Trabalho Prático da Galeria de Arte

Lucca Carvalho Augusto
2019006930

1. Introdução

O problema proposto foi implementar uma solução para o problema da galeria de arte, que consiste em colocar câmeras em uma galeria de forma que a galeria inteira possa ser vigiada com o menor gasto.

Além disso, era obrigatório o uso do algoritmo de *Ear Clipping*, gerando um polígono completamente triangulado, e a *3-Coloração* que dividiria os possíveis pontos da câmera em 3 grupos, então basta fazer a escolha de um deles.

2. Implementação

O algoritmo foi desenvolvido na linguagem Python utilizando a biblioteca HoloView para os plots com o backend Bokeh, como pedido na proposta do trabalho e o notebook está no repositório do seguinte link:

https://github.com/LuccaAug/GaleriaDeArte

3. Solução do Problema

Para a solução do problema foram utilizadas 6 funções, quatro do *Ear Clipping*, sendo uma para o algoritmo principal; uma para verificar se o ponto é uma orelha; uma para verificar se um determinado triângulo possui algum ponto interno e outra para o cálculo da direção entre um ponto e outros dois. As outras 2 para a coloração, sendo uma para a real coloração do grafo e outra para plotar o polígono com os vértices coloridos.

4. Ear Clipping

Agora serão descritos os algoritmos das quatro funções utilizadas no Ear Clipping.

4.1. EarClipping

- Input:
  - p: Polígono que rodará o algoritmo
- Output:
  - diagonais: Uma lista de diagonais da seguinte forma:

[((PontoA_CoordX, PontoA_CoordY), (PontoB_CoordX, PontoB_CoordY)), ...]

  - holomap: Plot do passo a passo do Ear Clipping
- Lógica:

Inicialmente o programa cria uma lista com todas as orelhas do polígono, após este passo enquanto houver mais de dois vértices do polígono ou a lista de orelhas não estiver vazia roda o seguinte algoritmo:

Pega uma orelha, plota a diagonal que corta esta orelha e adiciona no holomap. Depois verifica se estes dois pontos da diagonal ainda são orelhas ou se tornaram a ser.

4.2. VerificaOrelha

- Input:
  - poligono: Polígono a ser analisado no algoritmo
  - index: Índice da orelha no polígono passado no parâmetro
- Output:
  - Booleana que informa se o poligono[index] é uma orelha
- Lógica:

Verifica se a função ***ContemVertice*** retorna Falso e se a ***CalculaDirecao*** retorna um valor menor que zero, que significa que o ponto está “pra dentro” do polígono.

4.3. ContemVertice

- Input:
  - poligono: Polígono a ser analisado no algoritmo
  - a: Tupla que contém a coordenada X e Y de um ponto do triângulo
  - b: Tupla que contém a coordenada X e Y de um ponto do triângulo
  - c: Tupla que contém a coordenada X e Y de um ponto do triângulo
- Output:
  - Booleana que informa se algum vértice do polígono está contido no triângulo
- Lógica:

Verifica para cada vértice, que não seja igual aos dados nos parâmetros, se a direção deste vértice em relação aos três pontos do triângulo é para dentro, ou seja, o ponto está contido no triângulo.

4.4. CalculaDirecao

- Input:
    - a: Tupla que contém a coordenada X e Y do ponto principal
    - b: Tupla que contém a coordenada X e Y do primeiro ponto a ser traçado a reta
    - c: Tupla que contém a coordenada X e Y do ponto a ser calculada a direção
- Output:
    - Retorna o valor calculado
- Lógica:

Faz o cálculo aprendido em aula para retornar a direção da reta ab para a reta ac

5. 3-Coloração

Agora serão descritos os algoritmos as duas funções utilizadas na 3-Coloração.

5.1. PlotaColoração

- Input:
    - poligono: Polígono que rodará o algoritmo
    - diagonais: Lista de retas diagonais que triangulam o polígono
- Output:
    - Plota o polígono com os vértices coloridos
- Lógica:

Faz a chamada da função ***Coloração*** e depois plota o polígono triangulado e com os vértices coloridos

5.2. Coloração

- Input:
    - poligono: Polígono a ser analisado no algoritmo
    - index: Índice da orelha no polígono passado no parâmetro
- Output:
    - lista com três listas de pontos, sendo cada uma destas listas uma cor
- Lógica:

Inicialmente instanciamos um grafo no formato matricial, sendo uma matriz NxN com N o tamanho da lista de vértices do polígono, e cada célula Grafio[i][j] é 1 se existir uma aresta ou diagonal entre os vértices de índice *i* e *j* do polígono, e 0 se não existir ligação entre estes vértices.

Instancia um dicionário em que a chave é o vértice e o valor é a cor (0, 1 ou 2), após isso roda iterativamente o seguinte algoritmo iniciando *i* com 0 até o tamanho do polígono:

Instancia uma lista de cores usadas nos vizinhos, depois disso pega a primeira cor fora desta lista e coloca no dicionário

No final transforma o dicionário em três listas para retornar do jeito que a função ***PlotaColoração*** espera.

6. Resultado

O programa inicialmente foi rodado para um polígono aleatório criado, após o correto funcionamento foi testado nos quatro polígonos fornecidos para teste. Algumas modificações foram necessárias para o correto funcionamento do algoritmo, essa modificação é apenas a inversão da lista de polígonos para ficar na mesma rotação da implementação

7. Conclusão

O algoritmo em si foi relativamente complexo, porém fácil. A parte de coloração em específico foi mais difícil que a triangulação. A parte mais difícil foi fazer o plot iterativo do passo a passo, porém com a ajuda de um colega de classe consegui uma solução fácil para este problema com a função ***hv.Holomap()***.