4
IMPLEMENTAÇÕES DO PROCESSO ADAPTATIVO

Este capítulo apresenta as implementações do novo processo adaptativo tanto para os problemas 2D quanto para os 3D. Entende-se por processo adaptativo todo o procedimento necessário para a geração de malhas adaptativas. Portanto estão incluídos neste contexto a estratégia adaptativa (*h*), os métodos de suavização (*Z*2*-HC* [62] e *SPR* [63-66]), os estimadores de erro e as técnicas de geração de malhas.

Antes é importante esclarecer alguns detalhes. Por exemplo, o processo de simulação de um problema físico é dividido em três partes: representação geométrica e topológica do modelo (módulo de modelagem), solução do problema (módulo de análise) e interpretação dos resultados (módulo de visualização).

Tanto para o problema 2D quanto para o 3D, usou-se somente um programa de análise, denominado *FEMOOP* [43], mas os outros dois módulos são diferentes, dependendo da dimensão do problema. Para o caso 2D, o programa responsável pelo módulo de modelagem e pelo módulo de visualização é o *MTOOL* [47]. Para o caso 3D, o *MG* [48] é o programa responsável pelo módulo de modelagem e o *POS*-3*D* [16] é responsável pelo módulo de visualização. Em 2D, os módulos de modelagem e visualização integrados em um único programa (*MTOOL* [47]), que se comunica automaticamente com o módulo de análise (*FEMOOP* [43]). Esta integração ainda não foi feita para o caso 3D. Entretanto, para se fazer o processo adaptativo não é necessário o módulo de visualização. Portanto, o processo adaptativo foi implementado no *MTOOL* [47] (2D) e no *MG* [48] (3D). Outra diferença entre os dois casos é que o *MTOOL* foi desenvolvido usando-se a linguagem de programação C e o *MG* [48] foi desenvolvido em C++, fundamentada no paradigma de programação orientada a objetos.

Outro aspecto muito importante na simulação de problemas físicos é o gerenciamento dos atributos. Nos dois casos, o módulo responsável por esta parte é o ESAM (*Extensible System Attributes Management*) [*]. O gerenciamento

[*] para maiores detalhes ver [36].

de atributos é importante para que os modeladores sejam capazes de atender aos diversos tipos de análise em engenharia, como, por exemplo, estrutural, naval, termodinâmica, etc. Os atributos para cada tipo de simulação devem poder ser configurados sem a necessidade de se alterar o código-fonte do modelador.

O *ESAM* também possibilita que uma interface seja definida para a captura dos dados referentes aos atributos do *MTOOL* [47] e do *MG* [48]. Existe, ainda, um mecanismo que permite ao usuário do *MTOOL* [47] ou do *MG* [48] implementar a rotina responsável pela geração do arquivo de dados, referente a comunicação entre os pré-processadores e os módulos de análise. Assim, os novos atributos incorporados ao programa de análise podem ser facilmente inseridos aos pré-processadores, a partir da definição de uma interface para capturar os seus dados, bem como a inclusão, pelo próprio usuário do sistema, dos dados referentes a esses atributos na rotina que gera o arquivo de dados lido pelo programa de análise. No *ESAM*, a configuração dos atributos é feita através de arquivos que contém instruções a linguagem de programação extensível LUA [29] (linguagens extensíveis são usadas para estender ou configurar uma aplicação para fins particulares). Esta linguagem é interpretada, permitindo que a configuração dos atributos seja feita sem a necessidade de recompilação do sistema. Este sistema utiliza uma abordagem conhecida como modelagem baseada em geometria. Neste tipo de abordagem, os atributos são associados às entidades geométricas do modelo e a discretização desta geometria (malha de elementos finitos) herda automaticamente estes atributos. Este tipo de modelagem oferece diversas vantagens, como, por exemplo, um melhor suporte para a automação do processo de modelagem, incluindo geração automática de malhas e conseqüentemente facilitando a utilização do processo adaptativo, pois os atributos são associados às entidades geométricas do modelo, não precisando serem redefinidos durante o refinamento da malha.

4.1. *MTOOL* (2D)

Nesta seção, apresenta-se de forma sucinta o programa *MTOOL* [47], bem como todos os outros componentes usados no seu desenvolvimento.

O *MTOOL* [47] é um programa utilizado para simulações numéricas de problemas bidimensionais pelo Método dos Elementos Finitos. Este sistema integra, em um único ambiente, os módulos de pré-processamento (sistema gráfico interativo e configurável para geração, edição e manipulação de malhas de elementos finitos bidimensionais) e pós-processamento, responsáveis pela geração do modelo numérico para análise pelo MEF e visualização de resultados, com o programa de análise numérica *FEMOOP* [43] (análises numéricas quasi-estáticas, com não-linearidade física, de modelos de estado plano de tensão, estado plano de deformação e aximétricos). Este sistema é extensível e configurável, permitindo que novas funcionalidades sejam facilmente implementadas.

Este sistema foi desenvolvido utilizando-se o sistema de interface com o usuário IUP/LUA[*] que permite a obtenção de uma interface atrativa, configurável e eficiente para execução das tarefas. Sob o sistema de interface e o sistema para geração de malhas propriamente dito, encontra-se o sistema gráfico CD [19], que garante a portabilidade multi-plataforma da visualização. Outra ferramenta importante utilizada no desenvolvimento do *MTOOL* [47] é a biblioteca, HED [42], que manipula estruturas de dados topológicas para o gerenciamento de subdivisões planares. Além disso, foi utilizado um sistema configurável para o gerenciamento de atributos, *ESAM*, escrito na linguagem interpretada LUA [29].

O *MTOOL* [47] é composto por cinco módulos responsáveis pelas diversas tarefas envolvidas em um processo de simulação (modelagem geométrica, especificação dos atributos, geração da malha de elementos finitos, análise numérica e visualização dos resultados), e por um módulo principal responsável pelo gerenciamento e integração dos outros módulos (Figura 23).

[*] http://www.tecgraf.puc-rio.br/iup

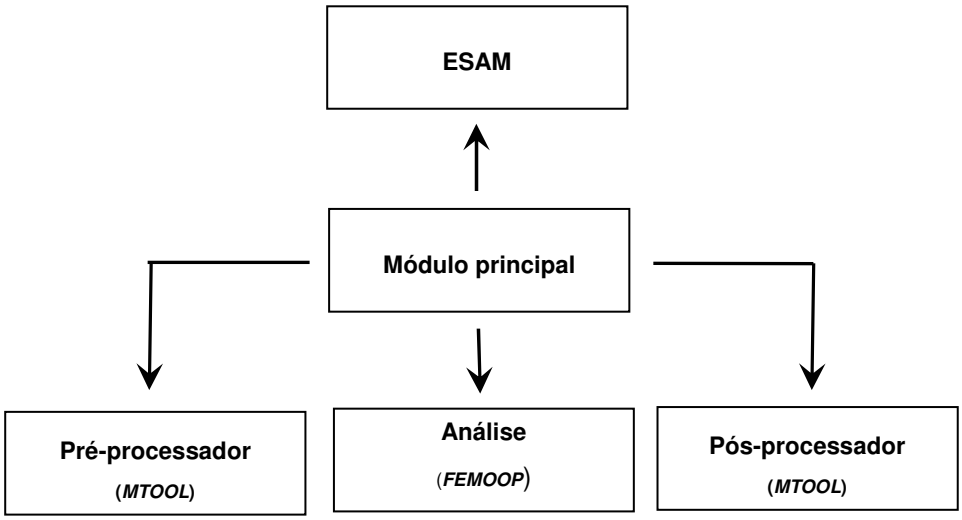

Figura 23 Fluxograma mostrando o funcionamento do *MTOOL*.

O *MTOOL* [47] (*Two-dimensional Mesh Tool*) possui dois ambientes bem distintos. O primeiro é responsável pela definição da geometria e topologia do modelo. Este ambiente é suportado pela estrutura de dados topológica *HED* (*half-edge data structure*) [42], que permite assegurar a consistência do modelo durante a sua criação. É possível, por exemplo, identificar interseções entre as curvas, determinar se uma região (face) foi definida ou dividida, verificar os efeitos indiretos de uma determinada atribuição (por exemplo, a troca do número de subdivisões de uma curva elimina as malhas adjacentes, caso contrário o modelo ficaria inconsistente), etc. A implementação desta estrutura de dados topológica é complexa e ocupa um espaço de memória relativamente grande para a sua realização. Entretanto, a biblioteca de rotinas *HED* [42] oferece uma interface de alto nível (nível abstrato), abstraindo a complexidade desta estrutura de dados. O segundo ambiente do *MTOOL* [47] apresenta uma representação única de malhas de elementos finitos convencional, obtida a partir da conversão das malhas do modelo de regiões, definido no ambiente de edição do modelo. No segundo ambiente, o usuário pode realizar operações típicas de modelos de elementos finitos, como, por exemplo, consultar a incidência nodal de cada elemento finito gerado na conversão. No entanto, não existem funções para edição da malha, pois não se tem o suporte de uma estrutura de dados topológica. Para isso, o usuário pode retornar ao ambiente de edição do modelo, realizar as alterações necessárias e fazer uma nova conversão para o modelo de elementos finitos.

O módulo de visualização fornece informações qualitativas e quantitativas sobre a malha e os resultados referentes à mesma. A análise quantitativa é feita através de gráficos de resposta e funções de consulta, que fornecem informações tais como número de nós ou incidência do elemento. A análise qualitativa consiste basicamente na representação visual de resultados, que podem ser de dois tipos: resultados escalares, como componentes de tensão ou campo de temperatura, e resultados vetoriais, tais como campo de deslocamentos ou campo de velocidades. Os resultados de campos escalares podem ser fornecidos nos pontos nodais, de forma suavizada ou não, ou nos pontos de Gauss. Para auxiliar na visualização do modelo, existem funções para manipular o campo de visão, tais como a possibilidade de mudar os limites da janela de visualização ou fazer um *zoom* em todo o modelo ou em alguma face do mesmo. O programa ainda permite visualizar a animação do modelo com a resposta de um determinado caso e campo ao longo de diversos passos. A entrada de dados referente à malha de elementos finitos e aos resultados obtidos a partir da análise numérica é feita através de arquivos neutros.

O módulo principal é responsável pelo gerenciamento e integração dos demais módulos do *MTOOL* [47] (*MTOOL, ESAM, FEMOOP*).

Neste módulo são chamadas as funções de inicialização referentes aos outros módulos do sistema. Outra tarefa realizada por este módulo consiste na conversão dos arquivos gerados pelo programa *FEMOOP* [43] contendo os resultados da análise, que possuem formatos próprios, para que possam ser interpretados pelo pós-processador *MTOOL* [47], que utiliza o formato do arquivo de dados *NeutralFile*[*].

[*] ver (http://www.tecgraf.puc-rio.br/neutralfile)

4.1.1. Adaptatividade no *MTOOL*

Para que o uso da adaptatividade pudesse ser disponibilizado pelo programa *MTOOL* [47], foram implementados módulos responsáveis pelo refinamento da *binary-tree* e a conseqüente subdivisão das curvas e módulos responsáveis pelo refinamento da *quadtree* e o consecutivo controle do tamanho dos elementos que serão gerados nas malhas de superfície.

Como o *MTOOL* [47] é dividido em dois ambientes (o ambiente de modelagem e o ambiente de malha) foi necessário separar o processo adaptativo nestes dois ambientes. Desta forma, após se selecionar o método de suavização e determinar à razão de erro máxima η^*, passa-se para o ambiente de malha e se faz a análise por elementos finitos. Em seguida executa-se a opção da análise adaptativa com ou sem a adaptatividade geométrica.

O processo adaptativo no *MTOOL* [47] está composto basicamente por duas funções, que são: *_MtoolAdaptiveQt* e *_MtoolAdaptiveMesh*.

A função *_MtoolAdaptiveQt* tem fundamentalmente as seguintes tarefas.

1) Ler as informações do erro de cada elemento e associa o valor de *h* ao centro geométrico de cada elemento (*MtoolGetError*);

2) Calcular em qual das coordenadas Cartesianas está a maior dimensão do sólido, guardando-as (*MtoolGetBoundBox*);

3) Construir a *quadtree* global (*Adapt_CreateQuadtree*).

As tarefas da função *_MtoolAdaptiveQt* podem ser visualizadas na Figura 24.

```
// variáveis da função
  vars = xmax, ymax, xmin, ymin, n_elem, n_nodes, eta, inf_el_eachnode,
          par0, par1, par2, par3

 // captura o parâmetro que verfica se será feita a adaptatividade geométrica
 par0 = 0 ou 1

// testa se é para ser feita a adaptatividade geométrica
  Se par0 = 1 capturam-se os outros parâmetros necessários para
  a adaptatividade geométrica.
   // tolerância mínima da distância entre as subdivisões estipulada pelo usuário
   // do programa
```

```
   par1 = d_MIN;
// tolerância máxima da distância entre as subdivisões estipulada pelo usuário
// do programa
   par2 = d_MAX;
 // tolerância angular estipulada pelo usuário do programa
   par3 = ang_TOL
 fim Se

 Se par0 = 0 não se fará a adaptatividade geométrica e inicializam-se
 os outros parâmetros como zero.
    par1 = 0; // tolerância mínima da distância entre os pontos
    par2 = 0; // tolerância máxima da distância entre os pontos
    par3 = 0; // tolerância angular
 fim Se

 // captura o parâmetro que verfica se será feita a adaptatividade baseada em
 // erro
 par4 = 0 ou 1

 // preenche o vetor inf_el_eachnode e retorna o valor do erro corrente
 eta = MtoolGetError (n_elem, n_nodes, inf_el_eachnode);

 // calcula as coordenadas da Bound Box (coordenadas cartesianas máximas e
mínimas)
 MtoolGetBoundBox (n_elem, xmax, xmin, ymax, ymin);

 // Cria a quadtree no ambiente de malha
 Adapt_CreateQuadtree (xmax, ymax, xmin, ymin, n_elem, inf_el_eachnode,
                       n_nodes, par0, par1, par2, par3, par4);

  // retorna o valor do erro corrente
 (eta)
```

Figura 24 Pseudo código da função *_MtoolAdaptiveQt*.

Na função *MtoolGetError* mostrada da Figura 24, calcula-se o valor de *h* discutido na seção 3.2, que é o valor que controlará o refinamento da *quadtree*, da *binary-tree* e o tamanho dos elementos triangulares que serão gerados no interior da malha.

A Figura 25 mostra o trecho de código com o cálculo de *h* quando se usa elementos lineares T3. No caso quadrático (T6), o trecho de código para o cálculo de *h* é praticamente o mesmo. A única diferença está no instante de calcular a área existente e no valor de p usado para obter o valor de *h*, pois conforme a equação (23), o valor de *p* varia de acordo com o grau do polinômio de aproximação, ou seja $p = 1$ para o T3 e $p = 2$ para o T6.

```
// variáveis da função
  vars = norm_beb, norm_bub, eta_barra, beb, bub, i, k, a, hold, hnew, fact, erro
  // calcula a área do elemento triangular T3 (a) através dos valores
  // nodais dos vértices do elemento P1(v1x,v1y), P2(v2x,v2y)  e P3(v3x,v3y)
  a = 0.5*det[(v1x*v2y)+(v2x*v3y)+(v3x*v1y)-(v2y*v3x)-(v1y*v2x)-(v1x*v3y)]

  // valor de L após se igualar às áreas existente e ideal
  hold = sqrt( 4 * a / sqrt(3) );

 // lê do arq '.pos' o valor de RATIO_ERRO para cada elemento
  De i = 0 até i < nelem faz-se i = i + 1
       // lê o RATIO_ERRO do elemento corrente (ε_i)
         error = ResElScalarField.nod[i]
       // lê o NORM_ERRO do elemento corrente (||ē||)
         beb   = ResElScalarField.nod[i]
       // lê o NORM_TENSOR do elemento corrente (||û||)
         bub   = ResElScalarField.nod[i]
       // atualiza os valores das variáveis
         norm_bub = norm_bub + bub*bub;
         norm_beb = norm_beb + beb*beb;
   fim De.
// preenche o valor do erro relativo corrente
  eta_barra = sqrt(norm_beb)/sqrt((norm_bub)+(norm_beb))
  eta = (eta_barra*100)
```

```
// calcula o novo valor de 'h'
  fact = pow( error, 1.0/1 );
  hnew = hold / fact;
 // preenche o novo valor de 'h'
 s = hnew
```

Figura 25 Pseudo-código da função que preenche o valor de *h* para elementos do tipo T3.

Na função *Adapt_CreateQuadtree* mostrada na Figura 24, constrói-se a *quadtree* global, considerando ou não a adaptatividade geométrica, o que é indicado pelo valor da variável *par*0 e considerando ou não a adaptatividade baseada no erro numérico, o que é indicado pelo valor da variável *par*4.

```
// variáveis da função
  vars = n_pts      // indica o número de pontos adicionados a curva devido os
                    // critérios da adaptatividade geométrica
         inf_pts    // vetor que guardará as coordenadas Cartesianas do centro
                    // geométrico de cada trecho e o tamanho do mesmo
         i          // contador
// parâmetros da função
         nelem      // número de elementos na malha
         inf_msh // vetor que guardará as coordenadas Cartesianas do centro
                 // geométrico de cada elemento da malha e o valor de h
Se par0 = 1 constrói-se a quadtree global levando em consideração a
adaptatividade geométrica
   Para cada curva do sólido
     SdvAdaptRefineStretchGeom(par1, par2, par3, n_pts, inf_pts);
     // passa as informações de todos os trechos de cada curva para a construção
     // da quadtree
     De i = 0 até i < n_pts faz-se i = i + 1
        AddVertexSize (inf_pts)
     Fim De;
   fim Para;
fim Se;
```

```
Se par4 = 1 constrói-se a quadtree global levando em consideração a
adaptatividade baseada no erro numérico
    // passa as informações de todos os elementos da malha para
    // a construção da quadtree
    De i = 0 até i < nelem faz-se i = i + 1
       AddVertexSize (inf_msh);
    Fim De;
fim Se;
```

Figura 26 Pseudo-código da função que constrói a *quadtree* considerando ou não a adaptatividade baseada no erro numérico e na adaptatividade geométrica.

Já a função *_MtoolAdaptiveMesh* tem como tarefa:

1) Chamar a função *Adapt_CreateMesh* que efetivamente gera a nova malha conforme os parâmetros de erro numérico e geométricos.

As tarefas da função *_ MtoolAdaptiveMesh* podem ser visualizadas na Figura 27.

```
// Gera a nova malha no ambiente geométrico
  Adapt_CreateMesh ( );

 // Desaloca as memórias que foram alocadas durante a execução da função
  Adapt_End ( );
```

Figura 27 Pseudo código da função *_MtoolAdaptiveMesh*.

Na Figura 27, uma função que merece destaque é a *Adapt_CreateMesh*, pois é nesta função que se subdivide as curvas do modelo e gera-se a nova malha de superfície conforme a *quadtree* global.

4.2. *MG* (3D)

O *MG* [48] é um pré-processador que foi originalmente desenvolvido para geração de superfícies (cascas) em modelos de elementos finitos, sendo, na seqüência, estendido para considerar também malhas sólidas. O *MG* [48] incorpora duas capacidades importantes em modelagem 3D pelo MEF. A primeira está relacionada com a interface com o usuário e procedimentos gráficos interativos para gerar malhas em superfícies e a segunda relaciona-se com a interseção de superfícies paramétricas

O algoritmo para interseção de superfícies usado no *MG* [48] é baseado em um esquema para interseção de superfícies paramétricas onde as malhas de superfícies existentes são usadas como suporte para a definição inicial das curvas de interseção. Uma estrutura de dados auxiliar, definida no espaço paramétrico de cada superfície, permite a construção das curvas de *trimming* (curvas resultantes da interseção entre superfícies e que possuem representação nos espaços paramétricos destas e no espaço Cartesiano) sem a necessidade de se realizar buscas globais.

O *MG* [48] foi incialmente útil para a representação de modelos geométricos, porém sua estrutura de dados original não foi baseada em qualquer conceito formal de modelagem geométrica. Em várias ocasiões, a consistência geométrica de um modelo era realizada com a intervenção do usuário. Por exemplo, não existia a capacidade de detectar automaticamente quando um volume no espaço é definido por um conjunto de retalhos de superfícies (ou seja, superfícies limitadas pelas suas curvas de bordo). O usuário devia indicar a construção explicitamente, o que pode não ser uma tarefa trivial na modelagem de um problema real de engenharia. Essa capacidade é particularmente importante na geração de malhas formadas por elementos finitos sólidos, onde algumas vezes é desejável ter várias regiões com mapeamentos distintos, cada uma definida com suas respectivas malhas de contorno geradas por diferentes tipos de algoritmos.

Lira em 2002 [37] propôs uma técnica que permite o reconhecimento automático de regiões sólidas, adotando uma metodologia que é baseada em uma representação topológica completa de uma subdivisão espacial, chamada *Complete Geometric Complex* (*CGC*) [14].

A representação *CGC* é implementada como uma biblioteca de classes, no contexto da programação orientada a objetos (POO), provendo um conjunto de operadores de alto nível que manipulam uma subdivisão espacial. Estes

operadores recebem como parâmetros de entrada as informações geométricas dos retalhos de superfícies que são, então, inseridas na subdivisão espacial. Estas informações geométricas são automaticamente transformadas em informações topológicas requeridas pelos operadores de baixo nível. A implementação original da *CGC* [14] tratava apenas de retalhos de superfícies planas. No trabalho de Lira isso foi estendido para superfícies curvas e paramétricas.

4.2.1. Adaptatividade no *MG*

O processo adaptativo foi inserido no *MG* [48] através de módulos que foram adicionados ao programa com as seguintes tarefas. Os responsáveis por todo o gerenciamento do processo adaptativo (classe *mgAdaptive*), os responsáveis pelo refinamento da *binary-tree* e a conseqüente subdivisão das curvas e os responsáveis pelo refinamento da *octree* e o consecutivo controle do tamanho dos elementos que serão gerados nas malhas de superfície e volumétrica.

Como o *MG* [48] constitui apenas a parte do pré-processamento, pode-se implementar todo o processo adaptativo apenas no ambiente de malha.

Partindo-se do princípio que já se tem o modelo com todos os seus atributos preenchidos, seleciona-se o método de suavização e se determina à máxima razão de erro η^*. Na seção que trata dos arquivos de saída está a opção de executar a análise adaptativa com ou sem a adaptatividade geométrica.

O processo de comunicação entre os programas é feito através de arquivos neutro. O *MG* [48] fornece como dado de saída um arquivo neutro que será lido pelo *FEMOOP* [43] (módulo de análise) que por sua vez fornece um outro arquivo de saída (ver Figura 28) sobre o qual se obtêm as informações de erro que serão utilizadas para o cálculo dos novos tamanhos dos elementos.

```
// variáveis da função
  vars = xmax, ymax, xmin, ymin, n_elem, n_nodes, eta, inf_el_eachnode,
         par0, par1, par2, par3,
         eta          // retorna o erro corrente
         adpt         // objeto da classe mgAdaptive
         et  = 0,     // 0=linear,        1=quadratic
         ek  = 1,     // 0=shell,         1=solid
```

```
        ei  = 0         // 0=all model, 1=selected
        command   // variável com a extensão certa para chamar o Femoop
        f               // referência para arquivo neutro
 // captura o parâmetro que verifica se será feita a adaptatividade geométrica
 par0 = 0 ou 1

// testa se é para ser feita a adaptatividade geométrica
  Se par0 = 1 capturam-se os outros parâmetros necessários para
  a adaptatividade geométrica.
// tolerância mínima da distância entre as subdivisões estipulada pelo usuário
// do programa
     par1 = dMIN;
// tolerância máxima da distância entre as subdivisões estipulada pelo usuário do
// programa
    par2 = dMAX;
 // tolerância angular estipulada pelo usuário do programa
    par3 = angTOL
  fim Se;

  Se par0 = 0 não se fará a adaptatividade geométrica e inicializam-se
  os outros parâmetros como zero.
     par1 = 0; // tolerância mínima da distância entre os pontos
     par2 = 0; // tolerância máxima da distância entre os pontos
     par3 = 0; // tolerância angular
  fim Se;

// captura o parâmetro que verifica se será feita a adaptatividade baseada em
 // erro
 par4 = 0 ou 1

// manipula o nome do arquivo para mandá-lo de forma adequada
 // (extensão correta) ao programa de análise constrói a malha
 building ( et, ek, ei )

 // salva o arquivo usando o Esam.
 createMesh( mesh )
 writeMesh ( f )
```

```
 // chama o programa de análise (femoop)
roda(command);

 // executa o processo adaptativo
mgRegenMeshes(mesh, f, eta, par0, par1, par2, par3, par4))

// destroi a malha.
 kill();
 // fecha o arquivo '.pos'
 fclose( f );
 // chama a função em Lua que mostra o resultado de erro da análise
 ResultError (eta);
```

Figura 28 Pseudo código no *MG* que gerencia a gravação e leitura dos arquivos neutros.

O método *mgRegenMeshes* da classe *mgAdaptive* é que realmente gerencia todo o processo adaptativo. Este método é mostrado detalhadamente na Figura 29.

```
// variáveis da função
  vars = new_size, n_elements, xmax, ymax, zmax, xmin, ymin, zmin, oct,
           new_tpts, pra_val

  // obtém o número de elementos da malha
  n_elements = getNumberOfElems( );

  // cria a Bound Box
  mgGetBoundBox(n_elements, xmax, xmin, ymax, ymin, zmax, zmin);

   // Inicia a octree passando as coordenadas Cartesianas máximas
  OctreeBegin (xmax, ymax, zmax, xmin, ymin, zmin);

  // preenche os valores new_size de todos os elementos da malha e o valor do
  // erro corrente eta
   mgGetErroSizeEachElement (mesh, new_size, eta));
```

```
 // refina a octree global considerando o erro numérico e com os critérios de erro
 // geométrico das curvas, caso esta opção tenha sido selecionada, porém as
// mesmas não são subdivididas neste estágio.
 mgCreateOctree (oct, xmax, ymax, zmax, xmin, ymin, zmin, mesh,
                        new_size, par0, par1, par2, par3);

 // Caso se opte pela adaptatividade geométrica as curvas subdividas
 Se par0 = 1
     Para cada curva do sólido
       // subdividem-se as curvas com os critérios de erro e de adaptatividade
       // geométrica passando-se o número de pontos (new_pts) e as suas
       // coordenadas paramétricas (par_val).
        segmentSetSubd(new_pts, par_val);
     fim Para;
 fim Se;

 // refina a octree global com os valores passados pela quadtree auxiliar
 // considerando a curvatura das mesmas
 mgRefineOctSurface(oct);

// método responsável pela subdivisão das curvas e pela geração das novas
// malhas de superfícies e volumétricas
mgAdapt_CreateMesh ( mesh );
```

Figura 29 Pseudo código no MG que mostra o método *mgRegenMeshes*.

Todos os métodos presentes na Figura 29 são da classe *mgAdaptive*. O método *mgGetBoundBox* da Figura 29 é responsável por obter as maiores coordenadas cartesianas do modelo, informação que será usada para a construção da octree.

O método *mgGetErroSizeEachElement* é responsável por ler as informações necessárias do arquivo neutro para a estimativa de erro e por calcular os novos valores de h para cada elemento da malha, conforme a equação 34, guardando estas informações na estrutura de dados *new_size* (ver Figura 29).

O método *mgCreateOctree* recebe os valores extremos do modelo preenchidos no método *mgGetBoundBox*, cria a *octree* global e a refina de acordo com os dados contidos na variável *new_size* e com os parâmetros da adaptatividade geométrica das curvas.

O método *mgRefineOctSurface* recebe, como parâmetros de entrada, as curvas de todo modelo subdivididas conforme os critérios de erro numérico e os parâmetros da adaptatividade geométrica das curvas. Este método tem a função de refinar a *octree* global conforme os critérios de curvatura em cada superfície, conforme o algoritmo de Miranda *et al* (vide – seção 3.4).

O último método que aparece na Figura 29, dentro do método *mgRegenMeshes*, é o *mgAdapt_CreateMesh*. Ele é responsável pelo refinamento das curvas, pela geração das novas malhas de superficie e pela geração da malha volumétrica. Este método pode ser visto na Figura 30.

A Figura 30 mostra em forma de pseudo código as principais funcionalidades do método *mgAdapt_CreateMesh* da classe *mgAdaptive*.


```
// variáveis do método
seg       // é um ponteiro para um objeto da classe mgSegment
octree    // é um objeto da classe mgOctree
p2        // é um ponteiro para um objeto da classe mgPatch2d
tetra     // é um ponteiro para um objeto da classe mgTetra
n_pts     // número de pontos da subdivisão da curva
c_par     // coordenadas paramétricas dos pontos das subdivisões de cada curva

// para cada curva do modelo gera-se a nova subdivisão usando as informações
// de erro contidas na octree, preenchendo-se o número de pontos com suas
// respectivas coordenadas paramétricas através do seguinte método
// da classe mgAdaptive
mgPointsOnEdgeOCT(seg, 1, octree, n_pts, c_par)

// de posse destas informações geram-se os novos pontos da subdivisão através
// do método segmentMakeUse da classe mgSegment
segmentSetSubd(new_pts, par_val)

// com as novas subdivisões das curvas geram-se as novas malhas de superfície
// através do método patch2dReplaceMesh da classe mgPatch2d
```

```
patch2dReplaceMesh( mgGenMesh2d::CONTRACTION )

 // com as novas subdivisões das curvas e com as novas malhas de superfície
 // geradas geram-se as novas malhas volumétricas através do método
 // RebuildTetra da classe mgTetra
 RebuildTetra ( )
```

Figura 30 Pseudo código no *MG* que mostra o método *mgAdapt_ CreateMesh*, que é o método que gerencia todo o processo adaptativo.