Relatório do Trabalho 2 de Geometria Computacional

Bruno Dal Pontte & Ruibin Mei

2 de julho de 2025

Resumo

Esse relatório tem como objetivo explicar as escolhas de implementação e o funcionamento do Trabalho 2 de Geometria Computacional. Este trabalho consiste em implementar um algoritmo que gera uma DCEL (*Doubly connected edge list*). Porém, antes de gerar a DCEL, são realizadas verificações a respeito da malha fornecida como entrada, que contém as descrições dos vértices e das faces nela contidos. Nelas, é verificado se a malha está inclusa em um de 3 casos, que incluem: uma malha aberta; uma malha que não é subdivisão planar; e uma malha superposta. No caso de malha superposta, são realizadas duas verificações, sendo uma delas auto-interseção dentro das faces, e a outra interseção entre as faces. Se a malha estiver incluída em um ou mais desses casos, um desses casos é impresso. Todavia, se após todas essas verificações a malha não estiver dentro de nenhum caso, então é gerada uma DCEL correspondente à entrada.

1 Introdução

1.1 Classificação da malha e estrutura da DCEL

O programa implementado para o Trabalho 2 de Geometria Computacional tem como objetivo gerar uma DCEL (*Doubly connected edge list*) correspondente a uma descrição de uma malha fornecida na entrada, na qual estão descritos os vértices e as faces nela contidos. Porém, antes disso, assim como especificado no enunciado, algumas verificações são realizadas a respeito dessa malha fornecida como entrada. Essas verificações estão listadas a seguir:

- Malha aberta, ou seja, as faces dadas não cobrem toda a região do espaço descrita por essa malha. Para essa verificação, é conferido se existem arestas que possuem apenas uma única face associadas a elas. Caso sim, isso significa que a malha é aberta;
- Não subdivisão planar, ou seja, a malha não corresponde a uma subdivisão de um plano, e portanto não pode ser representada em um ambiente bidimensional sem interseções entre as faces. Para essa verificação, é conferido se existem arestas que possuem mais de duas faces associadas a elas. Caso sim, isso significa que a malha não corresponde a uma subdivisão planar;
- Malha superposta, ou seja existem faces superpostas na malha. Este caso está subdividido em dois casos particulares:
  - Auto-interseção dentro de uma face: quando uma face tem interseção com ela própria, seja nas suas arestas ou nos seus vértices. Para cada face da malha, é verificado se há interseção entre cada par de arestas. Caso sim, ocorre auto-interseção nessa face;
  - Interseção entre faces: Para cada par de faces da malha verifica-se primeiro se há interseção entre as arestas de uma face com uma da outra face, o que indica sobreposição; depois, é verificado se uma face está dentro da outra, usando o Teorema da Curva de Jordan (*Ray Casting*) [1]. Entretanto, é preciso levar em consideração os casos onde as faces são externas à malha, ou seja, em que elas estão no sentido horário. Para isso, é considerado que não ocorre interseção entre arestas com sentidos diferentes. Além disso, existe o caso em que duas faces no sentido horário, embora não estejam contidas uma na outra do ponto de vista euclidiano, ainda assim levam a uma sobreposição de faces, pois elas contemplam todo o espaço externo, o que leva a uma sobreposição. Esses casos particulares são verificados e tratados nessa implementação;

Após essas verificações, é gerada como saída uma DCEL correspondente à malha fornecida. O algoritmo utilizado para gerar essa DCEL está descrito na seção correspondente a seguir.

2 Estruturas de Dados

As estruturas de dados utilizadas neste trabalho são:

2.1 Estruturas da `MALHA`

2.1.1 Estrutura `ArestaInfo`

Campo	Tipo	Descrição
`v1, v2`	`int`	Índices dos dois vértices da aresta.
`contagem`	`int`	Número de faces que utilizam esta aresta.
`indice_v1_v2`	`int`	Índice da semi-aresta que vai de `v1` para `v2`.
`indice_v2_v1`	`int`	Índice da semi-aresta que vai de `v2` para `v1`.

2.1.2 Estrutura `TabelaArestas`

Campo	Tipo	Descrição
`arestas`	Ponteiro para `ArestaInfo`	Array dinâmico contendo todas as arestas únicas da malha.
`num_arestas`	`int`	Quantidade total de arestas únicas.
`capacidade`	`int`	Capacidade alocada atual do array de arestas (para controle de alocação dinâmica).

2.1.3 Estrutura `VerticeEntrada`

Campo	Tipo	Descrição
`x, y`	`int`	Coordenadas do vértice.

2.1.4 Estrutura `FaceEntrada`

Campo	Tipo	Descrição
`indices`	`int*`	Ponteiro para um array de índices de vértices que formam a face.
`n_vertices`	`int`	Número de vértices que compõem a face.

2.1.5 Estrutura `Malha`

Campo	Tipo	Descrição
`vertices`	Ponteiro para `VerticeEntrada`	Lista de vértices da malha.
`faces`	Ponteiro para `FaceEntrada`	Lista de faces da malha.
`tabela`	Ponteiro para `TabelaArestas`	Estrutura com informações detalhadas das arestas.
`n_vertices`	`int`	Quantidade total de vértices.
`n_faces`	`int`	Quantidade total de faces.

2.2 Estruturas da DCEL (Doubly Connected Edge List)

Estrutura `VerticeDCEL`

Campo	Tipo	Descrição
`x, y`	`int`	Coordenadas do vértice.
`semi_aresta`	`int`	Índice de uma semi-aresta incidente ao vértice.

2.2.1 Estrutura `FaceDCEL`

Campo	Tipo	Descrição
`semi_aresta`	`int`	Índice de uma semi-aresta que tem a face como face esquerda.

2.2.2 Estrutura `SemiAresta`

Campo	Tipo	Descrição
`origem`	`int`	Índice do vértice de origem da semi-aresta.
`simetria`	`int`	Índice da semi-aresta oposta (simétrica).
`face_esquerda`	`int`	Índice da face à esquerda da semi-aresta.
`proximo`	`int`	Índice da próxima semi-aresta no sentido anti-horário.
`anterior`	`int`	Índice da semi-aresta anterior no sentido anti-horário.

2.2.3 Estrutura `DCEL`

Campo	Tipo	Descrição
`vertices`	Ponteiro para `VerticeDCEL`	Lista de vértices da DCEL.
`faces`	Ponteiro para `FaceDCEL`	Lista de faces.
`semi_arestas`	Ponteiro para `SemiAresta`	Lista de semi-arestas (duplas orientadas).
`n_vertices`	`int`	Quantidade total de vértices.
`n_arestas`	`int`	Quantidade total de arestas (não de semi-arestas).
`n_faces`	`int`	Quantidade total de faces.

3 Algoritmo de Verificação

3.1 Verificação de malha aberta

Para a verificação de malha aberta, foi feito um esquema de pré-processamento na leitura da entrada de dados, na qual, quando lê uma face, é chamado a função *contar_aresta()* que percorre todos vértices que essa face contém somando 1 indicando que essa aresta tem uma face associada a ela e assim por diante para todas as faces de entrada. No final de todas as contagens teremos as informações armazenadas na estrutura `ArestaInfo` no campo contagem, utilizando um laço podemos verificar se tem alguma aresta é fronteira de somente uma face, assim determinando se essa malha está aberta ou não.

Para o pré-processamento o custo é de $\Theta(f.v)$ sendo f o número de faces e v o número de vértices, e para a verificação da quantidade de face de cada aresta tem o custo no pior caso $\mathrm{O}(n)$ sendo n o número de arestas únicas.

Representando matematicamente:

Seja $F = \{f_1, f_2, \ldots, f_m\}$ o conjunto de faces da malha, onde cada face f_k é definida por uma sequência cíclica de vértices $(v_1, v_2, \ldots, v_n)$. A cada aresta $e_{i,j} = (v_i, v_j)$ associamos um contador $c(e_{i,j})$, inicializado com zero.

Para cada face $f_k \in F$, percorremos suas arestas e incrementamos o contador associado a cada aresta:

$$c(e_{i,j}) = \sum_{f \in F} \delta_{e_{i,j} \in f}$$

onde:

$$\delta_{e_{i,j} \in f} = \begin{cases} 1, & \text{se } e_{i,j} \text{ pertence à face } f \\ 0, & \text{caso contrário} \end{cases}$$

Ao final do processamento, verificamos se existe alguma aresta $e_{i,j}$ tal que:

$$c(e_{i,j}) = 1$$

Neste caso, a aresta é fronteira de apenas uma face, o que caracteriza uma malha aberta.

3.2 Verificação de malha não subdivisão planar

A verificação de malha não subdivisão planar foi feito aproveitando o pré-processamento citado na verificação de malha aberta, para determinar foi usado um laço percorrendo a estrutura `ArestaInfo` no campo contagem verificando se alguma aresta única tem mais de 2 faces relacionada a ela, ou seja,

$$c(e_{i,j}) > 2$$

O custo para pré-processamento foi de $\Theta(f.v)$ sendo f o número de faces e v o número de vértices. E a verificação é no pior caso $O(n)$ sendo n o número de arestas únicas.

3.3 Verificação de faces

Para verificar a superposição das faces, primeiro é verificado se cada uma das faces possui auto-interseção consigo mesma. Para isso, é utilizado um algoritmo simples de verificar interseções entre todos os pares de arestas das faces, o que possui custo $O(n^2)$ no número de arestas para cada face.

Depois, é verificado se ocorre interseção entre cada par de arestas dentre as faces, sendo uma aresta de cada face. Novamente é usado um algoritmo simples de custo $O(n^2)$ no número de arestas para cada par de faces.

Após isso, é verificado se uma das faces está contida na outra, usando um algoritmo de *Ray Casting* baseado no Teorema da Curva de Jordan [1]. Esse algoritmo também tem custo $O(n^2)$ no número de arestas para cada par de faces. Esse algoritmo simplesmente conta o número de intersecções entre as arestas (segmentos de reta) de uma face com uma semirreta horizontal que tem origem em um ponto da outra face e continua para toda posição x maior que a do ponto, mantendo a mesma posição y. Caso esse resultado seja ímpar, considera-se que o ponto está dentro da face; caso seja par, considera-se que esteja fora.

Para verificar se a aresta a de uma face intersecta essa semirreta com origem no ponto p, é utilizada a seguinte equação que calcula o x da interseção:

$$x_{\text{int}} = a_x + (p_y - a_y) \cdot (b_x - a_x)/(b_y - a_y)$$

Onde caso $x_{\text{int}} > p_x$, então ocorre interseção com essa semirreta; Caso contrário, essa interseção aconteceria antes da origem da semirreta, e portanto é ignorada.

Esse algoritmo também leva em conta os casos onde os polígonos estão no sentido horário (ou seja, são polígonos externos), tratando-os de maneira apropriada. Para isso, é necessário saber o sentido da orientação das faces, o que é implementado pelo algoritmo descrito a seguir.

3.3.1 Verificação da orientação da face

Dada uma face com n vértices representados pelos pontos:

$$(v_1, v_2, \ldots, v_n), \quad \text{com } v_i = (x_i, y_i)$$

A fórmula utilizada para verificar a orientação da face é baseada em uma versão modificada da fórmula de *Shoelace* [2], pois não precisamos do $\frac{1}{2}$ antes do somatório pela razão em que estamos interessados no sinal dele e não a área do polígono:

$$\text{Área relativa} = \sum_{i=1}^{n} (x_{i+1} - x_i)(y_{i+1} + y_i)$$

onde:

$$(x_{n+1}, y_{n+1}) = (x_1, y_1)$$

Essa soma não corresponde diretamente à área do polígono, mas mantém o mesmo sinal da área assinada (positiva para orientação horária, negativa para anti-horária).

A orientação da face é então determinada por:

$$\text{orientação} = \begin{cases} 1, & \text{se Área relativa} < 0 \quad \text{(anti-horário)} \\ -1, & \text{se Área relativa} > 0 \quad \text{(horário)} \\ 0, & \text{se Área relativa} = 0 \quad \text{(caso degenerado)} \end{cases}$$

4 Algoritmo para Construção da DCEL

A estrutura da DCEL [3] se consiste em n (n_vertices), m (n_arestas) e f (n_faces) sendo n o número de vértices, m o número de arestas únicas e f o número de faces.

- No campo vértices da estrutura é acompanhado junto com um índice da semi-aresta além das coordenadas, para a nossa implementação, foi feito a escolha do menor índice da semi-aresta que tem como origem esse vértice.
- No campo faces, é apenas um índice de uma semi-aresta que tem a face como face esquerda, para isso, foi considerado a primeira semi-aresta na ordem de entrada dos vértices da face.
- No campo semi-arestas, são todas as informações como origem, simetria, face_esquerda, próximo semi-aresta e por fim semi-aresta anterior, nesse último campo, teremos 2∗quantidade de arestas únicas, pois cada aresta contém duas semi-arestas.
  - As informações de origem indica o índice do vértice.
  - A simetria como temos as informações pré-processadas na entrada da malha (estrutura ArestaInfo) temos os dados de todas as simetrias de cada semi-aresta.
  - A face_esquerda é associado a cada semi-aresta percorrendo entre todas os vértices de todas as faces.
  - A semi-aresta próximo e anterior, aproveitando o processamento da face_esquerda, é obtido as informações junto no processamento.

É notável que as informações de semi-aresta, ou seja, simetria, semi-aresta próximo e anterior não precisou ser processada novamente, pois isso foi feito no pré-processamento na leitura da entrada das faces da malha, o que facilitou a montagem da DCEL.

5 Casos de Teste

Os casos de testes utilizados neste trabalho foram os seguintes:

5.1 aberta.txt

Desenha uma forma aberta, com uma entrada muito similar à fornecida pelo professor, porém com a face externa faltando.

```
10 6
3 6
0 4
3 5
6 4
1 3
5 3
2 1
4 1
1 0
5 0
1 2 5 3
1 3 6 4
4 6 8 10
10 8 7 9
9 7 5 2
3 5 7 8 6
```

5.2 auto_intersec.txt

Desenha uma forma simples, com uma face que contém auto-interseção, em formato de ampulheta.

```
4 2
0 0
2 2
0 2
2 0
1 2 3 4
4 3 2 1
```

5.3 auto_intersec.txt

Desenha uma forma simples, com uma face que contém auto-interseção, em formato de ampulheta.

```
4 2
0 0
2 2
0 2
2 0
1 2 3 4
4 3 2 1
```

5.4 entrada_enunciado.txt

A mesma entrada fornecida pelo professor no enunciado do trabalho. (Não será reproduzida aqui.)

5.5 intersec_dentro.txt

Desenha dois quadrados, onde um deles está completamente dentro do outro.

```
8 4
0 0
4 0
4 4
0 4
1 1
2 1
2 2
1 2
1 2 3 4
5 6 7 8
1 4 3 2
5 8 7 6
```

5.6 intersec_fora.txt

Desenha dois retângulos, que não estão propriamente dentro um do outro, mas que possui faces externas diferentes que se sobrepõem.

```
8 4
0 0
4 0
4 4
0 4
5 1
7 1
7 2
5 2
1 2 3 4
1 4 3 2
5 6 7 8
5 8 7 6
```

5.7 nao_sub_planar.txt

Desenha um quadrado composto de três triângulos, com faces que se sobrepõem nas semi-arestas.

```
4 4
0 0
1 0
1 1
0 1
1 2 3
3 4 1
1 3 2
1 4 3 2
```

Referências

[1] Wikipedia contributors. *Jordan curve theorem*. Disponível em https://en.wikipedia.org/wiki/Jordan_curve_theorem. Acessado em maio de 2024.

[2] Wikipedia contributors. *Shoelace formula*. Disponível em https://en.wikipedia.org/wiki/Shoelace_formula. Acessado em maio de 2024.

[3] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry – Algorithms and Applications*. Springer, third edition, 2008.