UNIVERSIDADE FEDERAL DE MINAS GERAIS - DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DCC207 - ALGORITMOS 2 - TRABALHO PRÁTICO 01

GEOVANNA MENEGASSE SILVA - 2017020057

Introdução

O objetivo desse trabalho é propor uma solução para o Problema da Galeria de Artes. Esse problema consiste em, dada uma galeria de artes, precisamos colocar câmeras em determinados pontos de forma que toda a área da galeria seja capturada pelas mesmas. Para isso, utilizamos o método de triangulação de polígonos Ear-Clipping seguido de um algoritmo para obter a 3-coloração do grafo resultante, nos dando as posições adequadas para cada câmera.

Triangulação com Ear-Clipping

Dada uma lista de pontos ***pts*** para um determinado polígono, primeiro geramos um dicionário de adjacências ***adjacencias*** onde, para cada ponto sendo uma chave, adicionamos seus pontos vizinhos em sua respectiva lista de valores. Esse dicionário será útil para a coloração do grafo posteriormente.

Para iniciarmos a triangulação do polígono ***pts***, percorremos os pontos de ***pts*** criando uma lista ***lista_pontos*** com todos os pontos que estão no sentido anti-horário. Com ***lista_pontos*** criado, chamamos a função de detecção de uma orelha ***pega_orelha*** até que toda a lista de pontos tenha sido testada. Caso ***lista_pontos*** contenha apenas três pontos, o resultado é trivial e precisamos apenas inserir esses pontos na nossa lista de triangulação ***triangulacao***. Caso contrário, entramos de fato na função ***pega_orelha***. Essa função, percorre toda a ***lista_pontos***, pegando sempre três pontos consecutivos ***p1***, ***p2***, ***p3***. Primeiramente conferimos se esses três pontos consecutivos formam uma ponta convexa do polígono. Para isso chamamos a função ***convexo*** do nosso programa. Se a função ***convexo*** retornar verdadeiro, então os três pontos formam uma ponta convexa para nosso polígono, e portanto, precisamos verificar se, ao traçarmos um segmento ligando ***p1*** e ***p3***, existe algum outro ponto do polígono que esteja dentro do triângulo resultante. Percorremos ***lista_pontos*** e chamamos a função ***dentro_do_triangulo*** para todos os p_i pontos. Essa função verifica se o ***x*** e o ***y*** do ponto p_i estão no intervalo dos menores valores de ***x*** e de ***y***, respectivamente, entre ***p1***, ***p2*** e ***p3***. Caso esteja, o valor verdadeiro é retornado significando que ***p1***, ***p2***, e ***p3*** não formam uma orelha. Caso contrário, significa que ***p1***, ***p2*** e ***p3*** formam uma orelha. Então removemos o ponto central ***p2*** de ***lista_pontos*** e retornamos ***p1***, ***p2*** e ***p3*** para serem inseridos na nossa lista de triangulação ***triangulacao***.

3-coloração do Grafo

Para iniciarmos a coloração do grafo resultante da triangulação do polígono, precisamos tratar os nossos dados para prepararmos para a coloração. Dispomos o grafo em dicionários para facilitar nas buscas por registros.

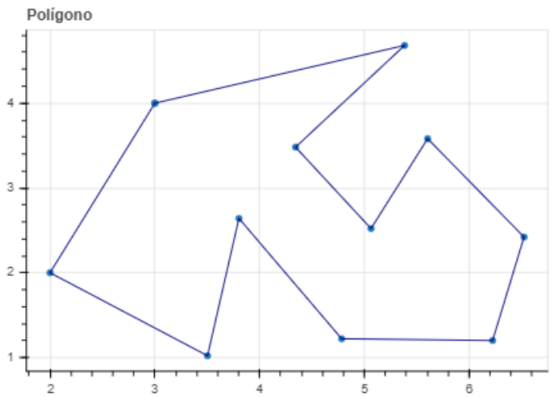
Agora vamos atualizar o dicionário de adjacências ***adjacencias*** para ***adjacencias2*** adicionando como valores dos pontos chaves os novos pontos adjacentes provenientes da triangulação. Em seguida precisamos ordenar as chaves do dicionário de adjacências com a mesma ordem em que aparecem na nossa lista de triangulação ***lista_triangulacao***.

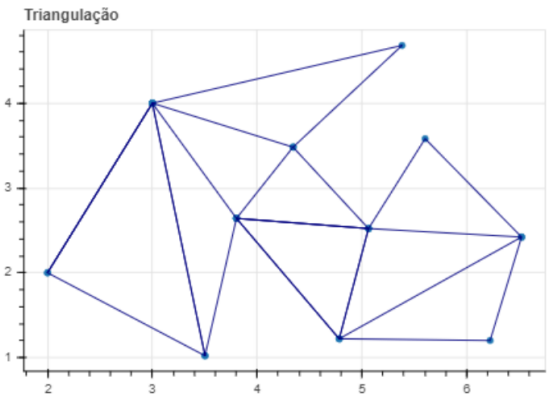
Para colorir os vértices, primeiro criamos um dicionário ***colorido*** e para cada registro no dicionário ***adjacencias2*** chamamos a função ***coloracao***. Essa função percorre o dicionário iniciando todos os registros como "falso" com relação a existir vizinhos com cores diferentes. Para cada valor de uma lista de cores ["blue", "red", "green"] verificamos se os vizinhos do ponto corrente possuem ou não a cor corrente no loop. Se possuírem, modificamos o registro para "verdadeiro" e a chave correspondente ao ponto corrente no dicionário ***colorido*** recebe como valor a cor corrente do loop, e assim quebramos o loop. Após percorrer todas as chaves no dicionário ***adjacencias2***, retornamos o dicionário ***colorido*** *com todos os pontos e suas cores.*

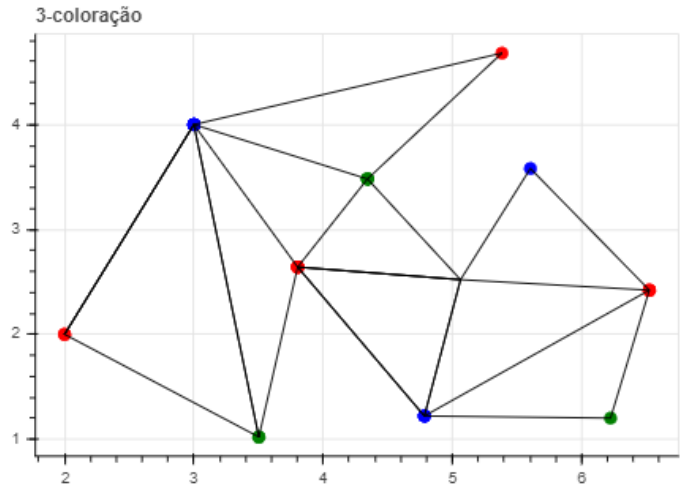
Esse algoritmo possui um problema pois, ele não garante que, independente do vértice que iniciarmos a coloração, a 3-coloração seja sempre a mesma. No próximo tópico, mostrarei o resultado da execução do programa para alguns casos de teste (disponibilizados no teams), e vamos ver que para alguns casos, a 3-coloração não funciona perfeitamente. Em contrapartida, para a triangulação, veremos que o algoritmo foi eficaz em todos os casos.

Testes e Resultados

Utilizando os polígonos de teste disponibilizados, podemos ver que apenas o último polígono teve problemas na execução do algoritmo de coloração do grafo, apesar de ter obtido sucesso no resultado da triangulação. Um dos vértices ficou sem nenhuma cor, indicando que possivelmente o polígono não consegue ser colorido com apenas três cores.



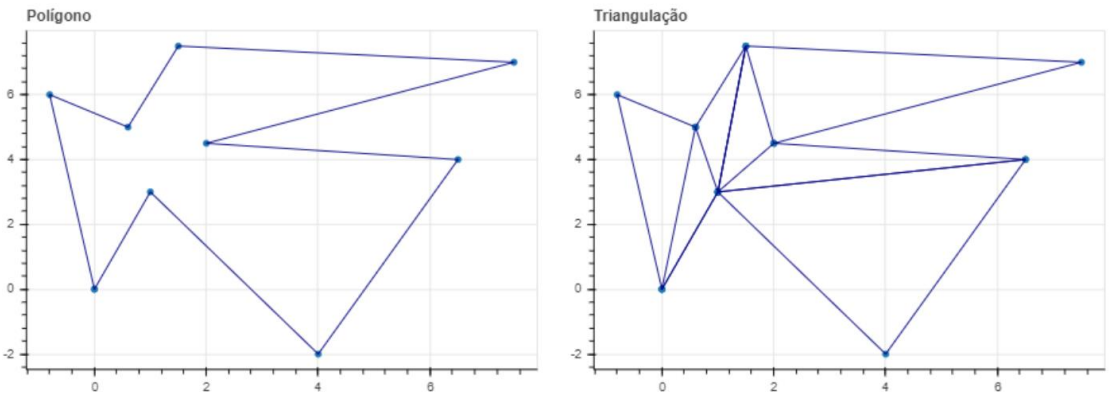


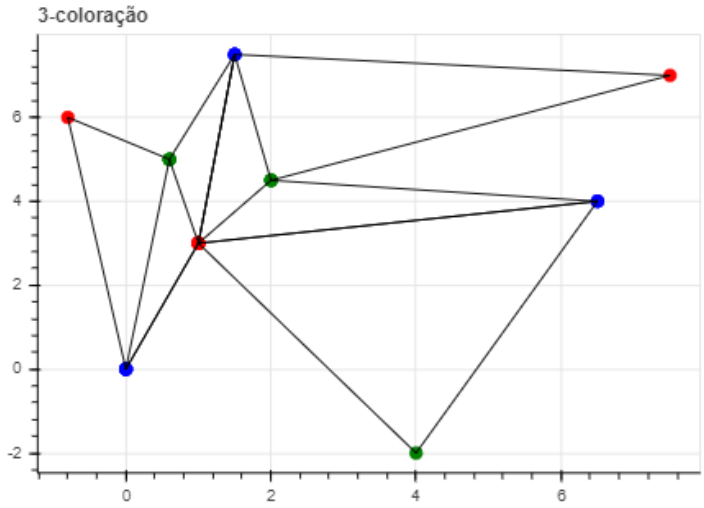


Porém, realizando a coloração por força bruta, pude encontrar uma possível 3-coloração exata para esse grafo e, por isso, descartamos a possibilidade de o problema estar no próprio polígono.

Sendo assim, conclui-se que o problema está definitivamente no algoritmo de 3-coloração criado por mim, pois a coloração varia de acordo com o vértice escolhido para iniciar. Várias tentativas foram feitas de forma a englobar o maior número de casos possível, como por exemplo: ordenar de forma decrescente o dicionário de adjacências de acordo com o número de pontos adjacentes, de forma que o algoritmo sempre se inicie dos vértices mais complicados e termine com os vértices mais simples; etc. Porém, nenhuma das tentativas de resolver o problema tiveram sucesso. Provavelmente o ideal seria analisar a aplicação do algoritmo de BFS no grafo antes de começar a coloração, mas por questões de falta de disponibilidade de tempo, não consegui prosseguir com a tentativa.

Para os outros casos de teste, o algoritmo funcionou corretamente. Como exemplo vamos mostrar o resultado do segundo polígono dado.





A coloração para esse grafo ocorreu de forma correta assim como a triangulação. Para os outros polígonos de teste o resultado também foi satisfatório, retornando resultados aparentemente corretos.

Referências

- Aulas e Slides sobre Algoritmos Geométricos (Ear-Clipping)
- Cap. 3; Computational Geometry Algorithms and Applications. Berg et al.
- https://holoviews.org (plots dos polígonos)
- https://github.com/scratchmex/art_gallery_problem/blob/master/main.py (função convexo)
- https://github.com/AyushBhandariNITK/Art_Gallery_problem/blob/master/Art_Gallery_problem/Scripts/PolygonVisibility.py (função sentido_horario)
- https://github.com/linuxlewis/tripy/blob/e3ac13ea0320cd37d0d577545f0839bbc97de1d5/tripy.py#L76 (função sentido_horario)