3
Uma Representação Econômica de Malhas Irregulares

Como já mencionado anteriormente pesquisadores em modelagem geométrica têm desenvolvido diversas representações para objetos sólidos em $\mathbb{R}^3$. Muitas estruturas de dados para malhas poligonais foram propostas. Todas elas tentam equilibrar o uso de memória com a eficiência dos algoritmos de busca para responder às relações de incidências e adjacências.

Malhas compostas só por triângulos ou só por quadrângulos são muito utilizadas em computação gráfica. Porém, em muitas aplicações de engenharia, principalmente na modelagem por elementos finitos, é muito comum a utilização de malhas híbridas compostas por triângulos e quadrângulos.

Neste capítulo será apresentada uma nova estrutura de dados topológica, chamada *CHalfEdge*, para representar superfícies compostas por triângulos e/ou quadrângulos. Trata-se de uma simplificação da estrutura de dados *HalfEdge* [11], com alto poder de expressão e baixo custo de armazenamento.

3.1
Representação CHalfEdge

A *CHalfEdge* é uma estrutura de dados concisa para superfícies compostas por triângulos e/ou quadrângulos. Utiliza-se do conceito de *semi-aresta* que representa a associação de uma face a uma de suas arestas.

Considere uma superfície combinatória orientável sem bordo $\mathcal{S}$ com NT triângulos, NQ quadrângulos, NE arestas e NV vértices.

Na *CHalfEdge* as semi-arestas, os vértices e as faces são indexadas por inteiros não negativos. Os índices dos vértices variam de 0 a $NV-1$, os dos

triângulos de 0 a $NT - 1$, e os dos quadrângulos de NT a $NT + NQ$.

A fronteira de cada triângulo é representada implicitamente por um ciclo orientado de 3 semi-arestas. Da mesma forma, um ciclo orientado de 4 semi-arestas definem a fronteira de um quadrângulo. Portanto, a superfície $\mathcal{S}$ terá $3NT + 4NQ$ semi-arestas.

A seguinte convenção é adotada para indexação das semi-arestas de $\mathcal{S}$:

- Os índices das semi-arestas que formam a fronteira de um triângulo de índice t são: $3t, 3t + 1$, e $3t + 2$.
- Os índices das semi-arestas que formam a fronteira de um quadrângulo de índice q são: $3NT + 4q, 3NT + 4q + 1, 3NT + 4q + 2$ e $3NT + 4q + 3$.

A Figura 5.3 ilustra uma superfície com forma de uma pirâmide com base quadrada. Ela é utilizada para exemplificar uma indexação válida para os vértices, para as semi-arestas e para as faces segundo a convenção acima.

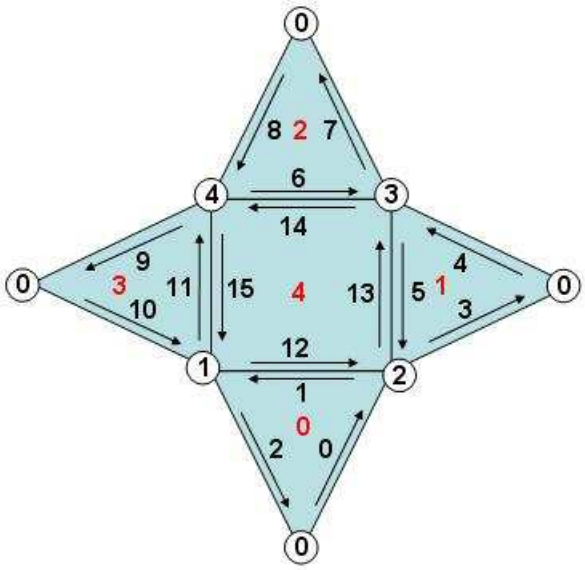


Figura 3.1: Representação Gráfica da *CHalfEdge*.

Essa convenção relaciona os índices das semi-arestas com os índices das faces de $\mathcal{S}$. De acordo com ela, é possível dizer quais são as semi-arestas de uma face. Por outro lado, dado o índice de uma semi-aresta, podemos definir através de um cálculo direto qual é o tipo (triângulo ou quadrângulo) e o índice da face à qual ela pertence. Mais precisamente, podemos dizer que uma semi-aresta com índice h pertence a um triângulo se $h \in [0, 3NT - 1]$, e que ela pertence a um quadrângulo se $h \in [3NT, 3NT + 4NQ - 1]$. Se

uma semi-aresta h pertence a um triângulo, então o índice desse triângulo é $t(h) = h \div 3$. Por outro lado, quando uma semi-aresta h pertence a um quadrângulo, então o índice desse quadrângulo é $q(h) = (h - 3NT) \div 4$. Vale notar que o operador $\div$ corresponde à operação de divisão por inteiros. Define-se que o índice geral de uma face f da superfície $\mathcal{S}$ é para um triângulo o seu próprio índice e para quadrângulos é o índice do quadrângulo adicionando de NT.

É sabido que cada face possui como fronteira um ciclo orientado de semi-arestas. Assim, essa convenção possui outra conseqüência importante, que é poder determinar através de uma fórmula simples qual é a semi-aresta sucessora e antecessora no ciclo da face.

Para uma semi-aresta h que pertence a um triângulo, os índices da sua semi-aresta sucessora ($n(h)$) e antecessora ($p(h)$) são, respectivamente, dados por:

$$n(h) = 3t(h) + (h + 1)\%3,$$
$$p(h) = 3t(h) + (h - 1)\%3.$$

Já, quando h pertence a um quadrângulo, tem-se que:

$$n(h) = 3NT + 4q(h) + (h + 1)\%4,$$
$$p(h) = 3NT + 4t(h) + (h - 1)\%4.$$

A operação $a\%b$ corresponde ao resto da divisão por inteiros de a por b.

Para representar a conectividade entre as células da superfícies, a estrutura de dados *CHalfEdge* utiliza duas tabelas de inteiros, chamadas de tabela V e tabela M. Ambas são indexadas pelas semi-arestas de $\mathcal{S}$. Portanto, a dimensão dessas duas tabelas é igual a $3NT + 4NQ$.

A entrada h da tabela V indica qual é o vértice origem da semi-aresta com índice h. O vértice destino da semi-aresta h é facilmente obtido como vértice origem da sua semi-aresta sucessora $n(h)$. Ao percorrer o ciclo de semi-arestas de uma face todos os seus vértices podem ser obtidos.

De acordo com a definição de superfície combinatória sem bordo, cada aresta é compartilhada por exatamente duas faces. As semi-arestas que estão

em duas faces adjacentes e que compartilham a mesma aresta, são chamadas de simétricas. Como a superfície $\mathcal{S}$ é orientável, tem-se que semi-arestas simétricas possuem orientações opostas (Figura 3.2).

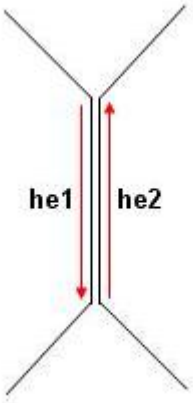
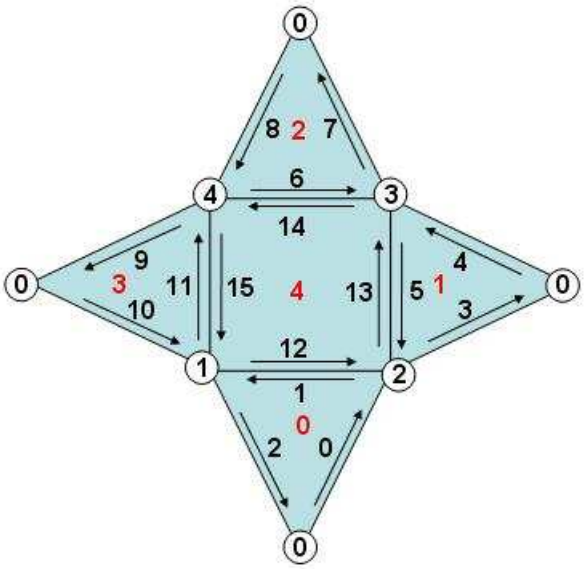


Figura 3.2: *Semi-arestas* simétricas.

Na tabela M (Figura 5.3) , cada entrada h armazena o índice da semi-aresta simétrica àquela com índice h. É fato que $M[M[h]] = h$, para todo h. Quando a semi-aresta h é associada a uma aresta de bordo, adota-se a convenção de que $M[h] = -1$.



V[0] = 0	M[0] = 3
V[1] = 2	M[1] = 12
V[2] = 1	M[2] = 10
V[3] = 2	M[3] = 0
V[4] = 0	M[4] = 7
V[5] = 3	M[5] = 13
V[6] = 4	M[6] =14
V[7] = 3	M[7] = 4
V[8] = 0	M[8] = 9
V[9] = 4	M[9] = 8
V[10] = 0	M[10] = 2
V[11] = 1	M[11] = 15
V[12] = 1	M[12] = 1
V[13] = 2	M[13] = 5
V[14] = 3	M[14] = 6
V[15] = 4	M[15] = 11

Figura 3.3: Tabelas da Estrutura de Dados *CHalEdge*.

A *CHalfEdge* utiliza uma terceira tabela G para armazenar os atributos geométricos dos vértices de $\mathcal{S}$, como, por exemplo, as coordenadas dos vértices em $\mathbb{R}^3$. A tabela G terá, portanto, NV entradas.

3.2
A classe CHalfEdge

Esta seção apresenta a classe (algoritmo 1) que implementa a representação *CHalfEdge*. Ela é baseada no que foi apresentado na seção anterior, e será descrita usando a linguagem $C++$.

```
class CHalfEdge
{
private:
   int NV;
   int NT;
   int NQ;
   int *M;
   int *V;
   Vector3D *G;
public:
   CHalfEdge();
   CHalfEdge(int nv, int nt, int nq);
   ~CHalfEdge();
   /* Métodos básicos */
   int nt();
   int nq();
   int nv();
   int n(int h);
   int p(int h);
   int facetype(int h);
   int t(int h);
   int q(int h);
   int f(int h);
   int v(int h);
   int m(int h);
   Vector3D g(int v);
   ...
}
```

Algoritmo 1: Classe CHalfEdge.

3.2.1
Os construtores e o destrutor

Para essa classe foram definidos dois construtores. O primeiro é o construtor padrão que inicia as variáveis `NV, NT` e `NQ` com o valor 0 e as

variáveis `M, V` e `G` como `NULL`. O segundo construtor recebe como dados de entrada o número de vértices, o número de triângulos e o número de quadrângulos que existem na superfície, e assim atribui, respectivamente, esses valores às variáveis `NV, NT` e `NQ`. Por fim, ele aloca na memória as tabelas `M, V` e `G`. Segue a implementação desses dois construtores (algoritmo 2 e 3).

```
CHalfEdge:: CHalfEdge()
{
  NV = NT = NQ = 0;
  M = V = G = NULL;
};
```

Algoritmo 2: Construtores da classe CHalfEdge.

```
CHalfEdge:: CHalfEdge(int nv, int nt, int nq)
{
  NV = nv; NT = nt; NQ = nq;
  M = new int [3*nt + 4 *nq];
  V = new int [3*nt + 4 *nq];
  G = new Vector3D [nv];
};
```

Algoritmo 3: Construtores da classe CHalfEdge.

O destrutor (algoritmo 4) simplesmente libera a memória alocada para os vetores `M, V` e `G`.

```
CHalfEdge:: ~CHalfEdge()
{
  if (M != NULL) delete [] M ;
  if (V != NULL) delete [] V ;
  if (V != NULL) delete [] G ;
};
```

Algoritmo 4: Destrutor da classe CHalfEdge.

3.2.2

Implementação dos métodos

Nessa seção são descritos os algoritmos que implementam as diversas funções definidas na seção anterior, assim como o acesso às tabelas `M, V` e `G`, visto que elas são variáveis privadas da classe.

Todos os métodos a serem apresentados nessa seção provocam o término da execução do programa se houver uma tentativa de acesso à informação através de um índice que esteja fora do domínio.

Seguem os algoritmos que para uma dada semi-aresta h obtêm o índice da semi-aresta sucessora $n(h)$ e antecessora $p(h)$ no ciclo de sua face(algoritmo 5 e 6).

```
int CHalfEdge:: n(int h)
{
  if ( (h >= 0) && (h < 3*NT) )
    return 3*(h/3) + (h+1) % 3;
  if ( (h >= 3*NT) && (h < 3*NT+4*NQ) )
    return 3*NT + 4*((h-3*NT)/4) + ((h-3*NT)+1) % 4;
  else exit(-1);
};
```

Algoritmo 5: Semi-aresta sucessora.

```
int CHalfEdge:: p(int h)
{
  if ( (h >= 0) && (h < 3*NT) )
    return 3*(h/3) + (h-1) % 3;
  if ( (h >= 3*NT) && (h < 3*NT+4*NQ) )
    return 3*NT + 4*((h-3*NT)/4) + ((h-3*NT)-1) % 4;
  else exit(-1);
};
```

Algoritmo 6: Semi-arestas antecessora.

A função `int facetype(int h)` define se uma semi-aresta `h` pertence a um triângulo ou a um quadrângulo. Ela retorna o número de semi-arestas que existem na face de `h`, ou seja: 3 ou 4 (algoritmo 7).

```
int CHalfEdge:: facetype(int h)
{
  if ( (h >= 0) && (h < 3*NT) )
    return 3;
  if ( (h >= 3*NT) && (h < 3*NT+4*NQ) )
    return 4;
  else exit(-1);
};
```

Algoritmo 7: Tipo de face de uma semi-aresta.

Para uma dada semi-aresta `h`, o método `t(h)` identifica o índice do triângulo dessa semi-aresta, e o método `q(h)` o índice do quadrângulo. Já a rotina `f(h)` retorna o índice geral da face de `h` (algoritmos 8, 9 e 10).

```
int CHalfEdge:: t(int h)
{
  if ( (h >= 0) && (h < 3*NT) )
    return h / 3 ;
  else
    exit(-1);
};
```

Algoritmo 8: Índice de um triângulo.

```
int CHalfEdge:: q(int h)
{
  if ( (h >= 3*NT) && (h < 3*NT+4*NQ) )
    return (h - 3*NT) / 4 ;
  else
    exit(-1);
};
```

Algoritmo 9: Índice de um quadrângulo.

```
int CHalfEdge:: f(int h)
{
  if ( facetype(h) == 3 )
    return t(h) ;
  else
    return NT + q(h);
};
```

Algoritmo 10: Índice de uma face.

Os métodos para acessar as variáveis privadas da classe: `nt, nq` e `nv` retornam, respectivamente, o número de triângulos, o número de quadrângulos e o número de vértices da superfície (algoritmos 11, 12 e 13).

```
int CHalfEdge:: nt()
{
  return NT;
};
```

Algoritmo 11: Número de triângulos.

```
int CHalfEdge:: nq()
{
  return NQ;
};
```

Algoritmo 12: Número de quadrângulos.

```
int CHalfEdge:: nv()
{
  return NV;
};
```

Algoritmo 13: Número de vértices.

Já os acessos às tabelas privadas `M, V` e `G` da classe, são obtidos, respectivamente, através das funções `m, v` e `g` (algoritmos 14, 15 e 16).

```
int CHalfEdge:: m(int h)
{
  if ( (h >= 0) && (h < 3*NT+4*NQ) )
    return M[h];
  else
    exit(-1);
};
```

Algoritmo 14: Acesso a tabela M.

```
int CHalfEdge:: v(int h)
{
  if ( (h >= 0) && (h < 3*NT+4*NQ) )
    return V[h];
  else
    exit(-1);
};
```

Algoritmo 15: Acesso a tabela V

```
Vector3D CHalfEdge:: g(int v)
{
  if ( (v >= 0) && (v < NV) )
    return G[v];
  else
    exit(-1);
};
```

Algoritmo 16: Acesso a tabela G.

3.3
Relações de Incidências e Adjacências

Nesta seção mostra-se como a estrutura de dados *CHalfEdge* pode ser utilizada para obter relações de adjacências e incidências, identificando assim a conectividade entre as células da superfície $\mathcal{S}$.

Foi visto na seção anterior que a estrutura de dados *CHalfEdge* utiliza uma representação intrínseca para as faces. Essa é uma das principais diferenças dela para a *HalfEdge* original. As arestas, por sua vez, são representadas implicitamente. Duas semi-arestas simétricas representam uma aresta do interior, e uma semi-aresta de bordo representa uma aresta de bordo.

A semi-aresta h é associada a uma aresta e de $\mathcal{S}$ cujos vértices extremos são $V[h]$ e $V[n(h)]$. Se a semi-aresta h é do interior, então a simétrica de h também será associada à mesma aresta e.

Em certos tipos de aplicações são necessárias muitas buscas nas arestas. Nesses casos, é adequado ter uma representação explícita para arestas. Para isso, duas novas tabelas, chamadas de HE e EH, podem ser facilmente construídas em tempo linear a partir da tabela M.

A tabela HE representa a associação de uma semi-aresta a uma aresta, ou seja, ela responde para uma dada semi-aresta h qual é o índice da aresta associada. O número de entradas da tabela HE é $3NT + 4NT$.

Por outro lado, a tabela EH representa a associação de uma aresta a uma semi-aresta. Com ela pode-se obter para uma dada aresta e, o índice de uma semi-aresta associada a ela. O número de entradas da tabela EH será igual ao número de arestas da superfície.

Note que para verificar a consistência dessas tabelas, deve sempre ser verdadeiro que $EH[HE[h]] = h$ ou $EH[HE[h]] = M[h]$ e que $HE[EH[e]] = e$.

O número de arestas de uma superfície é igual ao número de semi-arestas que estão associadas a número de arestas do interior dividido entre dois mais o número de arestas de bordo. Portanto, no caso em que a superfície $\mathcal{S}$ não possua bordo, o número de arestas de $\mathcal{S}$ é $(3NT+4NT)/2$.

Na Figura 3.4 a aresta a_1 é interior à malha, portanto $EH[a_1] = b$ ou $EH[a_1] = s$, enquanto a aresta a_2 é uma aresta de bordo e assim

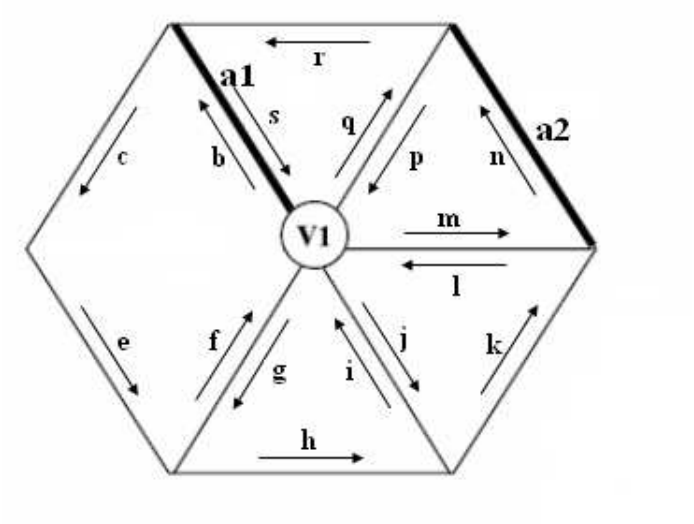


Figura 3.4: A estrela de uma vértice.

$EH[a_2] = n$.

Ainda na Figura 3.4, pode-se notar que o vértice origem das semi-arestas b, q, m, j, g é v_1. Em várias aplicações é muito útil poder identificar para cada vértice da superfície, pelo menos uma das semi-arestas que possui esse vértice como origem. Para isso, poder-se-ia alocar uma outra tabela de tamanho NV, denotada por VH, que em cada entrada v armazena o identificador de uma semi-aresta h tal que $V[h] = v$.

É importante observar que as tabelas EH, HE e VH não foram criadas diretamente na classe porque elas não são necessárias nos algoritmos de compressão e descompressão que posteriormente serão introduzidos, como também em muitos outros algoritmos de computação gráfica. Além disso, representa uma boa economia de memória sem muitos prejuízos para a eficiência dos algoritmos de busca para a obtenção das relações de incidência e adjacência. Porém todas elas são facilmente criadas em tempo linear no número de semi-arestas.

3.3.1

Vizinhança de um Vértice

Dado um vértice com índice v, em várias aplicações é muito importante percorrer a estrela de v obtendo, por exemplo, todos os vértices adjacentes a ele em tempo $O(deg(v))$, onde $deg(v)$ é o número de vértices adjacentes a v. O algoritmo 17 gera uma lista orientada de vértices adjacentes ao

vértice $V[h]$, para uma dada semi-aresta h. Nele utilizou-se a classe *List* da biblioteca STL (*Standard Template Library*) do C++.

Quando é necessária a inserção e/ou remoção de dados várias vezes no programa, uma das soluções é a utilização da classe *List* da biblioteca STL. A classe *List* implementa uma lista duplamente encadeada.

algoritmo `VizinhancaVertice`()
1 $j = h$; $v = V[h]$;
2 $List < int > vV$;
3 do
4 $vV.insert(V[next(j)])$;
5 $j = p(M[j])$;
6 while $(j! = h)$
fim

Algoritmo 17: Vizinhança de um vértice.

A mesma estratégia pode ser usada para obter uma lista de faces e de arestas que possuem v como vértice.

3.3.2
Vizinhança de uma Aresta

Dada uma semi-aresta he associada a uma aresta e, as faces que possuem e como aresta são: a face de he e a face de $M[he]$ quando a aresta e é do interior, e somente a de he quando ela estiver no bordo. Os vértices da aresta associada a he são $V[he]$ e $V[n(he)]$.

3.3.3
Vizinhança de uma Face

Para obter, por exemplo o ciclo orientado de vértices de uma face associada a semi-aresta h pode-se utilizar o seguinte algoritmo.

Esse mesmo algoritmo pode ser utilizado para obter as arestas e as faces adjacentes de uma face.

```
algoritmo VizinhancaFace()
    1 j = h; f = f(h);
    2 List < int > fV;
    3 do
    4    fV.insert(V[j]);
    5    j = n(j);
    6 while (j! = h)
fim
```

Algoritmo 18: Vizinhança de uma face.

3.4
Estrutura de Arquivo dos Modelos 3D

Para codificar malhas é preciso primeiro instânciar a superfície, o que pode ser feito através de um arquivo. É muito comum encontrar arquivos de malhas $3D$ disponíveis na Internet, nos mais diferentes formatos, os quais fornecem informações para uma construção direta das tabelas V e G. A tabela M é, por sua vez, gerada a partir da tabela V.

Segue a descrição da estrutura do arquivo de entrada de um determinado modelo.

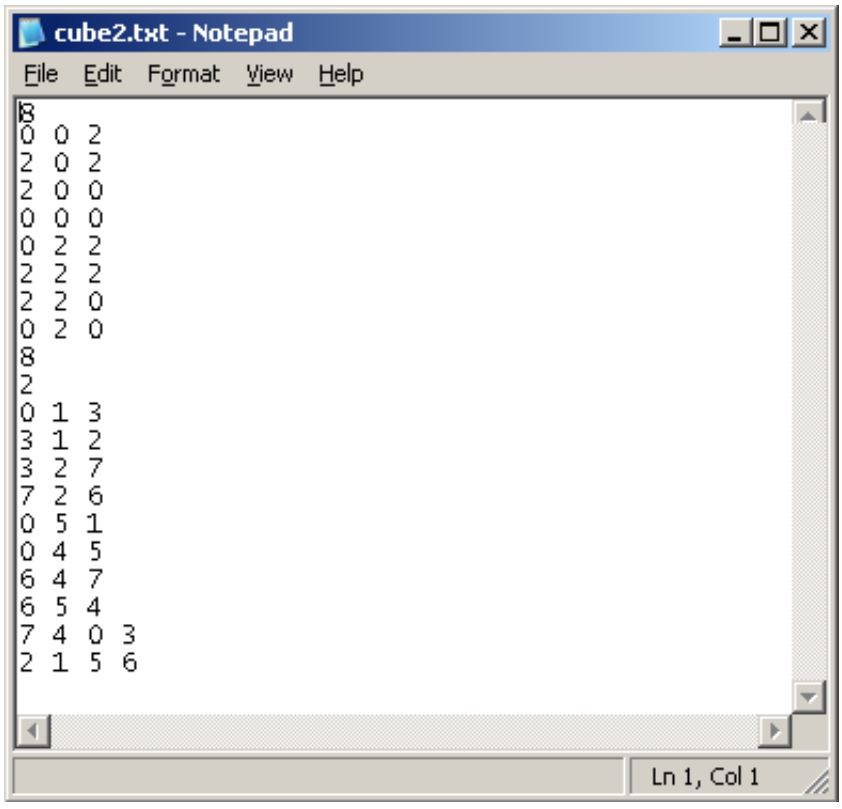


Figura 3.5: Modelo de dados de entrada.

A Figura 3.5 mostra o modelo de entrada de um cubo que contem uma malha que possui tanto triângulos como quadrângulos.

A primeira linha do arquivo de entrada indica o número NV de vértices da superfície. Após esta linha seguem NV linhas com as coordenadas dos vértices. Os vértices são indexados naturalmente através da ordem da leitura.

Após a leitura dos vértices, indica-se com dois números inteiros consecutivos a quantidade de triângulos, NT, e quantidade de quadrângulos, NQ. Após esses número seguem $NT+NQ$ linhas, cada uma contendo a lista orientada de vértices para cada face. As NT primeiras linhas correspondem aos triângulos e as NQ seguintes aos quadrângulos.

Portanto, as tabelas V e G são diretamente construídas com essas informações do arquivo. O passo seguinte será o de construir a tabela M. Isso é feito através da chamada à rotina `ComputeTableM()`, que será descrita na próxima subseção.

Para resumir, segue o algoritmo que lê a superfície de uma determinada malha no formato do arquivo de entrada especificado nos parágrafos anteriores.

```
algoritmo void CHalfEdge::ReadData(istream s)
 1 //Lê número de vértices da malha
 2 s >> NV;
 3 //Aloca memória para o vetor de geometria
 4 G = new Vector3D[NV];
 5 //Para cada vértice da malha, lê coordenadas x, y e z
 6 for ( i = 0 ; i < NV ; i++ )
 7 {
 8   s >> G[i][0] >> G[i][1] >> G[i][2];
 9 }
10 //Lê número de triângulos da malha
11 s >> NT;
12 //Lê número de quadrângulos da malha
13 s >> NQ;
14 //Aloca memória para vetor de triângulos
15 V = new int[3*NT+4*NQ];
16 //Aloca memória para verter de quadrângulos
17 M = new int[3*NT+4*NQ];
18 //Para cada triângulo da malha, lê topologia
19 for ( i = 0 ; i < NT ; i++ )
20 {
21   s >> V[3*i] >> V[3*i + 1] >> V[3*i + 2];
22 }
23 //Para cada quadrângulo da malha, lê topologia
24 for ( i = 0 ; i < NQ ; i++ )
25 {
26   s >> V[3*NT + 4*i] >> V[3*NT + 4*i + 1] >> V[3*NT
   + 4*i + 2] >> V[3*NT + 4*i + 3];
27 }
28 //Calcula tabela de adjacências e incidências
29 ComputeTableM();
fim
```

Algoritmo 19: Leitura de dados.

3.5
Construção da Tabela M

O algoritmo 20[1] tem por objetivo construir a tabela M a partir da tabela V. Esse algoritmo usa a estrutura `map` da biblioteca STL do C++.

A estrutura `map` pode ser entendido como um array associativo em que a chave pode ser de qualquer tipo, ao contrário do array normal, em que a chave deve ser do tipo inteiro. Para cada chave K, pode-se armazenar um valor V, sendo K e V de qualquer tipo, inclusive tipos definidos pelo programador. No `map` existe apenas uma chave K, não ocorrendo duplicação.

O construção da tabela M é feita usando-se a noção de par ordenado da STL. Nas linhas 9 e 11 obtém-se os vértices de uma aresta da superfície. Essa aresta é orientada sempre do menor índice de vértice para o de maior índice (linha 12 até a linha 18). Esse par ordenado de índices é que representa a aresta orientada. Cada novo par ordenado novo é incluído numa lista, onde o índice da lista é o oposto de uma semi-aresta. O uso desse par ordenado facilita a busca na estrutura `map`. Para cada par ordenado, ou aresta orientada, verifica-se se este está na lista. Caso este já esteja na lista, isso quer dizer que a aresta já havia sido visitada. Caso contrário, deve-se adicionar este novo par ordenado na lista e procurar pelo seu oposto. A cada par ordenado cujo oposto é encontrado elimina-se o par ordenado da lista. Veja linhas 19 até 33 do algoritmo 20.

[1]Esse método é definido na classe *CHalfEdge* como `friend void ComputeTableM();` para ter acesso aos membros privados da classe.

```
algoritmo void ComputeTableM()
 1 //Vetor de pares ordenados usando STL do C++
 2 map<OrderedPair> adjacency;
 3 //Iterador de busca de posição no vetor de adjacências
 4 map<OrderedPair>::iterador pos;
 5 //Para cada halfedge h presente na malha ..
 6 for( h = 0 ; h < 3*NT + 4*NQ ; h++ )
 7 {
 8 //Obtem vértice da halfedge corrente
 9   i = V[h];
10 //Obtem vértice da halfdege seguinte
11   j = V[next(h)];
12 //Ordena par ordenado
13   if (j < i)
14   {
15     tmp = i;
16     i = j;
17     j = tmp;
18   }
19 //Procura par ordenado no vetor de adjacências
20   pos = adjacency.find(<OrderedPair>(i,j));
21 //Se par ordenado existir no vetor de adjacências
22 //foi encontrado o oposto a uma halfedge h
23   if (pos != adjacency.end())
24   {
25     M[h] = pos->second;
26     M[M[h]] = h;
27     adjacency.erase(pos);
28   }
29 //Adiciona novo par ordenado no vetor de adjacências
30   else
31   {
32     adjacency[<OrderedPair>(i,j)] = h;
33   }
34 }
fim
```

Algoritmo 20: Construção da tabela M.

3.6 Conclusão

Este Capítulo apresentou uma nova estrutura de dados com baixo custo de armazenamento e alto poder de expressão para representar superfícies compostas por triângulos e/ou quadrângulos. Ela substitui várias

informações que eram armazenadas na estrutura *HalfEdge*, proposta por Mäntylä, por funções definidas sobre inteiros. Obtendo assim, uma representação intrínseca para faces e implícita para as arestas.

No próximo Capítulo serão utilizados os conceitos aqui apresentados para mostrar que é possível obter uma extensão do algoritmo Edge-Breaker [14]. Esta nova extensão será capaz de codificar malhas compostas por triângulos e/ou quadrângulos com ou sem alças.