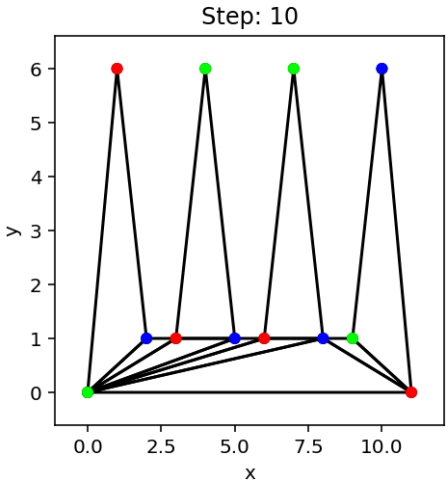


Resultado = ['red', 2, [(6, 0), (2, 1)]]









Resultado = ['red', 4, [(11, 0), (6, 1), (3, 1), (1, 6)]]

4. Conclusão

Após concluir o trabalho, foi possível perceber que o uso da biblioteca HoloViews facilitou muito na obtenção dos gráficos que mostram o processo da execução dos algoritmos. Mesmo assim, uma coisa que fica a desejar é a velocidade em que os plots são exibidos, pois dependendo da quantidade de vértices do polígono e, consequentemente, de passos do algoritmo, o plot pode demorar vários segundos para ser exibido.

Vale ressaltar que essa é uma maneira de solucionar o problema da Galeira de Arte, mas que não necessariamente retorna o menor número possível de câmeras, mas sempre uma quantidade suficiente, assim como foi ensinado pelo professor durante o curso. A implementação dos algoritmos também não é a melhor possível, resultando numa complexidade $O(n^2)$.

Com isso, foi possível desenvolver um programa que encontra a solução do problema apresentado para diferentes casos dentro do escopo solicitado.

5. Referências

Documentação da biblioteca do HoloViews do site http://holoviews.org/

Aulas da disciplina de Algoritmos 2, professor Renato Vimieiro, UFMG, 2021-1