# Uma estrutura de dados dinâmica para geração de malhas de Delaunay bidimensionais e tridimensionais

Mauro Massayoshi Sakamoto
Pesquisador do LMAG – PEA-EPUSP.
São Paulo – SP [Brasil]
mauro@pea.usp.br

José Roberto Cardoso
Professor do LMAG – PEA-EPUSP.
São Paulo – SP [Brasil]
cardoso@pea.usp.br

José Márcio Machado
Professor do DDCE – IBILCE-Unesp.
São José do Rio Preto – SP [Brasil]
jmarcio@ibilce.unesp.br

Neste trabalho, é apresentada a implementação de um algoritmo para geração de malhas de Delaunay triangulares e tetraédricas, por meio de um conjunto de estruturas de dados e métodos genéricos que permitem redimensionar dinamicamente o tamanho de cada uma dessas estruturas, produzindo classes e métodos tanto de ordem 2 quanto de ordem 3, conforme a dimensão do problema em estudo. A idéia principal desse algoritmo é proporcionar a geração de malhas bidimensionais e tridimensionais, sem a necessidade de dois algoritmos ou, ainda, de estruturas de armazenamentos complexas, eliminando, assim, a redundância de código. A metodologia apresentada foi desenvolvida com o auxílio dos contêineres da Standard Template Library (STL), os quais apresentam as ferramentas necessárias para o desenvolvimento desse algoritmo.

**Palavras-chave:** Algoritmo de Delaunay. Estrutura de dados. Geração de malhas bidimensionais e tridimensionais. Método dos elementos finitos. STL.

## 1 Introdução

Atualmente, o grande avanço tecnológico proporciona, cada vez mais, o acesso a computadores de grande capacidade de processamento e armazenamento para resolução de problemas, principalmente em modelos físicos complexos e fenômenos biológicos, uma vez que poucos desses problemas têm solução analítica exata.

Uma técnica bastante utilizada na busca por soluções é o método dos elementos finitos, tanto bidimensionais quanto tridimensionais.

No entanto, uma exigência fundamental para um programa que empregue o método dos elementos finitos é a capacidade de subdividir o domínio geométrico, de forma consistente, em simplexos (triângulos, no caso bidimensional, e tetraedros, no tridimensional). Essa subdivisão do domínio é feita com a aplicação de um método de geração de malhas.

Nesse contexto, uma técnica muito citada na literatura é o algoritmo bidimensional e tridimensional de Delaunay, cujo critério consiste em construir elementos de forma que não exista nenhum vértice do poliedro no interior da esfera circunscrita de outro poliedro (OWEN, 1998), ou seja, a idéia central desse algoritmo é que nenhum elemento possua pontos dentro do seu circuncírculo em problemas bidimensionais e circunsfera em tridimensionais. Essa propriedade maximiza o menor ângulo do poliedro, tornando-o bastante satisfatório para geometrias regulares, não regulares e multiplamente conexas (SAKAMOTO, 2007; MARRETTO; MACHADO, 1998; SHEWCHUK, 1996; RUPPERT, 1995).

A adequada implementação para os algoritmos descritos pode ser realizada, utilizando-se a biblioteca Standard Template Library (STL), que foi adicionada à linguagem C++ no fim de 1998 (ZHU; LUBKEMAN, 1997). A STL tem um conjunto de recursos otimizados para linguagem C++, tais como contêineres, iteradores, algoritmos, classes genéricas e polimorfismo em tempo de execução, fornecendo, assim, grande auxílio para o desenvolvimento de *softwares*.

Valendo-se de toda a flexibilidade das estruturas de dados e algoritmos otimizados da STL, neste trabalho, apresenta-se um conjunto de classes de objetos e métodos genéricos, que armazenam e manipulam os elementos necessários à construção de uma malha tanto bidimensional quanto tridimensional.

## 2 O algoritmo de Delaunay

Malha de Delaunay de um conjunto de pontos é uma triangulação (tetraedrização), em que cada elemento satisfaz à propriedade de circuncírculo (circunsfera) vazio, ou seja, não contém pontos em seu interior, fator que maximiza o menor ângulo do elemento.

A técnica escolhida para modelagem do algoritmo de Delaunay foi a implementação de Bowyer-Watson, uma das mais citadas na literatura (SCHROEDER; SHEPHARD, 1998; WATSON, 1981; OWEN, 1998).

Diferentemente da maioria das técnicas que iniciam com um elemento contendo o novo ponto, o algoritmo de Watson parte de uma malha inicial que engloba todos os pontos da geometria; assim, novos elementos são formados à medida que esses pontos vão sendo introduzidos.

As etapas para construção da malha bi e tridimensional são similares, e as poucas diferenças se referem à dimensão das estruturas de armazenamento e cálculo do circuncírculo (circunsfera). Os demais métodos e cálculos envolvidos podem ser facilmente adaptados para trabalhar de acordo com a dimensão do problema.

### a. Etapas de implementação

1. Construir uma lista com as coordenadas dos **n** pontos que descrevem o domínio. Essas coordenadas são pares ordenados para a dimensão de ordem 2, e triplas, para a de ordem 3.

2. Determinar um quadrilátero (hexaedro) englobante a partir do *extent* – cálculo das coordenadas máximas e mínimas dos **n** pontos – do domínio.
3. Decompor o quadrilátero em dois triângulos com os **n** pontos espalhados por eles (6 tetraedros para o hexaedro).
4. Os pontos da lista são inseridos no *extent*. Calculando-se o circuncírculo (circunsfera) da malha inicial, eliminam-se os elementos que incluem o ponto, gerando a partir dos restantes e do novo ponto.
5. Após a inclusão de todos os pontos, são eliminados os elementos com vértices que coincidem com os do *extent*.

Os Quadros 1 e 2 ilustram os algoritmos bi e tridimensional, respectivamente.

```
PARA i=1 ATÉ número de pontos
    PARA j=1 ATÉ número de triângulos
        Calcular o circuncírculo C_j do triângulo T_j
        SE C_j contiver o ponto i, ENTÃO
            Adicionar T_j para a lista de arestas
            Marcar T_j para remoção da lista de triângulos
        FIM SE
    FIM PARA
Apagar todos os triângulos marcados para remoção.
Remover todas as arestas comuns da lista de arestas.
Construir novos triângulos a partir do ponto i e da lista
de arestas. Adicionar os novos elementos para a lista de
triângulos.
FIM PARA.
```

**Quadro 1: Algoritmo para a triangulação de Delaunay**

Fonte: Os autores.

```
PARA i=1 ATÉ número de pontos
    PARA j=1 ATÉ número de tetraedros
    Calcular a circunsfera C_j do tetraedro T_j
        SE C_j contiver o ponto i ENTÃO
            Adicionar T_j para a lista de faces
            Marcar T_j para remoção da lista de tetraedros
        FIM SE
    FIM PARA
Apagar todos os tetraedros marcados para remoção. Remover todas as faces comuns da lista de faces. Construir
novos tetraedros a partir do ponto i e da lista de faces.
Adicionar os novos elementos para a lista de tetraedros.
FIM PARA.
```

**Quadro 2: Algoritmo para a tetraedrização de Delaunay**

Fonte: Os autores.

### b. Comparação dos algoritmos

Comparando os algoritmos, nota-se certa equivalência nas estruturas e métodos empregados em suas construções. A dimensão das estruturas é diferente; no entanto, grande parte das principais rotinas se refere à manipulação das listas de dados, o que facilita a generalização do gerador de malhas. A grande exceção fica a cargo do teste de inclusão do novo ponto, que, em duas dimensões, é realizado pelo cálculo do circuncírculo e, em três dimensões, pelo cálculo da circunsfera.

Um paralelo entre as principais estruturas e rotinas é mostrado na Tabela 1.

**Tabela 1: Equivalência entre os métodos**

| Malha triangular | Malha tetraédrica |
|---|---|
| Par ordenado | Tripla ordenada |
| Aresta | Faces |
| Triângulo | Tetraedro |
| Calcula "exent2d" | Calcula "extent3d" |
| Calcula circuncírculo | Calcula circunsfera |
| Adiciona aresta | Adiciona face |
| Apaga a aresta comum | Apaga a face comum |
| Calcula malha de triângulos | Calcula malha de tetraedros |

Fonte: Os autores.

## 3 Estrutura de armazenamento e funções

O algoritmo de Delaunay bi e tridimensional pode ser implementado, usando somente os recursos de programação orientada a objetos presentes na linguagem C++, tais como classe, herança, polimorfismo, ponteiro e lista ligada.

Dessa forma, são criadas as seguintes superclasses: Coordinates, Vertex, Elements, Mesh, Delaunay, as classes Edge e Face, além das derivadas Coordinate2D, Coordinate3D, Vertex2D, Vertex3D, Triangle, Tetrahedra, Mesh2D, Mesh3D, Delaunay2D e Delaunay3D, como mostra a Figura 1.

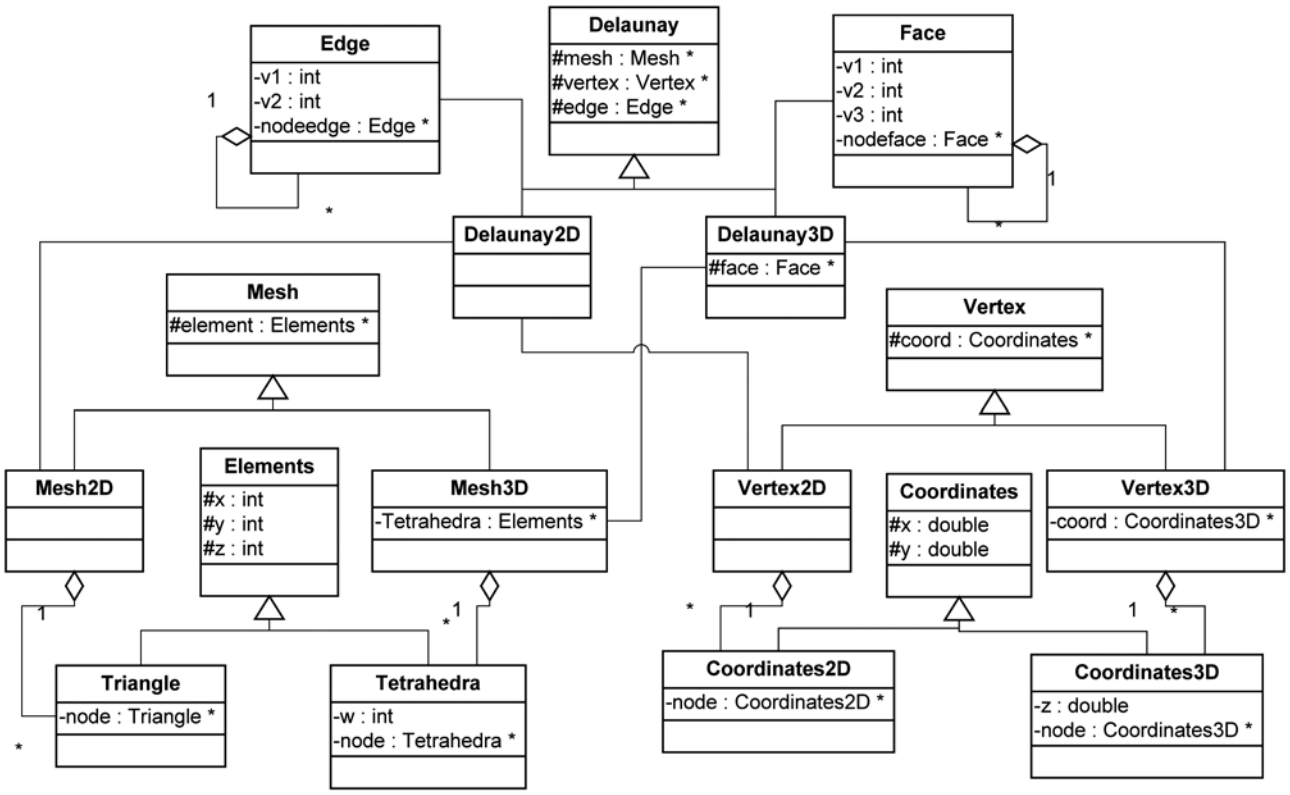



**Figura 1: Diagrama de classes com herança e lista ligada**
Fonte: Os autores.

Outra construção para o malhador genérico pode ser feita de modo mais simplificado, usando-se apenas listas ligadas, sem herança. Com isso, as classes Elements e Coordinates passariam a instanciar as novas classes NodeElto e NodeCoord, respectivamente (Figura 2).

Cada classe seria mais uma lista ligada e armazenaria apenas um valor *int*, no caso de elemento, e *double*, no caso de coordenada. Assim, para uma dimensão *d*, cada instância da classe Elements associaria *d*+1 instâncias da classe NodeElto, e a classe Coordinates associaria *d* instâncias do tipo NodeCoord.

Já as classes Edge e Face seriam substituídas pela Shape, que similarmente associaria a classe NodeElto com *d* instâncias.

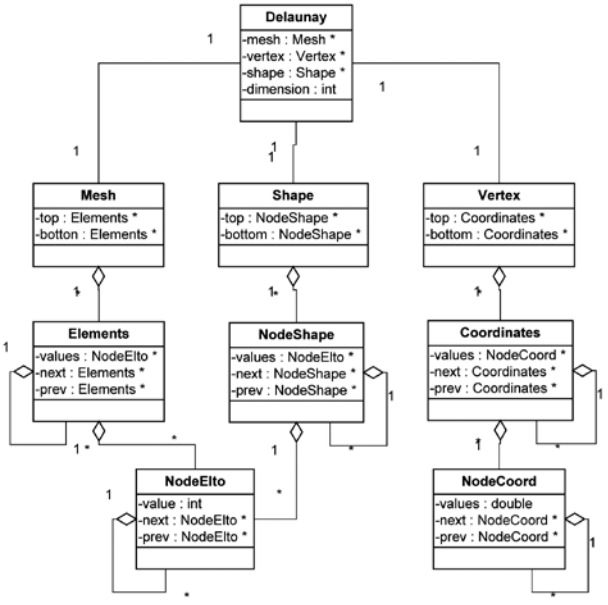



**Figura 2: Diagrama de classes apenas com lista ligada**
Fonte: Os autores.

### a. Modelagem das estruturas de armazenamento com STL

Outra construção ainda mais simplificada para implementação do algoritmo é generalizá-lo por meio da biblioteca STL. Isso é possível a partir da remodelagem das classes já apresentadas com a utilização do *vector*, que permite alocar dinamicamente a dimensão dos atributos das classes. A Figura 3 ilustra essas alterações.

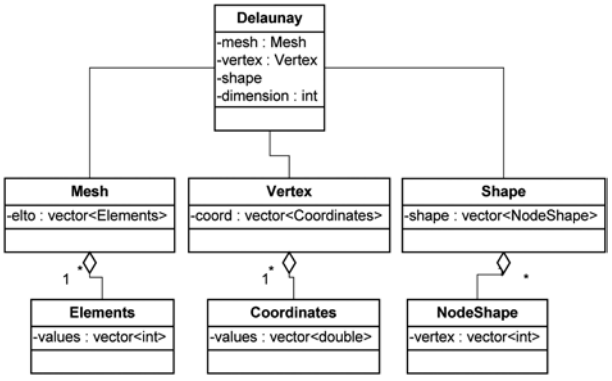



**Figura 3: Diagrama de classes com o *vector* da STL**
Fonte: Os autores.

Essa construção é bem parecida com a estrutura apresentada na Figura 2; no entanto, o *vector* e a STL oferecem uma série de facilidade, com métodos próprios, para manipulação de contêineres (JOSUTTIS, 2003), fatores que diminuem consideravelmente o tamanho do código e facilitam sua manutenção. A Tabela 2 ilustra algumas das principais operações do *vector* e da STL.

Uma malha triangular manipula triângulos, arestas e coordenadas bidimensionais. Assim, deve-se dimensionar os atributos das classes Elements, Shape e Coordinates com os valores 3, 2 e 2, respectivamente.

Em três dimensões, esses valores são 4, 3, 3, respectivamente, representando tetraedros, faces e coordenadas tridimensionais. Na Figura 4, os atributos das classes Elements, NodeShape e Coordinates são do tipo vector de ordem *d*, *d*-1 e *d*-1, respectivamente, em que *d* é a dimensão do problema.

**Tabela 2: Operações-padrão do *vector***

| Operação | Efeito |
|---|---|
| vector<Elem> v(*n*) | Cria um vetor v com n elementos |
| v.insert(*pos, elem*) | Insere elemento *elem* na posição *pos* de v |
| v.push_back() | Adiciona um elemento no fim de v |
| v.erase(*pos*) | Remove o elemento da posição *pos* de v |
| v.resize(*num*) | Altera a dimensão de v para *num* |
| v[*idx*] | Retorna o elemento de v com o índice *idx* |
| find_if(*beg, end, value*) | Retorna a posição do primeiro elemento de v no intervalo [*beg, end*), com valor igual a *value* |

Fonte: Os autores, com base em Josuttis, 2003, p. 148-159.

No teste de inclusão do novo ponto, verifica-se se o circuncentro (ou a circunsfera) não está vazio, em duas e três dimensões, respectivamente (Figura 5).

### b. Adaptação dos métodos com STL

A Tabela 2 apresenta os principais métodos para construção de uma malha de Delaunay. Essas funções podem ser generalizadas com os recursos otimizados da STL.

#### 1) *Extent*

O *extent* tem o objetivo de calcular o quadrilátero (hexaedro) que envolverá todos os pontos do domínio, por meio das coordenadas máximas e mínimas, para, então, calcular-se o *extent* com acréscimo de 50% em seus valores.

A partir da construção de uma lista de tamanho 2* dimensão, são armazenados nas posições pares os valores máximos, e nos ímpares, os mínimos das coordenadas, isto é, {*X, x, Y, y*} ou {*X, x, Y, y, Z, z*}, sendo *X, Y, Z* os valores máximos das coordenadas, e *x, y, z*, os mínimos.

Dessa forma, para i=0,1...,n e j=0,1,...*d*-1, temos:

$$limit[2*j] = coord[i][j] \; se \; limit[2*j] < coord[i][j] \quad (1)$$

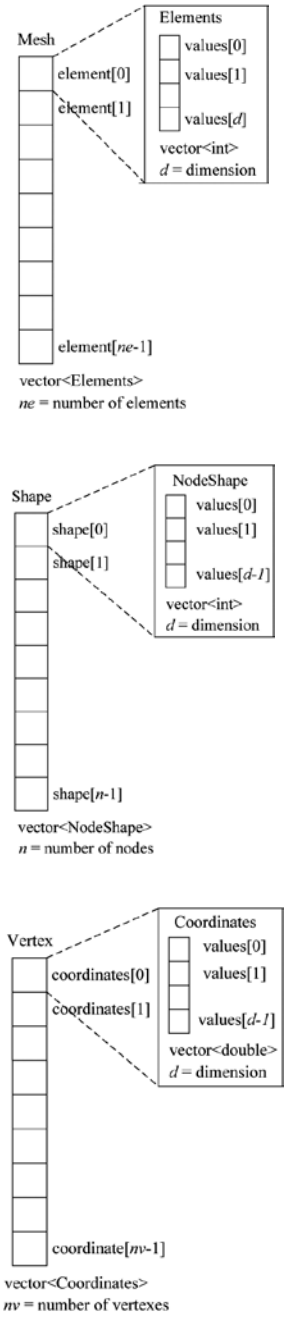



**Figura 4: Estrutura de armazenamento das classes**
Fonte: Os autores.

$$limit[2*j+1] = coord[i][j] \; se \; limit[2*j+1] < coord[i][j] \quad (2)$$

em que $d$ é a dimensão; *limit*, a lista de dimensão $2*d$ que armazena os máximos e mínimos das coordenadas, e *coord*, a lista de pontos de tamanho $n$.

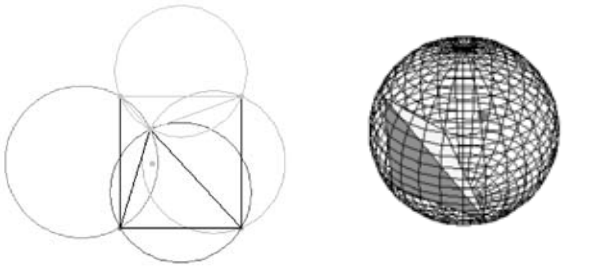


**Figura 5: Teste de inclusão do ponto em 2D e 3D, respectivamente**
Fonte: Os autores.

O próximo passo é combinar esses valores para encontrar as coordenadas do *extent*. Padronizando, de forma adequada, a lista dos $x$ e $y$, é possível generalizar parcialmente a montagem. O padrão adotado é dado por {{$x$, $Y$}, {$x$, $y$}, {$X$, $y$}, {$X$, $Y$}}.

No caso bidimensional, esses valores são o próprio *extent*. Para o tridimensional, é necessário replicá-los, ou seja, {{$x$, $Y$}, {$x$, $y$}, {$X$, $y$}, {$X$, $Y$}, {$x$, $Y$}, {$x$, $y$}, {$X$, $y$}, {$X$, $Y$}}. Por fim, são adicionados os valores de $z$ na metade inicial do *extent* e $Z$ no final (apenas para tetraedrização). Dessa forma, o *extent* tridimensional resulta em {{$x$, $Y$, $z$}, {$x$, $y$, $z$}, {$X$, $y$, $z$}, {$X$, $Y$, $z$}, {$x$, $Y$, $Z$}, {$x$, $y$, $Z$}, {$X$, $y$, $Z$}, {$X$, $Y$, $Z$}}.

O quadro 3 apresenta o pseudocódigo para esse algoritmo.

```
{X, x, Y, y, Z, z} o conjunto das coordenadas máximas e
mínimas
d ordem do problema
PARA i=0 ATÉ d
    crd(0+4*i) ← {x,y}
    crd(1+4*i) ← {x, Y}
    crd(2+4*i) ← {X, Y}
    crd(3+4*i) ← {X, y}
FIM
SE d é 3, ENTÃO
    PARA i=0 ATÉ 3
        crd(i,2) ← Z
        crd(i+4,2) ← z
    FIM PARA
FIM SE.
```

**Quadro 3: Algoritmo para o cálculo do *extent***
Fonte: Os autores.

### 2) AddShape

O método AddShape constrói uma lista de arestas (faces) dos elementos envolvidos na inclusão do novo ponto, ou seja, para um elemento {*x*, *y*, *z*}, obtém-se as arestas *xy*, *yz*, *zx*. Similarmente, as faces *xyz*, *yzw*, *zwx*, *wxy* são geradas a partir do elemento {*x*, *y*, *z*, *w*}.

De forma genérica, é possível utilizar os índices do contêiner *vector* em conjunto com o operador % (retorna o resto da divisão) e a dimensão *d*+1, ou seja:

$$ind = (i + j) \ \% \ (d + 1) i = 0, 1, \ldots, d\text{-}1 \quad (3)$$

O Quadro 4 apresenta o algoritmo para o procedimento desenvolvido.

```
d dimensão do problema
S nó do tipo shape com dimensão d+1
T elemento selecionado
PARA i=0 ATÉ d
    PARA j=0 ATÉ d-1
        ind ← Resto ( (d+1)/ (i+j))
        S_j ← Tind
    FIM PARA
    Insere-se nó S para a lista de shapes
FIM PARA.
```

**Quadro 4: Algoritmo AddShape**

Fonte: Os autores.

### 3) DelComShape

DelComShape apaga as arestas (faces) comuns, determinando uma região para construção dos novos elementos. Isso é feito movendo-se as arestas (faces) não repetidas a uma nova lista. Novos elementos são gerados combinando essas arestas (faces) como índice do novo ponto inserido.

A generalização é simples, uma vez que o método trabalha, em sua maioria, com a manipulação das listas de elementos e arestas (faces) e as respectivas dimensões de seus atributos (Quadro 5).

```
d dimensão do problema
E nó do tipo elemento
id índice do novo ponto inserido
Apaga as shapes comuns
PARA i=1 ATÉ número de shapes
    PARA j=1 ATÉ d
        E_i ← S_ij
    FIM PARA
    E_d ← id
    Adiciona E para a lista de elementos
FIM PARA.
```

**Quadro 5: Algoritmo DelComShape**

Fonte: Os autores.

### 4) ComputeMesh

ComputeMesh é o principal método, cujo objetivo é percorrer a lista de elementos da malha e testá-los para a construção dos novos elementos.

A malha triangular se inicia na posição 5, pois as quatro primeiras são do *extent*; já para a tetraédrica, o ponto de partida é a posição 9. De forma genérica, temos: *inicio*=4*(*dimensão*-1)+1.

Em determinado estágio do método, é realizado o teste de inclusão do novo ponto. Uma matriz 3 x 3 ou 4 x 4 (conforme a dimensão) deve ser iniciada com os vértices do elemento, sendo, em seguida, calculados o circuncírculo e a circunsfera.

A construção da malha triangular difere da tetraédrica durante o teste de inclusão do novo ponto. Assim, não foi possível generalizar essa etapa do código. O Quadro 6 apresenta o pseudo-código do método proposto.

```
D dimensão do problema
PARA i=(4*(d-1)+1) ATÉ número de pontos
    PARA j=1 ATÉ número de elementos
        Inicia matriz A_(d+1,d+1) com vértices do elemento T_j
        SE d é 2 ENTÃO
            Calcula o circuncírculo C_j do triângulo T_j
        SENÃO
            Calcula a circusfera C_j do tetraedro T_j
        FIM SE
    FIM PARA
    SE C_j contém o ponto i ENTÃO
        AddShape(Tj)
        Marca T_j para remoção da lista de elementos
    FIM SE
    Apaga todos os elementos marcados para remoção.
    DelComShape(i)
    Adiciona os novos elementos para a lista de elementos na
malha.
FIM PARA
```

**Quadro 6: Algoritmo ComputeMesh**

Fonte: Os autores.

### 5) Circuncírculo e circunsfera

O circuncírculo e a circunsfera são os métodos matemáticos responsáveis pelo teste de inclusão do novo ponto. Ambos têm por objetivo encontrar o centro e o raio do círculo e esfera circunscritos (respectivamente), a partir dos vértices dos elementos.

As semelhanças entre eles param nesse ponto, uma vez que os cálculos matemáticos envolvidos são distintos, não sendo possível generalizá-los, por serem necessários dois métodos separados para cada dimensão. O pseudocódigo não será mostrado; apenas a idéia dos cálculos executados:

Seja o triângulo com vértice $x_i$, $y_i$, $i$ = 1, 2 e 3. O circuncírculo é calculado pelo seguinte determinante.

$$\begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_3 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0 \quad (4)$$

A circunsfera de um tetraedro $x_i$, $y_i$, $z_i$, $i$ = 1,..., 4 é dada pelo cálculo do determinante:

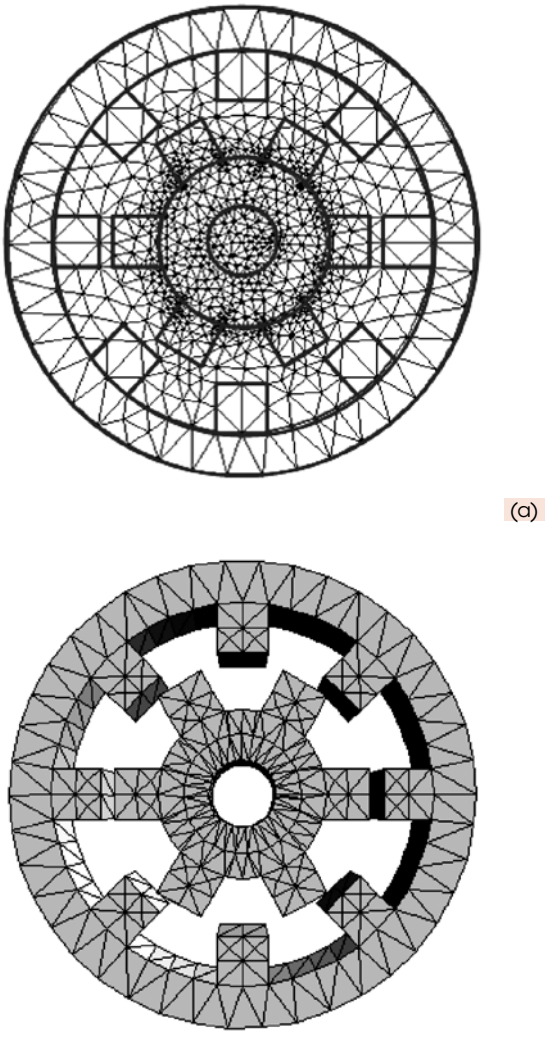
$$\begin{vmatrix} x^2+y^2+z^2 & x & y & z & 1 \\ x_1^2+y_1^2+z_1^2 & x_1 & y_1 & z_1 & 1 \\ x_2^2+y_2^2+z_2^2 & x_3 & y_2 & z_2 & 1 \\ x_3^2+y_3^2+z_3^2 & x_3 & y_3 & z_3 & 1 \\ x_3^2+y_3^2+z_3^2 & x_4 & y_4 & z_4 & 1 \end{vmatrix} = 0 \quad (5)$$

A teoria completa desses métodos pode ser facilmente encontrada em Wolfram (2006a, 2006b).

## 4 Resultados e testes

Os algoritmos apresentados foram implementados com a utilização da biblioteca STL e da linguagem de programação C++. O programa recebe como entrada um arquivo-texto com os pontos da geometria em estudo. A partir da leitura do arquivo de entrada, o programa determina a dimensão da geometria e define a dos atributos das classes. A malha é, então, calculada e os resultados obtidos são visualizados em uma interface desenvolvida com OpenGL (WOO et al., 1999), ou exportados para o Ambiente Mathematica 5.0 (WOLFRAM, 1991).

Os testes foram executados em geometrias bi e tridimensionais, com diferentes complexidades. Uma comparação entre as malhas triangulares e tetraédricas geradas com o algoritmo aqui apresentado pode ser visualizada nas Figuras 6, 7 e 8.



(a)

(b)

**Figura 6: Malha triangular (a) e tetraédrica (b) de um motor de relutância chaveado**
Fonte: Os autores.

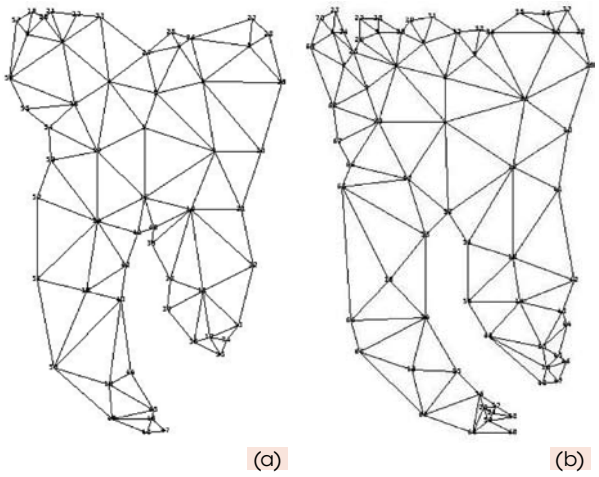



**Figura 7: Cortes consecutivos (a) e (b) de um dente molar inferior**
Fonte: Os autores.

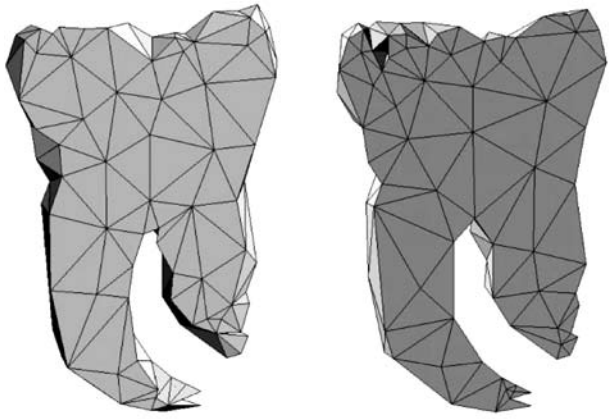


**Figura 8: Seções tridimensionais de um dente molar inferior**
Fonte: Os autores.

**Tabela 3: Avaliação do tempo de processamento**

| Geometria | N° de pontos | N° de elementos | Tempo(s) |
|---|---|---|---|
| Motor de relutância 2D | 809 | 1566 | <3 |
| Motor de relutância 3D | 614 | 3016 | <6 |
| Corte (a) do dente | 57 | 73 | <1 |
| Corte (b) do dente | 70 | 90 | <1 |
| Seção 3D do dente | 135 | 675 | <1 |

Fonte: Os autores.

No campo da engenharia, a Figura 6 representa as malhas de ordem 2 e ordem 3 para um motor de relutância chaveado. Já no ramo da biomedicina, a Figura 7 representa a malha triangular de dois cortes consecutivos de um dente molar inferior, e a Figura 8, suas respectivas representações com malha de tetraedros.

O algoritmo apresentou bom desempenho em malhas tanto bidimensionais quanto em tridimensionais. A avaliação do tempo de processamento dos exemplos apresentados está ilustrada na Tabela 3.

Essas avaliações foram realizadas em um PC, com CPU AMD Athlon XP 2.6+ e 2Gb de memória RAM.

## 5 Conclusões

Neste trabalho, é apresentada apenas uma síntese da implementação completa do gerador de malhas triangular e tetraédrica. Métodos e classes adicionais, além dos mostrados, são necessários para construção completa do algoritmo proposto.

O desempenho computacional foi similar aos métodos tradicionais, tanto na velocidade de geração da malha quanto na qualidade dos elementos gerados. A vantagem dessa implementação é o menor volume de código produzido e, em decorrência disso, maior facilidade e simplicidade na manutenção.

Contudo, em malhas com milhões de elementos, pode haver aumento no tempo de processamento, em razão dos testes adicionais e da não-especificidade dos métodos apresentados.

Os resultados obtidos nas Figuras 6, 7 e 8 comprovam a eficiência da estrutura genérica de dados em combinação com os contêineres e funções otimizadas da biblioteca STL.

## A dynamic data structure for bidimensional and tridimensional Delaunay mesh generation

In this work it is presented the implementation of an algorithm for triangular and tetrahedral Delaunay mesh generation, by the implementation of a set data structure and generic methods that allow selecting dynamically the size of each of these structures, producing classes and methods of order 2, as well as of order 3, according to the dimension of the studied problem. The key idea of this algorithm is to provide 2-dimensional and 3-dimensional mesh generation, without needing two algorithms, or still, of complex data structures, and thus, eliminating code redundancy. The presented methodology was developed with the aid of the containers of Standard Template Library (STL), which supports the necessary tools for the algorithm development.



**Key words:** Bidimensional and tridimensional mesh generation. Data structure. Delaunay algorithm. Finite element methods. STL.


## Referências


JOSUTTIS, N. M. *The C++ standard library*: a tutorial and reference. 11. ed. United States: Addison-Wesley, 2003.

MARRETTO, C. A. R.; MACHADO, J. M. A compact library for automatic generation of tetrahedral meshes using the mathematic system. In: INTERNATIONAL IGTE SYMPOSIUM ON NUMERICAL FIELD CALCULATION IN ELECTRICAL ENGINEERING & EUROPEAN TEAM WORKSHOP, 8., 1998, Áustria. *Proceedings*... Áustria, 1998. p. 116-120.

OWEN, S. J. A Survey of unstructured mesh generation technology. In: INTERNATIONAL MESHING ROUNDTABLE, 7., 1998, Dearborn, Michigan, 1998. p. 239-267.

RUPPERT, J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, Orlando, Florida, v. 18, n. 3 p. 548-585, May 1995.

SAKAMOTO, M. M. Algoritmo de refinamento de Delaunay a malhas seqüenciais, adaptativas e com processamento paralelo. 2007. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2007.

SCHROEDER, W. J.; SHEPHARD, M. S. Shephard, Geometry-based fully automatic mesh generation and the Delaunay triangulation, *International Journal for Numerical Methods in Engineering*, New York, v. 26, p. 2503-2515, 1998.

SHEWCHUK, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: WORKSHOP IN APPLIED COMPUTATIONAL GEOMETRY, 1., 1996, Berlin. *Proceedings*... Berlin, 1996. p. 123-133.

WATSON, D. F. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, *Comput. J.*, London, England, v. 24, p. 167-172, 1981.

WOLFRAM, S. *Mathematica*: a system for doing mathematics by computer. 2. ed. Redwood City, Canada: Addison-Wesley, 1991.

_____. Circumcircle – from Wolfram MathWord. *Site Wolfram MathWord*, 2006 Disponível em: <http://mathworld.wolfram.com/Circumcircle.html> Acesso em: 13 de jun. 2006a.

_____. Circumsphere – from Wolfram MathWord. *Site Wolfram MathWord*, 2006. Disponível em: <http://mathworld.wolfram.com/Circumsphere.html> Acesso em: 13 de jun. 2006b.

WOO, M. et al., D. *OpenGL programming guide,* 3. ed. United States Addison-Wesley, 1999.

ZHU, J; LUBKEMAN, D. L. Object-oriented development of software systems for power system simulations. *IEEE Transactions on Power Systems,* v. 12, n.2, p. 1002-1007, May 1997.



Recebido em 24 out. 2007 / aprovado em 19 nov. 2007

**Para referenciar este texto**

SAKAMOTO, M. M.; CARDOSO, J. R.; MACHADO, J. M. Uma estrutura de dados dinâmica para geração de malhas de Delaunay bidimensionais e tridimensionais. *Exacta*, São Paulo, v. 5, n. 2, p. 391-300, jul./dez. 2007.