Exercício Prático 3 - Representação de malhas de triângulos

Aluno Kleyton da Costa (2312730)
Professor Waldemar Celes (DI/PUC-Rio)

1 Introdução

Este relatório tem como finalidade apresentar os resultados de aplicação para a representação de malhas de triângulos através de um grafo dual.

2 Metodologia

Algorithm 1 Transform to Dual

```
1:  procedure TransformToDual(n, m, v, t)
2:      dualGraph ← list()
3:      for i ← 0 to n − 1 do
4:          vertexInfo ← vertices[i] + (−1)
5:          for j ← 0 to m − 1 do
6:              if i in t[j] then
7:                  adjacentTriangule ← [t for t in t[j] if t ≠ i]
8:                  vertexInfo ← vertexInfo + tuple(adjacentTriangule)
9:              end if
10:         end for
11:         dualGraph.append(vertexInfo)
12:     end for
13:     return dualGraph
14: end procedure
```

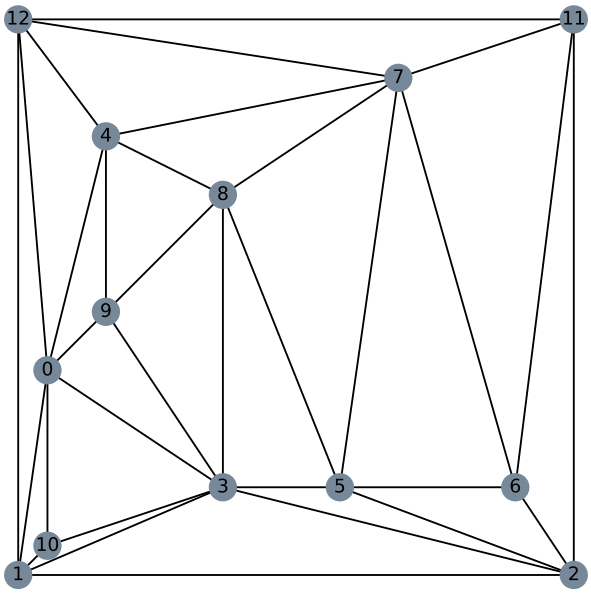
O Algoritmo 1 possui quatro elementos de entrada: o número de vértices (n); o número de triângulos (m); a lista de vértices (v); e a lista de triângulos (t). O algoritmo inicia com uma lista vazia *dualGraph* para realizar o armazenamento da representação do grafo dual. Depois os seguintes passos são executados:

- iteração sobre cada vértice de 0 até $n-1$. Para cada vértice, inicializa-se *vertexInfo* através da concatenação das coordenadas do vértice presente na lista v com -1, indicando que de início o vértice não possui um triângulo adjacente;
- o segundo loop itera de 0 até *m-1*. Para cada triângulo, checa-se se o vértice atual é parte do triângulo utilizando o operador *in*. Se o vértice for encontrado no triângulo, uma lista chamada *adjacentTriangule* é criada iterando sobre os vértices do triângulo e excluindo o vértice atual. Essa lista possui o triângulo adjacente do vértice atual;
- Após o loop, o *vertexInfo* é atualizado por uma concatenação com a lista de *adjacentTriangule*. Por fim, o *vertexInfo* é anexado a lista *dualGraph*;
- Uma vez que todos os vértices forem processados, o algoritmo retorna a lista *dualGraph* - sendo esta a representação do grafo dual.

A complexidade do pseudocóidigo é de ordem $O(nm)$, em que n é o número de vértices e m é o número de triângulos. A justificativa é de que o primeiro loop itera n vezes e o segundo loop itera m vezes. Dentro dos loops as operações (concatenação de listas e tuplas, a utilização do operador *in* e iterar sobre os vértices do triângulo) são constantes.

3 Experimentos

Através da malha de triângulos disponibilizada, chegamos no seguinte grafo dual (Figura 3).



Cada linha na Tabela 3 é uma lista contendo informações sobre um vértice e seus triângulos adjacentes.

O experimento realizado (Figura 3) mostra que, como esperado, o tempo de execução do algoritmo cresce linearmente em função do produto entre o número de vértices e triângulos. Para este experimento foram consideradas 100 amostras com tamanho de entrada $n = 100$ *to* 10100.

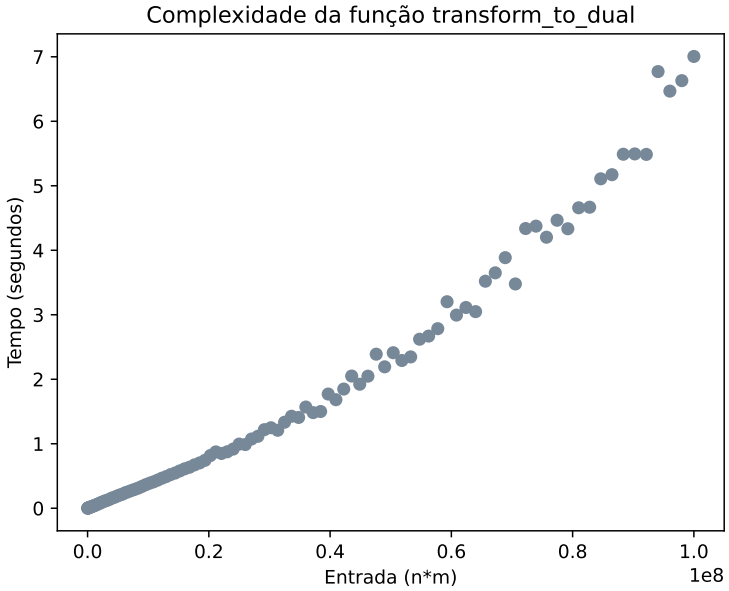


Tabela 1: Informações da malha

n	**m**	**t's**																													
100.0	400.0	-1	1	10	1	12	3	9	4	12	4	9	3	10																	
50.0	50.0	-1	0	10	0	12	2	3	3	10	-1	0	10	0	12	2	3	3	10												
1000.0	50.0	-1	1	3	3	5	5	6	6	11	-1	1	3	3	5	5	6	6	11												
400.0	200.0	-1	0	9	1	2	2	5	8	9	5	8	1	10	0	10	-1	0	9	1	2	2	5	8	9	5	8	1	10	0	10
200.0	800.0	-1	0	12	7	12	7	8	0	9	8	9	-1	0	12	7	12	7	8	0	9	8	9								
600.0	200.0	-1	3	2	3	8	2	6	6	7	7	8	-1	3	2	3	8	2	6	6	7	7	8								
900.0	200.0	-1	5	2	5	7	2	11	7	11	-1	5	2	5	7	2	11	7	11												
700.0	900.0	-1	5	6	4	12	5	8	4	8	6	11	11	12	-1	5	6	4	12	5	8	4	8	6	11	11	12				
400.0	700.0	-1	3	9	3	5	5	7	7	4	4	9	-1	3	9	3	5	5	7	7	4	4	9								
200.0	500.0	-1	0	3	3	8	4	0	8	4	-1	0	3	3	8	4	0	8	4												
100.0	100.0	-1	0	1	1	3	3	0	-1	0	1	1	3	3	0																
1000.0	1000.0	-1	6	2	7	6	7	12	-1	6	2	7	6	7	12																
50.0	1000.0	-1	1	0	0	4	4	7	7	11	-1	1	0	0	4	4	7	7	11												