*Ministério da Ciência e Tecnologia*

*INPE – Instituto Nacional de Pesquisas Espaciais*

# TRANSPORTE TRANSIENTE DE CALOR EM GEOMETRIAS TRIDIMENSIONAIS COMPLEXAS

Relatório Final de Projeto de Iniciação Científica
(PIBIC/INPE/CNPq)

**Viviane Ribeiro de Siqueira** (UNIP, Bolsista PIBIC/CNPq)
E-mail: viviane@lac.inpe.br

Dr. *Jerônimo dos Santos Travelho* (LAC/CTE/INPE, Orientador)
E-mail: jeff@lac.inpe.br

INPE

São José dos Campos

Maio de 2002

# Agradecimentos

Ao INPE pelas facilidades concedidas para a realização deste projeto de Iniciação Científica; ao PIBIC/CNPq pelo apoio financeiro; ao Doutor Jerônimo pelo incentivo, apoio e valiosa orientação para o desenvolvimento deste projeto; aos amigos do LAC, principalmente Eduardo, Lombardi e Baldan pelo apoio, incentivo, cooperação e amizade dedicados durante todo o período de desenvolvimento do projeto.

Em especial agradeço à minha família e à Deus pelo apoio nos momentos de dificuldade, pelo grande incentivo para o desenvolvimento deste projeto. E agradeço também à todos que colaboraram de maneira direta ou indireta para o desenvolvimento e realização deste projeto.

# Resumo

Este trabalho apresenta a aplicação do método numérico desenvolvido por Frink com a utilização de malhas não-estruturadas bidimensionais e tridimensionais para o cálculo da transferência de calor em regime transiente.

# Lista de figuras

# Lista de Símbolos

| | |
|---|---|
| T | Temperatura |
| k | Condutividade térmica do material |
| l | Tamanho do lado do triângulo |
| A | Área do triângulo |
| $\rho$ | Massa específica do material |
| $c$ | Calor específico do material |
| $\hat{n}$ | Vetor normal aos lados do triângulo |
| $\nabla \cdot$ | Divergente |
| $\nabla t$ | Gradiente de temperatura |
| $\Delta t$ | Intervalo de tempo |

# Índice

# *1. Introdução*

## *1.1 Objetivo do trabalho*

O objetivo desse trabalho é a aplicação do método desenvolvido por Frink [1] para modelagem numérica de problemas tridimensionais transientes de transferência de calor. A transferência de calor é um dos grandes problemas nos projetos de vários ramos da engenharia. As necessidades das diversas áreas da engenharia, na busca de competitividade, tem levado à aplicação de modelos numéricos a geometrias de complexidade crescente. Por essa razão a utilização de malhas não estruturadas tem recebido cada vez mais atenção.

No primeiro projeto de Iniciação Científica [5] foi desenvolvido o estudo com a utilização de métodos centrados na célula baseados no circuncentro. Para geometrias bidimensionais esse método é muito eficaz pois conseguimos gerar malhas de boa qualidade. Porém para geometrias tridimensionais o uso desse método não é muito eficaz pois os mesmos dependem da qualidade da malha que está sendo utilizada e, os geradores tridimensionais atualmente disponíveis no mercado não geram malhas tridimensionais de boa qualidade, com tetraedros regulares. Assim houve a necessidade de utilizar outro método que não seja tão dependente da malha que está sendo utilizada. Assim, neste ano de Iniciação Científica, o projeto tem intuito em estudar a utilização do método desenvolvido por Frink, pois este método tem-se mostrado robusto para esses casos. O método desenvolvido por Frink é destinado à solucionar problemas de escoamento compressíveis em alta velocidade, podendo, no entanto, ser aplicado a outras situações físicas. Nós utilizaremos o método para a solução de problemas de transferência de calor em regime transiente. Neste método as variáveis são calculadas no baricentro das células e o cálculo dos fluxos difusivos são obtidos utilizando os valores das variáveis calculadas na iteração anterior, através do uso do teorema do gradiente.

Esse projeto utilizando o baricentro para o cálculo das variáveis é o início para o desenvolvimento de outros projetos no grupo. No capítulo 2 são apresentadas a fundamentação teórica e aspectos da modelagem numérica, método dos volumes finitos e equação de balanço. No capítulo 3 são apresentadas a metodologia que será utilizada no desenvolvimento desse projeto e o método de Frink. No capítulo 4 são

apresentados os resultados obtidos pela implementação do método. No capítulo 5 são apresentadas as referências bibliográficas.

# 2. *Fundamentação Teórica*

## *2.1 – Uso da Modelagem Numérica*

A modelagem numérica de fenômenos físicos é um meio termo entre a teoria e a experimentação. Ela possui características de ambas. Como na teoria pode-se estudar situações impossíveis na prática, E na experimentação, seus resultados são particulares e são obtidos, muitas vezes, após várias tentativas.

Qualquer modelagem, numérica ou analítica, é feita admitindo-se uma série de hipóteses. Na modelagem numérica, além das hipóteses de validade dos modelos físicos, existem hipóteses referentes às aproximações numéricas. Essas hipóteses devem ser verificadas através da comparação com soluções analíticas. Finalmente, as hipóteses do modelo físico devem ser verificadas através da comparação com resultados experimentais. A modelagem numérica é muito útil, pois permite a análise de várias situações de maneira rápida e econômica, antes de construir protótipos.

A primeira etapa da modelagem numérica de nosso projeto é a geração da malha. As malhas podem ser estruturadas ou não-estruturadas. Uma malha é estruturada se os vértices das células, que não estão no contorno, pertencerem ao mesmo número de células. As células são facilmente acessadas numa malha estruturada, aumentando a velocidade do código. No entanto, as malhas não-estruturadas são mais adequadas para contornos complexos devido à facilidade em acompanhar o contorno da geometria em estudo. Essas malhas geralmente são formadas por triângulos, no caso bidimensional, e tetraedros, no caso tridimensional. O método numérico para malhas não-estruturadas utilizado neste projeto é o método desenvolvido por Frink, centrado na célula, onde as variáveis são calculadas no baricentro das mesmas.

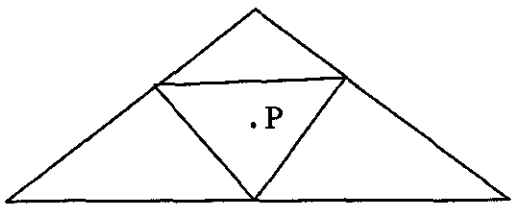


**Figura 1: Exemplo de esquema centrado na célula**

## *2.2 Método dos Volumes Finitos*

No método dos Volumes Finitos divide-se o domínio, a região de estudo, em um conjunto de células para a discretização das equações de balanço e se aplica as equações de balanço a cada célula da malha. Se os fluxos forem calculados corretamente nas faces das células pode-se garantir que as equações de balanço serão válidas para qualquer grupo de células vizinhas. Esse é o método numérico mais próximo das leis físicas e o método utilizado para a discretização da equações no decorrer desse projeto.

## *2.3 Equação de Balanço*

A equação de balanço é utilizada para verificar, em sistemas de modelagem numérica, se a quantidade de calor que entra em uma extremidade da superfície é a mesma quantidade de calor que sai na outra superfície. A equação de balanço para o transporte tridimensional estacionário, ou seja, o campo de temperaturas não depende do tempo:

$$\rho c \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left( k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y}\left( k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z}\left( k \frac{\partial T}{\partial z} \right) \qquad (1)$$

onde $\rho$ é a massa específica do material, c o calor específico e k a condutividade térmica do material.

## *3. Metodologia*

Os programas serão implementados usando ambiente Unix e a linguagem C com o compilador gcc [10] da GNU, disponibilizado gratuitamente pela internet. Para a geração das malhas bidimensionais será utilizado o software EasyMesh [8] e para a geração das malhas tridimensionais será utilizado o software Tetgen [9]. Ambos os softwares são disponibilizados gratuitamente via internet. As figuras 1 e 2 são exemplos de malhas geradas por esses softwares.

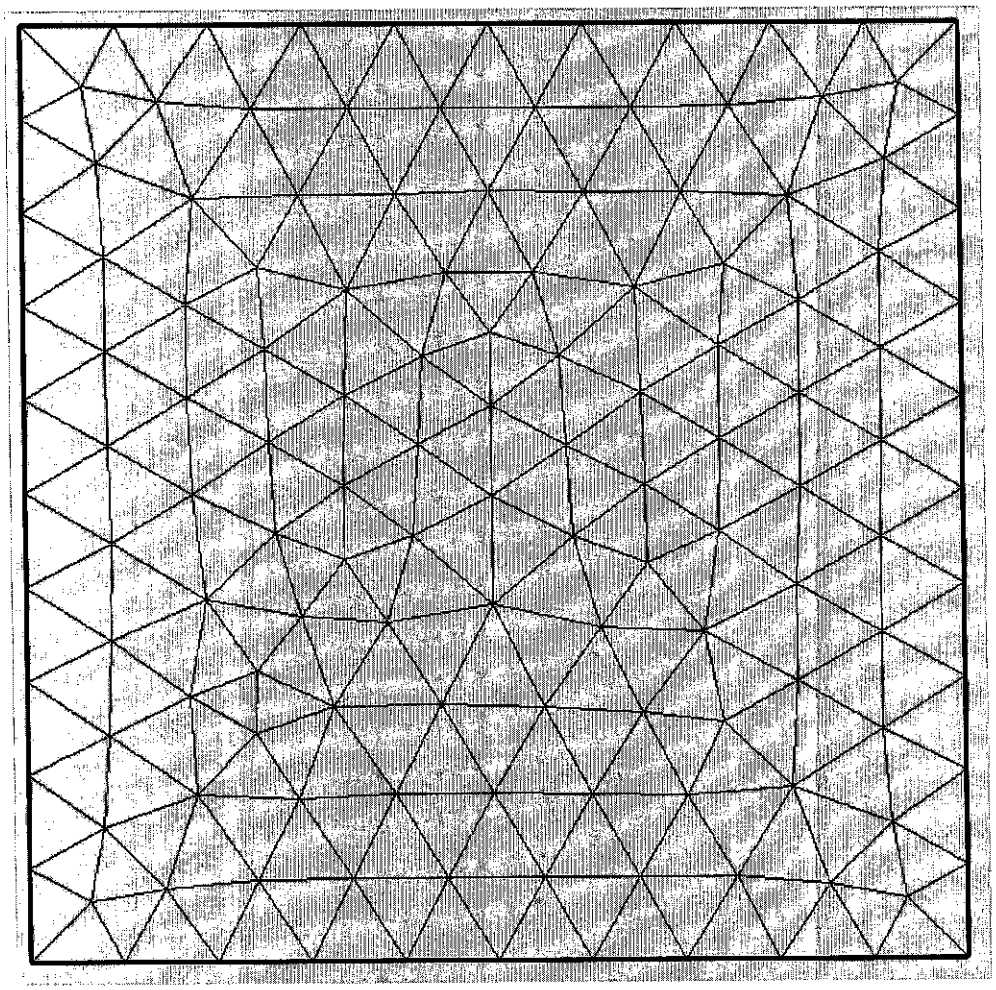

**Figura 2: Malha gerada pelo EasyMesh**

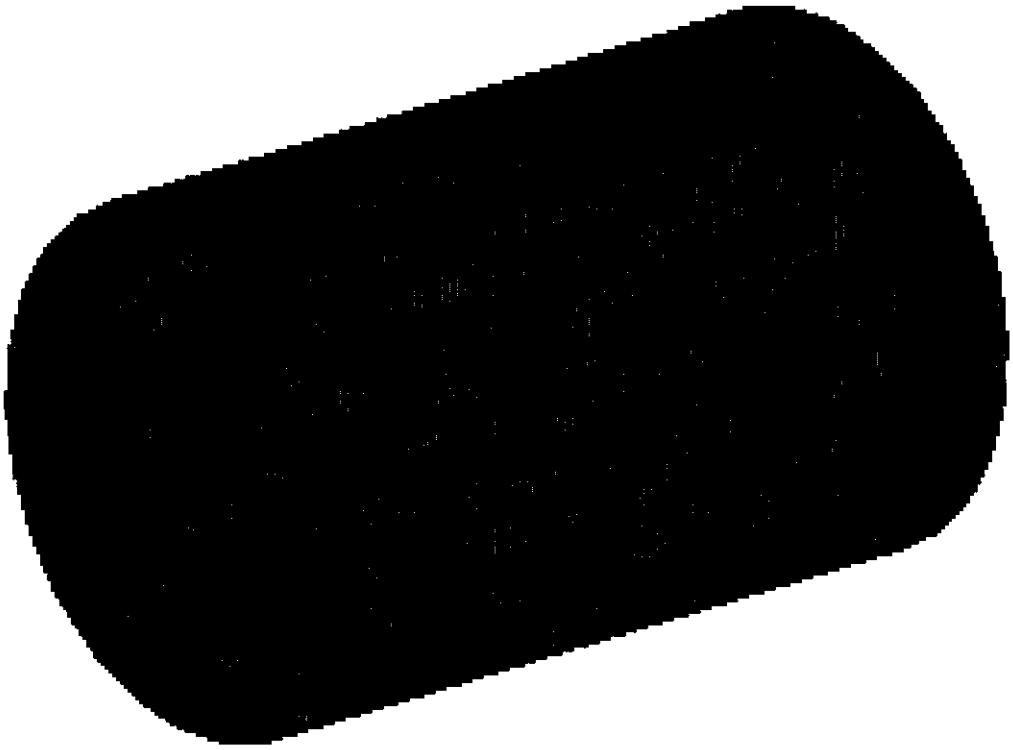

**Figura 3: Malha gerada pelo Tetgen**

Também foi utilizado o gerador de malhas desenvolvido por um dos integrantes do grupo. Este gerador gera, inicialmente, um cubo que é dividido em pequenos cubos que, estes por sua vez, são divididos interiormente em tetraedros.

Foi desenvolvido no grupo um pré-processamento onde serão calculadas as grandezas geométricas como áreas, distâncias, volumes, normais às faces e também ordenação dos vértices e células da malha. Este tornou-se muito viável à todo grupo pois reduziu o tempo de processamento e também gerou um arquivo padrão de saída de resultados de variáveis que irá dinamizar o trabalho do grupo e tornar mais eficiente a comparação de resultados entre os integrantes do grupo.

## *3.1 Método de Frink*

É um esquema do método dos volumes finitos centrados na célula onde o baricentro é o utilizado para o cálculo das variáveis.

*Tratamento do termo difusivo*

A equação de balanço é dado por:

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) \tag{2}$$

Exemplo para o caso bidimensional:

Integrando no volume de controle (a área do triângulo):

$$\iint \rho c \frac{\partial T}{\partial t} dA = \iint \nabla \cdot (k \nabla T) dA \tag{3}$$

Pelo teorema do gradiente:

$$\iint \rho c \frac{\partial T}{\partial t} dA = \oint k \nabla T \cdot \hat{n} \, dl \tag{4}$$

Supondo que a primeira integral pode ser aproximada por:

$$\iint \rho c \frac{\partial T}{\partial t} dA \approx \rho c \frac{T^{n+1} - T^n}{\Delta t} A \tag{5}$$

E a segunda integral pode ser aproximada por:

$$\oint k \nabla T \cdot \hat{n}\, dl \approx \sum_{i=1}^{3} k \nabla T \cdot n_i\, l_i \tag{6}$$

Obtemos a seguinte equação:

$$\rho c \frac{T^{n+1} - T^n}{\Delta t} A = \sum_{i=1}^{3} k \nabla T \cdot n_i\, l_i \tag{7}$$

onde A é a área da célula em estudo, Δt é o intervalo de tempo, ∇t o gradiente de temperatura, $n$ a normal aos lados e $l$ o comprimento dos lados.

Para obter o gradiente de temperatura em cada vértice das células da região de estudo utilizamos o teorema do gradiente:

$$\iint \nabla T\, ds = \oint T\, \hat{n}\, dl \tag{8}$$

Para a implementação do método precisamos calcular o valor da temperatura no baricentro da célula, assim o subconjunto de triângulos onde o lado $l'$ é paralelo ao lado l, passa pelo baricentro dos triângulos maiores, conforme mostra a figura 4.

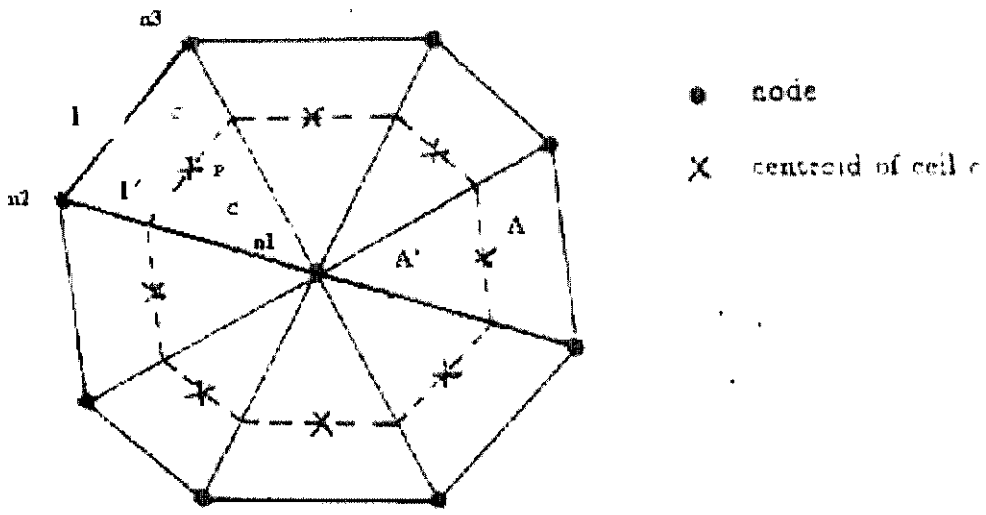


**Figura 4: Estencil para os termos difusivos**

Fonte: [1]

Assim discretizando a equação obtemos:

$$\sum \frac{2}{3} l_{oposto}\, T_{BaricentroVizinho} \cdot \hat{n} = \nabla T_{vertice} \frac{4}{9} \sum A' \tag{9}$$

$$\nabla T_{vertice} = \frac{3}{2} \sum A' \sum T_{BaricentroVizinho} \cdot l_{oposto}\, \hat{n} \tag{10}$$

onde A´ é a soma das áreas dos triângulos que compartilham o mesmo vértice. Como toda linha que sai de um vértice e passa pelo baricentro do triângulo sempre intercepta a metade do lado oposto ao vértice, e como a distância do vértice ao baricentro é 2/3 da distância do vértice ao lado oposto, usando a semelhança de triângulos obtemos:

$$l' = \frac{2}{3} l \tag{11}$$

$$A' = \frac{4}{9} A \tag{12}$$

As equações (7) e (10) são utilizadas na implementação do modelo bidimensional transiente. Para a implementação do modelo tridimensional transiente são usadas as seguintes equações discretizadas, onde a equação de balanço foi integrada no volume de controle (volume do tetraedro).

Para o cálculo da difusão:

$$\rho c \frac{T^{n+1} - T^n}{\Delta t} V = \sum_{i=1}^{4} k \nabla T \cdot n_i \, A_i \tag{13}$$

E para o cálculo dos gradientes nos vértices:

$$\nabla T_{vertice} = \frac{4}{3} \sum V' \sum T_{BaricentroVizinho} \cdot A_{oposta} \, \hat{n} \tag{14}$$

## *4. Resultados*

O trabalho se encontra em andamento e está sendo implementado e verificado, sendo feitos testes para a detecção de bugs. A visualização dos resultados ainda não pode ser mostrada neste relatório devido a implementação estar em desenvolvimento. No apêndice segue a listagem do programa em andamento. Foi utilizado uma malha cujo contorno é um cubo. As condições de contorno foram definidas todas as faces do cubo a 100°C e as demais células à 0°C. Para cálculo das iterações foi utilizado o método de Gauss-Siedel. Também está sendo implementado outros casos, como por exemplo, uma malha cujo contorno é uma esfera. As condições de contorno são definidas da mesma forma que a malha cujo contorno é um cubo, 100°C no contorno e 0°C nas demais células da malha.

## *5. Conclusão*

O método desenvolvido por Frink é tido como robusto para solucionar problemas de transferência de calor. Porém, sua implementação é muito mais complexa do que a implementação usada no projeto de Iniciação Científica anterior, a Abordagem Baseada no Circuncentro. Para a implementação do método desenvolvido por Frink houve a necessidade de ampliar meus conhecimentos na área de Geometria Diferencial e aprender teoremas que não conhecia.

## *6. Proposta de Projetos Futuros*

Para o próximo ano de Iniciação Científica propomos o uso da Abordagem Baseada no Circuncentro para solução de problemas de transferência de calor em geometrias bidimensionais mistas, ou seja, interiormente a malha estruturada e dividida em retângulos e no contorno a malha será não estruturada e dividida em triângulos. Isso facilita e agiliza o processamento pois com a utilização de malhas estruturadas economizamos grande tempo de processamento e refinamos a malha somente em pontos críticos, no nosso caso o contorno da figura.

## *7. Referências Bibliográficas*


1 Frink, N. T.; **"Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver"**, AIAA paper 94-0061, 1994

2 Herbin, R; Labergerie; O. **"Finite volume schemes for elliptic and elliptic-hyperbolic problems on triangular meshes".** Computer Methods in Applied Mechanics and Engineering, Vol:147, Issue: 1-2, pp. 85-103, 1997.

3 Kwak, D. Y.; Kwon, H. J.; Sungyun, L. **"Multigrid algorithm for cell centered finite difference on triangular meshes".** Applied Mathematics and Computation, Vol: 105, pp. 77-85, 1999.

4 Fazenda, A.L.; Enari, E.H.; Travelho, J.S. **"Abordagem pelo Circuncentro para Malhas Não Estruturadas em Tansferência de Calor".** XV Congresso Brasileiro de Engenharia Mecânica - COBEM 99 - Águas de Lindóia – SP, 1999.

5 Siqueira, V. R.; Travelho, J. S. **"Transporte Transiente de Calor em Geometrias Tridimensionais Complexas",** XXIV CNMAC - Setembro de 2001 - Belo Horizonte, MG.

6 Bejan, Adrian **"Transferência de Calor".** Editora Edgard Blücher Ltda, São Paulo, 1996.

7 Fortuna, A. O. **"Técnicas Computacionais para Dinâmica dos Fluidos – Conceitos Básicos e Aplicações".** Editora da Universidade de São Paulo, 2000.

8 Bajpai, A. C.; Mustoe, L. R.; Wlaker, D.; **"Matemática para Engenharia".** Hemus, São Paulo, 1980.

9 Fazenda, A.L.; Travelho, J.S. **"Modelagem Bidimensional de Escoamentos Viscosos Incompressíveis com Malhas Não Estruturadas utilizando a Abordagem Baseada no Circuncentro ".** Tese de Doutorado apresentada no INPE/CAP, São José dos Campos – SP, 2002.

10 Filho, N. P.; Travelho, J.S. **"Simulação de Escoamentos Incompressíveis com o uso da Abordagem Baseada no Circuncentro ".** Tese de Doutorado apresentada no INPE/CAP, São José dos Campos – SP, 2000.

11 **EasyMesh**; http://www-dinma.univ.trieste.it/~nirftc/research/easymesh/
Último acesso: Fevereiro de 2002

12 **Tetgen**; http://www.weboo.com/sh/tetgen.htm
Último acesso: Fevereiro de 2002

13 **GNU**; http://www.gnu.org
Último acesso: Fevereiro de 2002

## 8. *Apêndice*

```
/*************************************************************

TRANSFERENCIA DE CALOR TRIDIMENSIONAL TRANSIENTE E

CONDUTIVIDADE TERMICA CONSTANTE

METODO FRINK

Dominio cujo contorno e um CUBO

Paredes do contorno adiabaticas (Gradiente nos vertices do
triangulo igual a zero)

Condicao inicial de temperatura:
Dois triangulos possuem temperaturas conhecidas: 100C e 0C


Viviane Ribeiro de Siqueira
viviane@lac.inpe.br

Data de inicio: 26 de dezembro de 2001

Ultima atualizacao: 15 de maio de 2002

*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>

#define Max 2000              // Limite maximo para elementos do triangulo
#define Min 50

#define k 150.0
#define deltat 0.01
#define roc 2747.25
```

```
#define prec 0.001

/*************************************************************
   Definicao das Estruturas
*************************************************************/
typedef struct _Vertice
     {
     float x, y, z;        // Coordenadas x, y e z de cada vertice
     float Volredor; // Volume formado pelos tetraedros que compartilham o
vertice
     int NTcV;             // Numero de tetraedros que compartilham o vertice
                           // (E igual ao numero de faces opostas ao vertice)
     int iTcV[Min];        // Indice do tetraedro que compartilha o vertice
     int iFopV[Min];       // Indice da face oposta ao vertice (Sao as faces
                           // de cada tetraedro armazenado na linha acima)
     int Fat[Min];         // Fator da face oposta ao vertice (De acordo com as
faces
                           // acima armazenadas)
     int NFCcV;            // Numero de faces no contorno que compartilham o
vertice)
     int iFCcV[Min];       // Indice da face do contorno que compartilha o
vertice
     int FatCont[Min];     // Fator da normal da face no contorno que
compartilha o vertice
     float gradx;          // Gradiente de Temperatura em cada vertice
(componente x)
     float grady;          // Gradiente de Temperatura em cada vertice
(componente y)
     float gradz;          // Gradiente de Temperatura em cada vertice
(componente z)
     } Vertice;

typedef struct _Face
     {
     int v1, v2, v3; // Indice dos Vertices que formam a face
     float Af;             // Area da Face
     float bx, by, bz;     // Coordenadas x, y e z do baricentro de cada face
     float Nx, Ny, Nz;     // Valor da normal
     int Ta, Tb;           // Indice dos tetraedros que      compartilham a face
```

```
    } Face;

typedef struct _Componentes
    {
    int v1, v2, v3, v4;    // Indice dos Vertices que formam o Tetraedro
    float V;               // Volume total do Tetraedro
    float bx, by, bz;      // Coordenadas x, y e z de cada tetraedro
    float dbV1, dbV2;      // Distancia do baricentro aos vertices
    float dbV3, dbV4;      // Distancia do baricentro aos vertices
    int FopV1, FopV2;      // Face oposta aos vertices
    int FopV3, FopV4;      // Face oposta aos vertices
    int fatf1, fatf2;   // Fator da face
    int fatf3, fatf4;      // Fator da face
    int Viz1, Viz2;        // Vizinhos as faces
    int Viz3, Viz4; // Vizinhos as faces
    float T;       // Temperatura no baricentro do triangulo (instante
atual)
    float Tf;      // Temperatura no baricentro do triangulo
(instante futuro)
    float Tnx;             // Produto entre a temperatura e normal e Area
face(componente x)
    float Tny;             // Produto entre a temperatura e normal e Area
face(componente y)
    float Tnz;             // Produto entre a temperatura e normal e Area
face(componente z)
    float Som1;            // Somatorio de K*GradT.nl no vertice 1
    float Som2;            // Somatorio de K*GradT.nl no vertice 2
    float Som3;            // Somatorio de K*GradT.nl no vertice 3
    float Som4;            // Somatorio de K*GradT.nl no vertice 4
    float Som;             // Somatorio de K*GradT.nl nso quatro vertices
    } Componentes;


/*************************************************************
   Rotina Principal
*************************************************************/

void main (void)
  {
```

```
/*************************************************************
   Declaracao das variaveis
*************************************************************/

  Vertice V[Max];          // Vertices
  Componentes C[Max];      // Componentes dos tetraedros
  Face F[Max];             // Faces do tetraedro

  int NV,                  // Numero de Vertices
     NVC,                  // Numero de Vertices no Contorno
     QTCV, QTCF,           // Quantidade de Tipos de Contorno
     NVCT[Min],            // Numero de Vertices em cada Contorno (T = Tipo 1,
2, ...)
     NF,                   // Numero de Faces
     NFC,                  // Numero de Faces no Contorno
     NFCont[Min],          // Numero de Faces em cada Contorno
     NT,                   // Numero de Tetraedros
     NT3FC, NT2FC,         // Numero de Tetraedros com 3 e 2 faces no
contorno
     NT1FC, NT0FC;         // Numero de Tetraedros com 1 e 0 faces no
contorno

  int i, j, aux1, aux2, aux3, aux4, aux11, aux22, aux33, aux44, auxt,
auxfn, aux, cont;

  float soma;              // Variavel usada como parada do programa

  FILE *fe, *fs;           // Ponteiros para arquivos (fe = entrada, fs =
saida)

  puts ("Iniciando o programa...");

/*************************************************************
   Abrindo arquivos
*************************************************************/
  if ((fe = fopen ("cubo.grd", "r")) != NULL) printf ("Abriu o arquivo
de entrada \n");
  else printf ("Nao conseguiu abrir o arquivo de          entrada \n");
```

```
  if((fs = fopen ("cubo.saida","w")) != NULL) printf("Abriu o arquivo
de saida \n");
  else printf("Nao conseguiu abrir o arquivo de saida \n");

/*************************************************************
   Leitura dos dados
*************************************************************/
  // Le numero de vertices
  fscanf (fe, "%d", &NV);

  // Le numero de vertices no contorno
  fscanf (fe, "%d", &NVC);

  // Le quantidade de tipos de contorno
  fscanf (fe, "%d", &QTCV);

  // Le quantidade de vertices em cada contorno
  for (i = 0; i < QTCV; i++) fscanf (fe, "%d", &NVCT[i]);

  for (i = 0; i < NV; i++)
    {
    // Le coordenadas x, y, z de cada vertice
    fscanf (fe, "%f %f %f", &V[i].x, &V[i].y, &V[i].z);

    // Le volume de contorno ao redor de cada vertice
    fscanf (fe, "%f", &V[i].Volredor);

    // Le numero de tetraedros que compartilham cada vertice
    fscanf (fe, "%d", &V[i].NTcV);

    // Le indice dos tetraedros que compartilham cada vertice
    for (j = 0; j < V[i].NTcV; j++) fscanf (fe, "%d", &V[i].iTcV[j]);

    // Le indice face oposta ao vertice de cada tetraedro lido
anteriormente
    for (j = 0; j < V[i].NTcV; j++) fscanf (fe, "%d", &V[i].iFopV[j]);

    // Le o valor do fator da normal a face
```

```
   for (j = 0; j < V[i].NTcV; j++) fscanf (fe, "%d  ", &V[i].Fat[j]);

   // Le numero de faces no contorno que compartilham cada vertice
   fscanf (fe, "%d", &V[i].NFCcV);

   if (V[i].NFCcV != 0)
   {
   // Le o indice da face do contorno que compartilha o vertice
   for (j = 0; j < V[i].NFCcV; j++) fscanf (fe, "%d",
&V[i].iFCcV[j]);

   // Le o valor do fator da normal a face
   for (j = 0; j < V[i].NFCcV; j++) fscanf (fe, "%d",
&V[i].FatCont[j]);
   }
   }

//  for (i =0; i<NV; i++ ) fprintf (fs, " %d %f %f %f\n", i, V[i].x,
V[i].y, V[i].z);

  // Le numero de faces
  fscanf (fe, "%d", &NF);

  // Le numero de faces no contorno
  fscanf (fe, "%d", &NFC);

  // Le quantidade de tipos de contorno
  fscanf (fe, "%d", &QTCF);

 // fprintf (fs, "%d %d %d\n", NF, NFC, QTCF);

  // Le quantidade de faces em cada contorno
  for (i = 0; i < QTCF; i++) fscanf (fe, "%d", &NFCont[i]);

  for (i = 0; i < NF; i++)
   {
   // Le o indice dos vertices de cada face
   fscanf (fe, "%d %d %d", &F[i].v1, &F[i].v2, &F[i].v3);
```

```
  // Le a area da face
  fscanf (fe, "%f", &F[i].Af);

  // Le o baricentro de cada face
  fscanf (fe, "%f %f %f", &F[i].bx, &F[i].by, &F[i].bz);

  // Le o valor da normal a cada face
  fscanf (fe, "%f %f %f", &F[i].Nx, &F[i].Ny, &F[i].Nz);

  // Le o indice dos tetraedros que compartilham a face
  fscanf (fe, "%d %d", &F[i].Ta, &F[i].Tb);
  }

// Le numero de tetraedros
fscanf (fe, "%d", &NT);

// Le numero de tetraedros com 3 faces no contorno
fscanf (fe, "%d", &NT3FC);

// Le numero de tetraedros com 2 faces no contorno
fscanf (fe, "%d", &NT2FC);

// Le numero de tetraedros com 1 faces no contorno
fscanf (fe, "%d", &NT1FC);

// Le numero de tetraedros com 0 faces no contorno
fscanf (fe, "%d", &NT0FC);

// fprintf (fs, "%d %d %d %d %d\n", NT, NT3FC, NT2FC, NT1FC, NT0FC);

for (i = 0; i < NT; i++)
  {
  // Le o indice dos vertices de cada tetraedro
  fscanf (fe, "%d %d %d %d", &C[i].v1, &C[i].v2, &C[i].v3,
&C[i].v4);

  // Le o volume do tetraedro
  fscanf (fe, "%f", &C[i].V);
```

```
 //  fprintf (fs, "%d %d %d %d %d %f\n", i, C[i].v1, C[i].v2,
C[i].v3, C[i].v4, C[i].V);

   // Le o baricentro de cada tetraedro
   fscanf (fe, "%f %f %f", &C[i].bx, &C[i].by, &C[i].bz);

   // Le a distancia do baricentro aos vertices
   fscanf (fe, "%f %f %f %f", &C[i].dbV1, &C[i].dbV2, &C[i].dbV3,
&C[i].dbV4);

   // Le o indice das faces opostas aos vertices
   fscanf (fe, "%d %d %d %d", &C[i].FopV1, &C[i].FopV2, &C[i].FopV3,
&C[i].FopV4);

   // Le o fator das faces opostas aos vertices
   fscanf (fe, "%d %d %d %d", &C[i].fatf1, &C[i].fatf2, &C[i].fatf3,
&C[i].fatf4);

   // Le o indice dos tetraedros vizinhos
   fscanf (fe, "%d %d %d %d", &C[i].Viz1, &C[i].Viz2, &C[i].Viz3,
&C[i].Viz4);

 //  fprintf (fs, "%d %d %d %d %d\n", i, C[i].Viz1, C[i].Viz2,
C[i].Viz3, C[i].Viz4);
   }


   puts ("Iniciando o processamento...");

   cont = 0;

/*************************************************************
   Condicoes iniciais de temperatura
*************************************************************/
  for (j = 0; j < NT; j++)
   {
   C[j].T = 0.0;
   C[j].Tf = 0.0;
```

```
 // if ((C[j].Viz1 == -1) || (C[j].Viz2 == -1) || (C[j].Viz3 == -1)
|| (C[j].Viz4 == -1)) C[j].T = 100.0;
    }

   for (j = 0; j < NT/2; j++) C[j].T = 100.0;

 // for (j = 0; j < NT; j++) fprintf (fs, "%d %f\n", j, C[j].T);

/*****************************************************************
   Calculo do Fluxo difusivo
*****************************************************************/

  while ((soma > prec) || (cont < 1000))
     {

/*****************************************************************
   Calculo do Produto entre a temperatura e normal*Area
*****************************************************************/
     for (j = 0; j < NV; j++)
      {
      C[j].Tnx = 0.0;
      C[j].Tny = 0.0;
      C[j].Tnz = 0.0;
      }

     for (j = 0; j < NV; j++)
      {
      for (i = 0; i < V[j].NTcV; i++)
       {
       auxt = V[j].iTcV[i];
       auxfn = V[j].iFopV[i];

       C[j].Tnx += C[auxt].T * F[auxfn].Af * F[auxfn].Nx *
V[j].Fat[i];
       C[j].Tny += C[auxt].T * F[auxfn].Af * F[auxfn].Ny *
V[j].Fat[i];
       C[j].Tnz += C[auxt].T * F[auxfn].Af * F[auxfn].Nz *
V[j].Fat[i];
       }
```

```
    // fprintf (fs, "%d %f %f %f\n", j, C[j].Tnx, C[j].Tny,
C[j].Tnz);
        }

/*************************************************************
   Calculo do Gradiente no vertices
*************************************************************/
     for (i = 0; i < NV; i++)
      {
      V[i].gradx = 1.3333 * 1/V[i].Volredor * C[i].Tnx;
      V[i].grady = 1.3333 * 1/V[i].Volredor * C[i].Tny;
      V[i].gradz = 1.3333 * 1/V[i].Volredor * C[i].Tnz;

   // fprintf (fs, "%d %f %f %f\n", j, V[j].gradx, V[j].grady,
V[j].gradz);
        }

/*************************************************************
   Calculo do somatorio dos quatro vertices de cada triangulo:
   Condutividade Termica * (Produto Escalar do Gradiente
   nos vertices entre o produto da normal e o lado)
*************************************************************/
     for (j = 0; j < NT; j++)
      {
   /*  if ((C[j].Viz1 == -1) || (C[j].Viz2 == -1) || (C[j].Viz3 ==
-1) || (C[j].Viz4 == -1))
        {
        C[j].T = 100.0;
        C[j].Tf = 100.0;
        }

       else
        {*/
        aux1 = C[j].v1;
        aux2 = C[j].v2;
        aux3 = C[j].v3;
        aux4 = C[j].v4;

        aux11 = C[aux1].FopV1;
```

```
        aux22 = C[aux2].FopV2;
        aux33 = C[aux3].FopV3;
        aux44 = C[aux4].FopV4;

      C[j].Som1 = (V[aux1].gradx * F[aux11].Af * F[aux11].Nx *
C[aux1].fatf1) +
              (V[aux1].grady * F[aux11].Af * F[aux11].Ny *
C[aux1].fatf1) +
              (V[aux1].gradz * F[aux11].Af * F[aux11].Nz *
C[aux1].fatf1);

      C[j].Som2 = (V[aux2].gradx * F[aux22].Af * F[aux22].Nx *
C[aux2].fatf2) +
              (V[aux2].grady * F[aux22].Af * F[aux22].Ny *
C[aux2].fatf2) +
              (V[aux2].gradz * F[aux22].Af * F[aux22].Nz *
C[aux2].fatf2);

      C[j].Som3 = (V[aux3].gradx * F[aux33].Af * F[aux33].Nx *
C[aux3].fatf3) +
              (V[aux3].grady * F[aux33].Af * F[aux33].Ny *
C[aux3].fatf3) +
              (V[aux3].gradz * F[aux33].Af * F[aux33].Nz *
C[aux3].fatf3);

      C[j].Som4 = (V[aux4].gradx * F[aux44].Af * F[aux44].Nx *
C[aux4].fatf4) +
              (V[aux4].grady * F[aux44].Af * F[aux44].Ny *
C[aux4].fatf4) +
              (V[aux4].gradz * F[aux44].Af * F[aux44].Nz *
C[aux4].fatf4);


      C[j].Som = (C[j].Som1 + C[j].Som2 + C[j].Som3 + C[j].Som4) *
k * deltat;
      C[j].Tf = C[j].Som / (roc * C[j].V);
      C[j].Tf += C[j].T;

      C[j].T = C[j].Tf;
```

```
    }

/*************************************************************
   Calculo do Erro
*************************************************************/
    soma = 0.0;

    for (j = 0; j < NT; j++) soma += pow((C[j].Tf - C[j].T), 2.0);
    soma = pow(soma, 0.5);

    cont++;
    }

/*************************************************************
   Impressao dos dados
*************************************************************/
  // Imprime o valor da temperatura em cada tetraedro

   fprintf (fs, "%d\n", cont);

   for (j = 0; j < NT; j++) fprintf (fs, "%d %f %f\n", j, C[j].T,
C[j].Tf);


  fclose (fe);
  fclose (fs);
  puts ("Terminou o processamento.");
  }
```