Trabalho Prático 1
Relatório

Thiago Poeiras Silva

11 de julho de 2021

1 Introdução

Nesse trabalho prático, foi implementado o algoritmo de triangulação de polígonos Ear-Clipping e um algoritmo de 3-coloração do grafo obtido pela triangulação.

A implementação foi feita usando a linguagem de programação Python e se encontra no arquivo ear_clipping.py. Ela pode ser executada com o comando:

```
python ear_clipping.py
```

O programa recebe como entrada primeiramente a quantidade de vértices do polígono. Na sequência, ele lê, em uma linha para cada vértice, as coordenadas dos vértices do polígono separadas por espaço.

O programa imprime as escolhas feitas em cada algoritmo e no final a coloração completa do polígono.

2 Algoritmos e Estruturas de Dados

Quantos as estruturas de dados utilizadas, foram criadas classes para pontos, contendo as coordenadas, rótulo e cor do ponto, e para o polígono contendo a lista dos pontos na ordem dada.

Foi necessária a implementação de algumas funções adicionais:

- segments: retorna um iterador com todos os segmentos do polígono.
- contains: retorna se um ponto está contido num polígono.
- vertical_intersects: determina se a semirreta vertical partindo de um ponto intersecta um segmento dado.
- turn: usa produto interno para determinar a curva feita de um segmento para outro (se menor que 0 a curva é para a direita, se maior que 0 a curva é para a esquerda e se é igual a 0 não há curva).

O algoritmo Ear-Clipping é implementado na função *ear_clipping*. Foi criada uma classe *Triangulation* para armazenar a triangulação encontrada. Essa classe consiste num dicionário onde as chaves são as diagonais escolhidas (que consiste a triangulação) e os valores são as orelhas dessas diagonais. Essa estrutura foi utilizada para facilitar o algoritmo de coloração subsequente: dessa forma é mais fácil e eficiente caminhar pela triangulação.

Foram utilizadas também duas classes adicionais, *EarPoints* e *EarPointsList* para guardar os vértices restantes a serem analisados e se cada um é uma orelha no momento ou não.

A função *ear_clipping* começa calculando, para todos os vértices, se eles são orelhas ou não. Na sequência vamos tirando orelhas dessa lista, recalculando essa propriedade para os vértices vizinhos, até que só restem 3 pontos, quando retornamos a triangulação. Durante a execução dessa função é utilizada a função adicional *is_ear* que calcula se um ponto com índice *index* é orelha no polígono formado pelos vértices em *remaining_points*.

Sempre que um vértice é retirado dessa lista, colocamos a diagonal correspondente na triangulação (isso é feito no método *add* da classe *Triangulation* que adiciona essa entrada no dicionário e atualiza os valores das orelhas anteriores). Para o último vértice é usado o método *add_last* que procura no dicionário o outro triângulo que a última diagonal adicionada também corresponde.

Na sequência, usando essa triangulação encontrada, é chamado o método *colorize* que calcula a 3-coloração do polígono. Esse método começa de um triângulo aleatório da triangulação atribuindo cores diferentes para cada vértice e coloca as 3 arestas do triângulo numa pilha para serem exploradas (também é incluído um outro valor correspondente a ponto que determina o triângulo de onde se chegou a essa aresta, dessa forma o algoritmo não anda para trás).

O algoritmo então roda uma busca em profundidade utilizando essa pilha: sempre que ela chega a um novo triângulo ele colore o vértice correspondente e coloca as outras duas arestas na pilha. O algoritmo termina quando a pilha fica vazia (indicando que a busca em profundidade terminou).

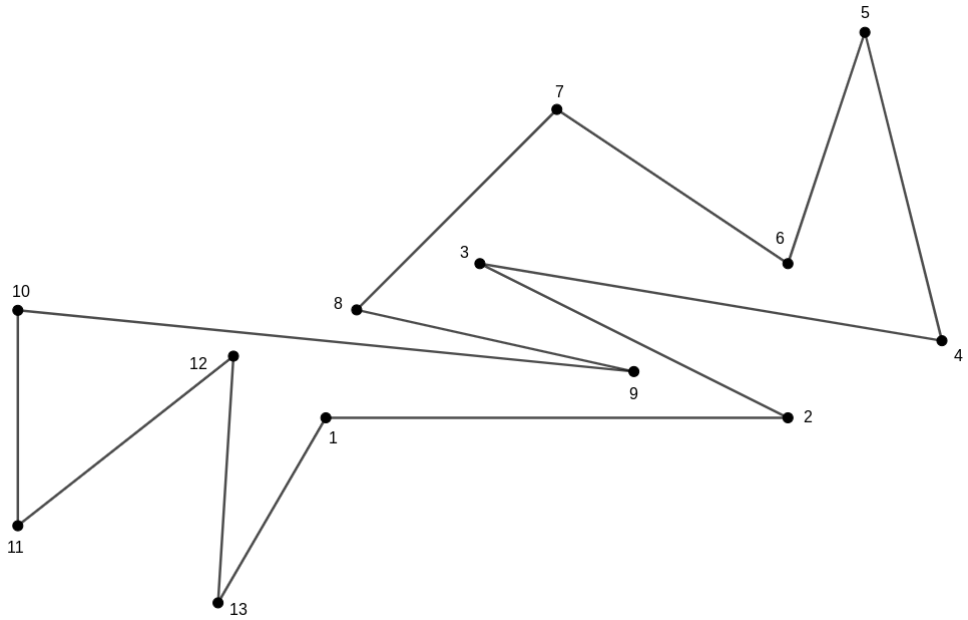
3 Exemplo

Dando a seguinte entrada para o programa:

```
13
3 2
6 2
4 3
7 2.5
6.5 4.5
6 3
4.5 4
3.2 2.7
5 2.3
1 2.7
```

```
1 1.3
2.4 2.4
2.3 0.8
```

que corresponde ao polígono:

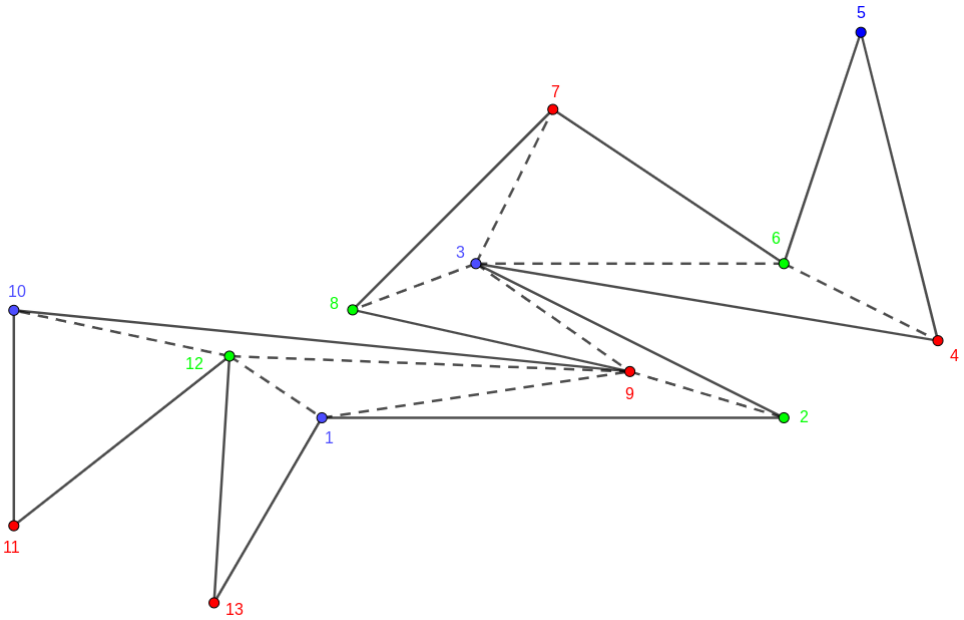


a saída do programa será:

```
 Entrada
Ponto 1: (3.0, 2.0)
Ponto 2: (6.0, 2.0)
Ponto 3: (4.0, 3.0)
Ponto 4: (7.0, 2.5)
Ponto 5: (6.5, 4.5)
Ponto 6: (6.0, 3.0)
Ponto 7: (4.5, 4.0)
Ponto 8: (3.2, 2.7)
Ponto 9: (5.0, 2.3)
Ponto 10: (1.0, 2.7)
Ponto 11: (1.0, 1.3)
Ponto 12: (2.4, 2.4)
Ponto 13: (2.3, 0.8)

 Ear-Clipping
 -> adicionou diagonal 4-6 à triangulação
 -> adicionou diagonal 3-6 à triangulação
 -> adicionou diagonal 3-7 à triangulação
 -> adicionou diagonal 3-8 à triangulação
 -> adicionou diagonal 3-9 à triangulação
 -> adicionou diagonal 2-9 à triangulação
 -> adicionou diagonal 1-9 à triangulação
```

```
 -> adicionou diagonal 10-12 à triangulação
 -> adicionou diagonal 9-12 à triangulação
 -> adicionou diagonal 1-12 à triangulação

 Coloração
 -> coloriu ponto 4 com a cor R
 -> coloriu ponto 6 com a cor G
 -> coloriu ponto 5 com a cor B
 -> coloriu ponto 3 com a cor B
 -> coloriu ponto 7 com a cor R
 -> coloriu ponto 8 com a cor G
 -> coloriu ponto 9 com a cor R
 -> coloriu ponto 2 com a cor G
 -> coloriu ponto 1 com a cor B
 -> coloriu ponto 12 com a cor G
 -> coloriu ponto 10 com a cor B
 -> coloriu ponto 11 com a cor R
 -> coloriu ponto 13 com a cor R

 Final
Ponto 1: (3.0, 2.0) -> Cor: B
Ponto 2: (6.0, 2.0) -> Cor: G
Ponto 3: (4.0, 3.0) -> Cor: B
Ponto 4: (7.0, 2.5) -> Cor: R
Ponto 5: (6.5, 4.5) -> Cor: B
Ponto 6: (6.0, 3.0) -> Cor: G
Ponto 7: (4.5, 4.0) -> Cor: R
Ponto 8: (3.2, 2.7) -> Cor: G
Ponto 9: (5.0, 2.3) -> Cor: R
Ponto 10: (1.0, 2.7) -> Cor: B
Ponto 11: (1.0, 1.3) -> Cor: R
Ponto 12: (2.4, 2.4) -> Cor: G
Ponto 13: (2.3, 0.8) -> Cor: R
```

que corresponde a seguinte triangulação e 3-coloração:


5
7
6
3
10
8
12
4
9
2
1
11
13