MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE

REPRESENTAÇÃO DE CAMPOS DE INFORMAÇÃO EM APLICAÇÕES DE CIÊNCIAS ESPACIAIS E ATMOSFÉRICAS UTILIZANDO SOFTWARES DE LIVRE DISTRIBUIÇÃO

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA (PIBIC/CNPq/INPE)

Marilyn Menecucci Ibañez (UNIFEI, Bolsista, PIBIC/CNPq)
marilyn_mba@yahoo.com.br

Dr. Odim Mendes Júnior (DGE/INPE, Orientador)
odim@dge.inpe.br

Dra. Margarete Oliveira Domingues (LAC/INPE, Orientadora)
mo.domingues@lac.inpe.br

Dr. Stephan Stephany (LAC/INPE, Orientador)
stephan@lac.inpe.br

julho 2008

RESUMO

Este relatório tem o objetivo de apresentar os esforços de pesquisa para a "Representação de campos de informação em aplicações de Ciências Espaciais e Atmosféricas utilizando softwares de livre distribuição", realizada com a cooperação da Divisão de Geofísica Espacial e o Laboratório de Computação e Matemática Aplicada. A capacidade da visualização científica é extremamente necessária nas áreas de estudo de fenômenos complexos, tanto pelo volume de dados quanto pelos processos físicos de maior complexidade representados, e requer também o desenvolvimento concomitante de formas de lidar com os dados de análises numéricas. A metodologia consiste do uso de ferramentas de livre distribuição, do manuseio de dados espaciais e atmosféricos, e a participação em implementações de métodos numéricos de vanguarda. Como resultados foram desenvolvidos os programas para a criação de uma estratégia de resolução de EDP's por meio de técnicas waveletes utilizando a estrutura de dados *Quadtree*. Dessa forma, como um dos exemplos de aplicação, por essas características de dados e tratamento numérico que existem na área de modelagens de fenômenos eletrodinâmicos planetários, os desenvolvimentos feitos neste trabalho de iniciação científica revelam-se de grande importância, propiciando o uso ou desenvolvimento de métodos e ferramentas avançados de computação científica de interesse do INPE.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

CAPÍTULO 1

INTRODUÇÃO

A utilização das técnicas wavelets em diversas áreas da ciência tem ganhado muita importância principalmente na área de análise numérica. A solução de EDPś por meio das técnicas wavelets é de grande interesse para o desenvolvimento de métodos adaptativos.

Os métodos adaptativos apresentam as soluções como refinamento da sua entrada de dados. Este refinamento depende da regularidade em um local específico do dado.

As estruturas de dados, tal como as árvores, estão sendo bastante aplicadas na realização do refinamento dos dados de uma análise numérica. Para isto vários modelos de estruturas tem sido utilizados, como: *binary tree*, *quadtree*, *octtree*, etc.

Com base nesse contexto, o objetivo deste projeto é utilizar essas técnicas no refinamento e análise de imagens por meio de malhas adaptativas. Neste projeto utiliza-se a estrutura de dados *quadtree* na implementação do refinamento. Uma *quadtree* é, basicamente, uma estrutura de árvore que possui quatro filhos.

Para o desenvolvimento do projeto, utiliza-se a linguagem de programação C++ e a biblioteca Blitz++, que facilita a manipulação de *arrays*, na IDE Kdevelop.

Para um melhor entendimento do projeto, este relatório está dividido da seguinte forma: no 1 é apresentado o histórico da aluna durante a vigência da bolsa de iniciação científica. No 2 é apresentado a metodologia, os dados e as ferramentas utilizados no projeto. Os resultados obtidos durante o desenvolvimento do trabalho é são apresentados no 3. E por fim, a conclusão e os trabalhos futuros são apresentados no 4.

1.1 Histórico no PIBIC

A bolsista iniciou sua participação no projeto em 2003 quando começou a estudar a ferramenta OpenDX . O primeiro contato com a ferramenta foi por meio do VPE (*Visual Program Editor*), o ambiente gráfico do OpenDX . Em paralelo a esse estudo, a bolsista também iniciou a aprendizagem da linguagem de programação C. Esses estudos iniciais tiveram o intuito de desenvolver capacidades de computação científica, que auxiliam os trabalhos de Ciências Atmosféricas e Espaciais. Neste ano de 2003, obteve como resultados preliminares a criação de visualizações de dados de ocorrência de descargas elétricas e do contorno político do Brasil, bem como de seus estados.

No ano de 2004, aprimorou as visualizações existentes dos dados de descargas elétricas, de-

senvolveu visualizações da topografia dos estados do Brasil e iniciou o estudo do pacote Thor/SLA. Também em 2004, deu início ao estudo da linguagem de programação C++ e de ferramentas que auxiliam na otimização da programação como o depurador de programas GDB, os controladores de versões RCS[7] e CVS [4] e o Makefile para poder implementar o pacote Thor/SLA.

No ano de 2005 implementou aos recursos de saída do pacote Thor/SLA as visualizações feitas no OPENDX com dados de descargas elétricas provenientes do sensor *Storm*. Para realização desta implementação iniciou a utilização da programação em *script* do OPENDX .

No ano de 2006, no primeiro semestre, a aluna concluiu a implementação das funcionalidades do Thor/SLA, desenvolveu visualizações aplicadas a eletrodiâmica da Terra , aprofundou-se nos estudos da computação gráfica e aprimorou as visualizações desenvolvidas com dados de descargas elétricas. No segundo semestre, a bolsista desenvolveu visualizações com os novos tipos de dados do sensor Storm para o projeto Wotan. Para estes tipos de dados foram necessários a criação de novos programas, desenvolvidos em linguagem C++, para realizar o processamento da entrada de dados para as visualizações do OPENDX . Neste mesmo período, a aluna iniciou os estudos de uma nova ferramenta de visualização científica (ParaView) e continuou os estudos da teoria da computação gráfica [5].

No ano de 2007, no primeiro semestre, a aluna aprofundou-se nos estudos da teoria da computação gráfica [5] participando de um curso de verão entitulado *Conceitos Básicos de Computação Gráfica* no Instituto de Matemática pura e Aplicada (IMPA). Este curso foi ministrado durante os meses de janeiro e fevereiro de 2007. A bolsista também conclui as visualizações necessárias para o projeto Wotan e finalizou uma etapa inicial das visualizações para a dinâmica interplanetária. Já no segundo semestre, a bolsista iniciou o estudo das estruturas de dados de árvores quaternárias e técnicas como z-ordering para aplicação em computação gráfica na segmentação de modelos de imagens e superfícies. Também iniciou a aplicação deste conhecimento adquirido na subdivisão de malhas adaptativas [3] construídas por meio de programas desenvolvidos na linguagem C++ [8] e [9].

No primeiro semestre de 2008, a bolsista deu continuidade ao desenvolvimento do programa WABR em linguagem C++ [8] e [9] referentes a suvdivisão de malhas adaptativas. A Aluna desenvolveu, neste período, as funções referentes a criação da árvore e da busca dos vizinhos de nós da árvore.

CAPÍTULO 2

METODOLOGIA E DADOS

O projeto apresenta a construção de uma de estratégia para a resolução de equações diferencias parciais que se baseam no refinamnemto de uma malha adaptativa por meio da técnica *wavelet*. Esta malha adaptativa pode ser representada por uma matriz (NxN) ou uma função que gere esta *array*. Para o desenvolvimento do método usa-se a vantagem da estrutura de dados *quadtree* de blocos *dyadic* em regiões retangulares do plano. O processo de refinamento só é realizado dependendo da regularidade da malha em um determinado local.

No programa WABR que está sendo desenvolvido, na linguagem C++, para simular esta técnica, desenvolve-se a estrututra de dados *quadtree* inicialmente criando-se o nó principal (*root*) da árvore que recebe a malha (NxN). Para a aplicação do refinamento realiza-se uma subdivisão temporária da malha em quatro novos blocos (a ordem da disposição dos blocos obedece á técnica de *z-ordering*). Nestes blocos são armazenados as malhas referentes aos respectivos quadrantes da malha principal. As novas malhas são preenchidas por meio da interpolação dos pontos das posições ímpares da matriz. Cada valor a ser interpolado é comparado com uma constante epslon, se um valor interpolado de um quadrante for maior que a constante então um nó é alocado, uma nova malha (NxN) gerada e a malha analisada é refinada na naquele quadrante. O processo prossegue até não se satisfazer a condição da constante ou se alcançar um nível máximo estipulado de refinamento.

2.1 Dado de Entrada

A entrada de dados do programa é feita por meio de arquivos textos que descrevem a malha a ser analisada, uma função que representa uma malha, a ordem da matriz a ser analisada, o tipo de *wavelet* a ser apliada no programa para realizar o refinamento da malha.

2.2 Estrutura de Dados *QuadTree*

A estrutura de dados *QuadTree* é um árvore em que cada nó possui no máximo quatro filhos [1]. As *QuadTree* podem ser utilizadas em diversas áreas, principalmente na área gráfica, como exemplo em processamento digital de imagens, computação gráfica, banco de dados de imagens entre outras. As figuras 2.1 representa a árvore *Quadtree*.

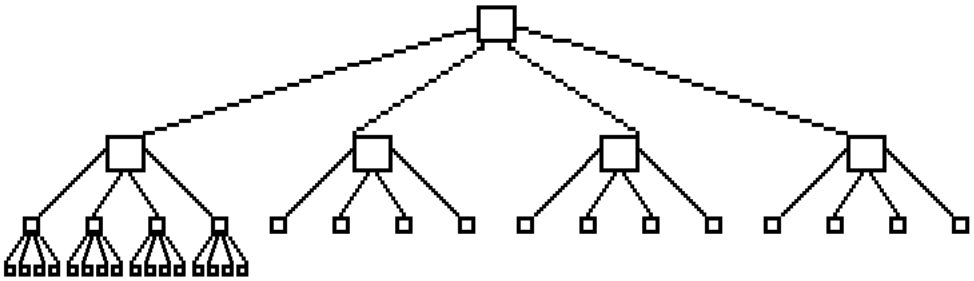

FIGURA 2.1 – Exxemplo de uma árvore *Quadtree*.

2.3 Z-Ordering

O *Z-Ordering* é um esquema de indexação para *QuadTree*. Este esquema determina como a navegação entre os nós da árvore deve ser realizada. A Figura 2.2 apresenta um exemplo desta navegação.

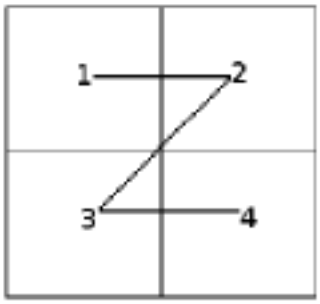

FIGURA 2.2 – Esquema de navegação *Z-Ordering*.

2.4 Malha Adaptativa

A Malha Adaptativa é um técinica que representa a solução por meio de refinamento de uma entrada de dados [2]. O processo de refinamento é realizado em determinado local da malha dependendo da regualridade que este apresenta. A Figura 2.3 mostra um malha adaptativa, referente à árvore da Figura 2.1 com maior refinamento no primeiro quadrante (vermelho) da matriz.

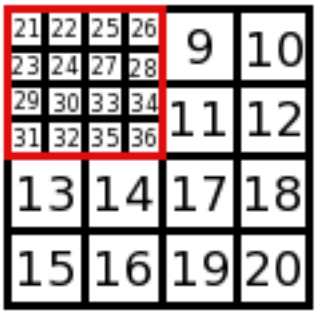

FIGURA 2.3 – Exemplo de malha adaptativa.

2.5 Vizinhança

O conceito de vizinhança aplicado no desenvolvimento do projeto consiste na busca dos nós vizinhos na malha adaptaitva a um especificado nó. Os vizinhos de um nó na malha podem estar no mesmo nível quadrante do nó, em quadrantes diferentes e em níveis difirentes. A Figura 2.4 mostra na malha os vizinhos do nó de índice 17 em suas diversas localidades.

- **Mesmo Quadrante:** 18, 19 e 20.
- **Quadrante Diferente:** 11, 12, 14,16 e 36.
- **Nível Diferente:** 36.

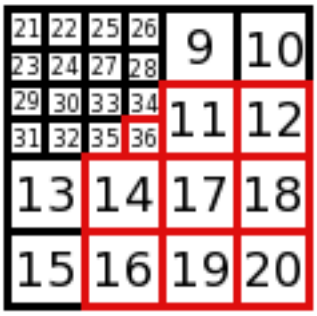

FIGURA 2.4 – Exemplo de vizinhança em uma malha adaptativa.

2.6 Ferramentas Utilizadas

2.6.1 IDE Kdevelop

Para auxiliar a bolsista no desenvolvimento do programa na linguagem C++ utiliza-se o ambiente de desenvolvimento Kdevelop 3.5 para a distribuição Kurumin 7 do sistema operacional Linux. Esta ferramenta possibilita uma maior organização das classes e arquivos do programa, praticidade para controlar as versões do programa por meio do controlador de versões CVS e facilidade na depuração e compilação do código.

2.6.2 Controlador de Versões-CVS

O CVS (Concurrent Versions System) [4] é um sistema de controle de versões que permite salvar as etapas da modificação de arquivos fontes e documentos. Ele desempenha um papel similar ao RCS, no entanto, possui algumas vantagens como: permitir que vários programadores ou um grupo de programadores mantenham a sua própria versão, que trabalhem no mesmo programa em máquinas diferentes por meio de uma rede de computadores, entre outras.

CAPÍTULO 3

RESULTADOS E ANÁLISES

Os resultados obtidos compreendem em programas desenvolvidos na linguagem de programação C++. Para o seu desenvolvimento inicialmente fez-se a análise dos seus requisitos e a modelagem do programa e por fim a implementação do código.

3.1 Modelagem

A modelagem do programa consiste na análise dos requisistos do sistema e na criação neste caso dos diagramas caso de uso, de classe e componente. As Seções 3.1.1, 3.1.2 e 3.1.3 apresentam as descrições destes diagramas.

3.1.1 Diagrama de Casos de Uso

O diagrama de casos de uso especifica uma seqüência de ações , inclusive varianates, que um sistema realiza e que produz um observável resultado de valor para um particular ator [6]. Desta forma, neste sistema o ator pode realizar a seguintes ações.

- **Definir o modelo físico:** este modelo é representado no sistema pela definição das condições iniciais, de fronteira e das equações diferencias parciais utilizadas.
- **Definir o modelo numérico:** o modelo numérico refere-se a definição de constantes matemáticas utilizada no sistema.
- **Definir o modelo computacional:** este modelo representa o programa desenvolvido.

A Figura 3.1 mostra este diagrama aplicado no sistema.

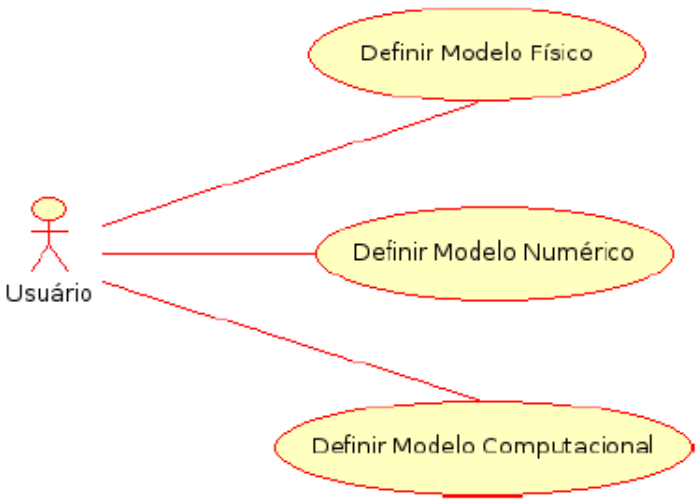

FIGURA 3.1 – Diagrama de Casos de Uso do programa.

3.1.2 Diagrama de Classes

Um diagrama de classes descreve os tipos de objetos no sistema e os vários tipos de relacionamento estático existente entre eles, bem como atributos e operações de uma classe e as restrições [6]. A Figura 3.2 mostra este diagrama aplicado no sistema.

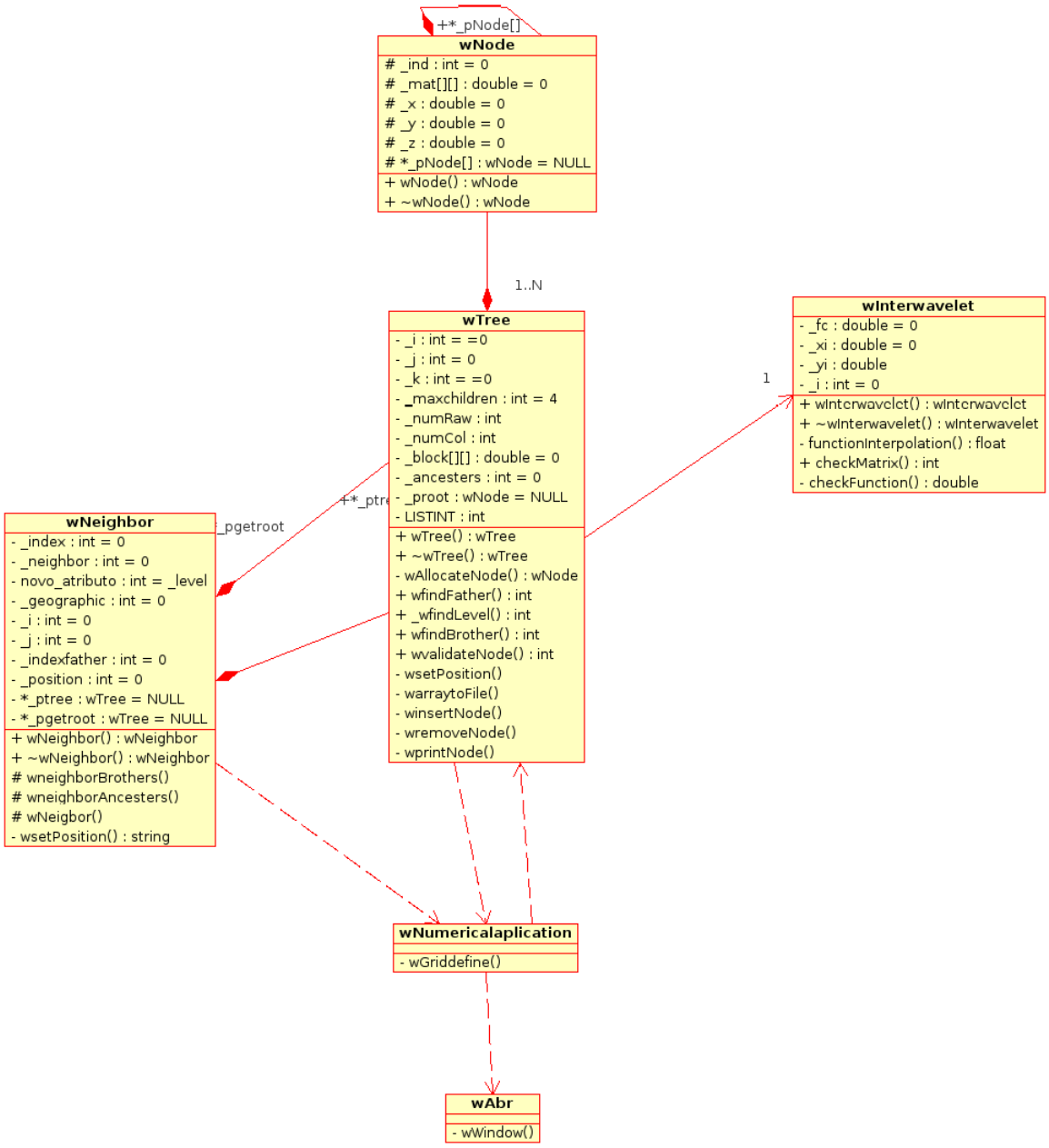

FIGURA 3.2 – Diagrama de Classe do programa.

3.1.3 Diagrama de Componente

O diagrama de componente descreve os componentes de software e suas dependências entre si, representando a estrutura do código gerado. Os componentes são a implementação na arquitetura física dos conceitos e da funcionalidade definidos na arquitetura lógica (classes, objetos e seus relacionamentos). Eles são tipicamente os arquivos implementados no ambiente de desenvolvimento [6]. A Figura 3.3 mostra este diagrama aplicado no sistema.

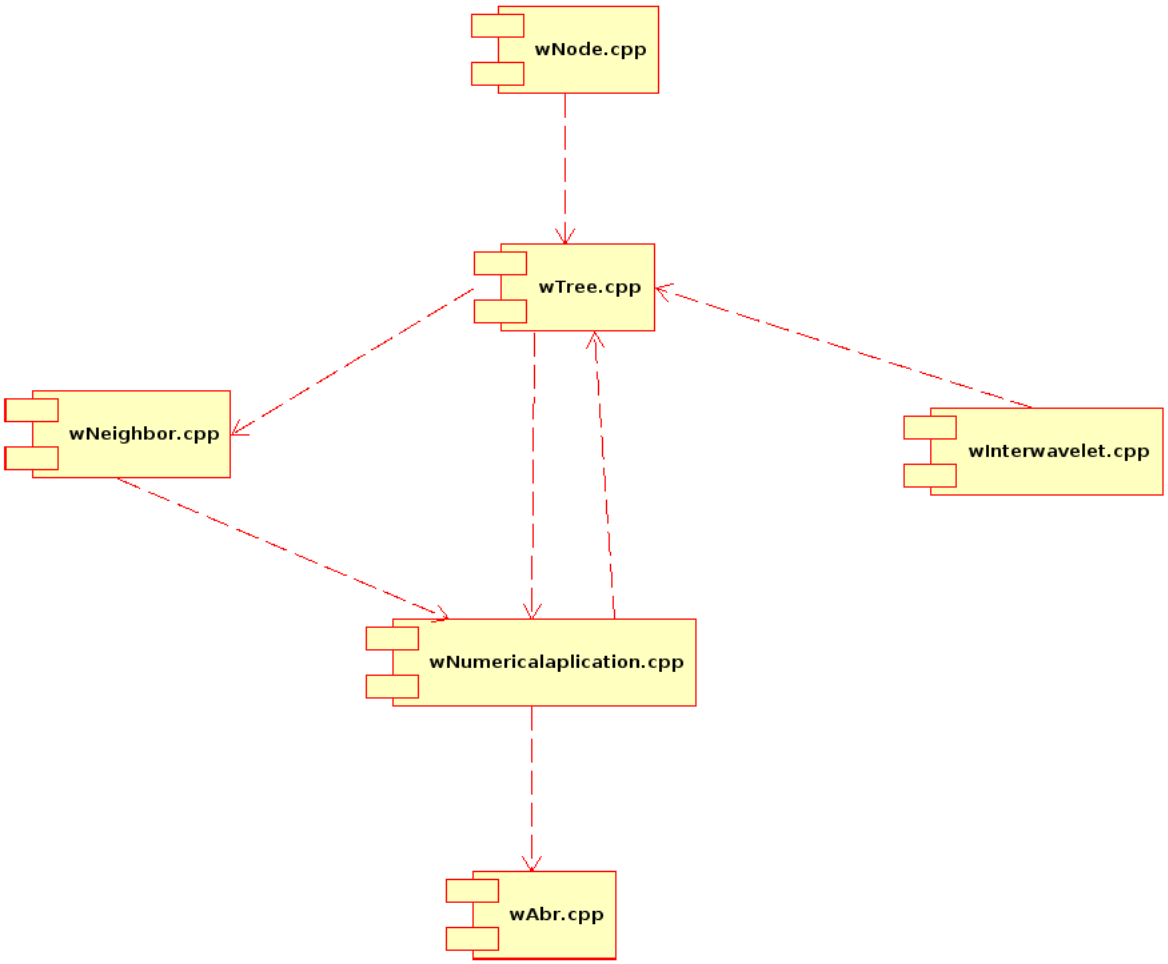

FIGURA 3.3 – Diagrama de Componente do programa.

3.2 Desenvolvimento do Programa

O programa foi desenvolvido de acordo com os diagramas elaborados e utilizando a linguagem de programação C++ na IDE Kdevelop 3.5. As classes desenvolvidas no programa até o presente momento foram wNode(), wTree(), wInterwavelet(), wNeighbor(). A seguir é apresentada uma explicação sobre como desenvolveu-se cada classe.

- **Clasee wNode():** Esta classe contém a estrutura do nó da árvore. Cada nó possui informações sobre o número do índice, a matriz de valores, posição *xyz* do nó na árvore e ponteiro para o próximo bloco.

- **Classe wTree():** A classe wTree() contém as funções de criação e manipulação da árvore. A classe wTree() herda as características da clasee wNode().Para chegar a função final de construÃ§Ã£o da árvore *quadtree* foram desenvolvidas diversas versões cada uma com suas propriedades características que nÃ£o chegavam a atender as necessidades do programa. A seguir é apresentado as principais características de cada versão.

  A primeira versão construía a árvore retornando ao nó raiz a cada nível criado. Este processo dificultava o gerenciamentos dos nós, por causa da quantidade de filhos gerados a cada loop, e também aumentava o tempo de processamento do programa.

  A segunda versão desenvolvida utilizava-se de uma estrutura que possui um ponteiro para caminhar na árvore. Este ponteiro retornava para o nó pai do respectivo nó. Porém, esbarrou na mesma dificuldade que versão anterior que foi o gerenciamento da quantidade de nós filhos criados a cada loop. Neste caso, este problema impossibilitava o controle de quantos níveis se deveria subir a cada loop.

  A versão final utiliza-se da técnica de programação dinâmica para auxiliar na construção da árvore por nível. Esta técnica permite por meio de uma tabela armazenar dados para serem usados posteriormente. Neste caso, é armazenado os endereços de memória de todos os nós criados em um nível. Estes dados de memória são utilizados para determinar os próximos nós a serem criados (filhos dos nós da tabela). A cada loop os nós da tabela são substituídos pelos seus filhos. As principais funções presentes nesta classe são: wConvert(), wReadFile(), wresizeVect(), wresizenewVect(), winsertVector(), wgetVector(), wCreateTree(), wRemoveNodes() e wRefine().

  wConvert(): Converte um número em ponto flutuante para inteiro.

  wReadFile(): Realiza a leitura do arquivo de entrada que possui a matriz quadrada NxN.

wresizeVect(): Redimensiona o vetor denominado *vect* de acordo com a necessisdade do programa.

wresizenewVect(): Redimensiona o vetor auxiliar denominado *newvect* de acordo com a necessisdade do programa.

winsertVector(): Insere os novos nós alocados no vetor *vect*. É esta função que realiza a técnia de programação dinâmica.

wgetVector(): Retorna o vetor gerado pela função winsetVecto().

wCreateTree(): Gera a árvore utilizando alocação dinâmica de memória. A cada novo nó inserido na árvore o campo de informação, no qual se insere o valor da matriz, do seu nó pai é anulado. Desta forma, somente os nós folhas da árvore contém a matriz, porém pode se construir facilmente os ascendentes destes nós. A matriz é preenchida fazendo-se a interpolação.

wRefine(): Verifica os dados da matriz de cada nó folha para ver se podem ser subdivididos (refinados). Caso possam é adicionado a árvore mais um nível. Caso contrário o nó avaliado é removido.

wRemoveNode(): A função wRemoveNode() é responsável por remover somente os nós folhas da árvore e fazer as manipulações necessárias para se mater a estrutura da árvore.

- **Classe wInterwavelet():** Nesta classe é calculada a interpolação dos pontos ímpares da matriz. Estão sendo implementados dois tipos de interpolação (linear e cúbica). Esta classe possui as seguintes funções functionInterpolation(), checkMatriz(), checkFunction().

  functionInterpolation(): Esta função calcula a interpolação dos pontos ímpares utizando pontos pares da matriz de dados. A função apresenta dois tipos de interpolação a linear, que utiliza dois pontos pares da matriz para o calculo e a cúbica que utiliza quatro pontos pares. Para o cálculo também deve se considerar a posição dos pontos da matriz. Por exemplo,

  checkMatriz():Nesta função verifica-se se o valor calculado na interpolação é maior ou menor a uma constante epslon Caso a condição não seja satisfeita o valor caculado na interpolação não é inserido na matriz.

  checkFunction: Compara a função de avaliação com o valor *epslon*

- **Classe wNeighbor():** Também é importante saber quem é o vizinho de cada nó. Assim, esta função procura o vizinho de um nó solicitado. Para o cálculo dos vizinhos utiliza-se a idéia de uma técnica de desenvolvimento de algoritmos denominada programação dinâmica. A programação dinâmica calcula a solução de todos

os subproblemas, partindo dos subproblemas menores para os maiores, armazenando o resultado em uma tabela. Assim, uma vez que o subproblema é resolvido a resposta é armazenada e nunca mais recalculada [zivianni]. Esta classe possui as seguintes funções neighborBrothers(), neighborAncesters(), neighbors().

neighborBrothers(): Esta função encontra os vizinhos referentes aos irmãos dos nós.

neighborAncesters(): Esta função encontra os vizinhos referentes aos demais ancestrais do nó.

neighbors():Esta função chama a neighborBrothers() e neighborAncesters() e encontra todos os vizinhos dos nós

CAPÍTULO 4

CONSIDERAÇÕES FINAIS

O projeto desenvolvido neste período de bolsa consiste na aplicação das técnicas *Wavelet* de refinamento de malha adaptativa para a resolução de equações diferenciais parciais utilizando a estrutura dados *Quadtree*

Na implementação do projeto utiliza-se a linguagem de programação C++ no desenvolvimento das funções de criação da árvore e de busca dos vizinhos de um nó. Algumas técnicas de otimização de algoritmo foram utilizadas durante a implemntação do programa como a técnica de programação dinâmica.

4.1 Perspectivas Futuras

A continuidade do projeto desenvolvido será realizada aplicando o conhecimento adiquirido no desenvolvimento de estratégias para solução de EDP's por meio das técnicas *wavelets* utilizando a estutura de dados de árvores *Octree* dentro do progrma de pós-graduação.

4.2 Conclusão

A bolsista obteve como resultados o desenvolvimento da modelagem do programa e do código em linguagem C++ das funções para inserção de um nó, remoção de um nó folha, refinamento da árvore, interpolação de dos pontos na posicão ímpar na matriz de dados e listagem dos vizinhos de um nó solicitado. Também se aprofundou nos estudos de técnicas de programação, estruturas de dados e malhas adaptativas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] L. Balmello, J. Kovacevié, and M. Vetterli. *Quadtrees for embedded surface visualization: Constraints and efficient data structures*. FranÃ§a, 2005. 5

[2] M. O. Domingues, S. M. Gomes, and M. A. Diaz. *Análise Wavelet na Simulação Numérica de Equações Diferenciais Parciais com Adaptabilidade Espacial.* Universidade Estadual de Campinas - Departamento de Matemática Aplicada, 2001. PhD Thesis. 6

[3] M. O. Domingues, S. M. Gomes, and M. A. Diaz. *Adaptative wavelet representation and differentiation and block-structure grids.*, 2005. 4

[4] GNU. Cvs, 2005. <http://www.gnu.org/software/cvs/cvs.html>. 4, 8

[5] J. Gomes and Luiz Velho. *Fundamentos da Computação Gráfica*. Computação e Matemática. Instituto Nacional de Matemítica Aplicada - IMPA, Rio de Janeiro, RJ, 2003. 603p. 4

[6] R. C. Lee and W. M. Tepfenhart. *UML e C++ - Guia Prítico de Desenvolvimento Orientado a Objeto*. Makron Books, São Paulo, SP, 2002. 550p. 9, 10, 12

[7] Mike Loukides. *Programando Com Ferramentas GNU*. CONECTIVA INFORMATICA, 2000. 269p. 4

[8] S. Oualline. *Practical C++ Programming*. O'Reilly, 1997. 4

[9] G. Satir and D. Brown. *C++: The Core Language*. O'Reilly, 1995. 4

CAPÍTULO A

CÓDIGO FONTE DO PROGRAMA

A.1 Classe wNode.h

A classe *wNode.h* é responsável por definir os componentes da estrutura de cada nó criado na árvore.

```
#ifndef WNODE_H
#define WNODE_H

#include "wGeneral.h"

class wNode{
public:
int _ind;
Array2D _mat;
double _x;
double _y;
double _z;
wNode *_pNext[5];
public:
     wNode(void){
//starting matrix
Range r(-2,34);
_mat(r,r) = 0;

//resize matrix
_mat.resize(r,r);
_mat = 0;};
     ~wNode();

};
```

A.2 Classe wTree.cpp

A classe *wTree.cpp* possui todas os métodos responsáveis pela criação e manipulação da árvore. As funções presentes na classe são apresentadas a seguir.

A.2.1 *Método wAlocateNode()*

Este método realiza a alocação dinâmica de memória para a construção e refinamento da árvore.

```
wNode * wTree::wAlocateNode(){

return (new wNode());
}

\end{document}
```

A.2.2 *Método wReadFile ()*

O método *wReadFile()* faz a leitura dos dados de entrada representados por uma matriz (NxN).

```
void wTree::wReadFile ( char *FileName )
{
TinyVector<int, 2> index;

ifstream pIn, auxpIn;

//Open File
pIn.open ( FileName,ios::in );

if ( !pIn )
{
cout<<"Error opening Data File (1)!!!"<<endl;
return;
}

//Read File and storage in matrix
while ( pIn )
{
index = _i, _j;

pIn>>_block ( index );

_j++;
cout<<"j= "<<_j<<endl;
if ( _j== _numCol-2 )
{
```

```
_i++;
_j=-2;
}
}
//Close File
//cout<<_block<<endl;
pIn.close();
```

À.2.3 *Método wresizeVect()*

Este método redimensiona o tamanho do vetor utilizado para armazenar os ponteiros.

```
void wTree::wresizeVect(double level){
_vect.clear();
_vect.resize(wConvert(pow(4.0,level)));
```

À.2.4 *Método winsertVect()*

Este método insere os ponteiros no vetor.

```
void wTree::wresizeVect(double level){
_vect.clear();
_vect.resize(wConvert(pow(4.0,level)));
```

À.2.5 *Método wCreateTree()*

O método *wCreateTree* é responsável por gerar a árvore até um nível determinado pelo usuário.

```
void wTree::wCreateTree ( wNode *pRoot, int level, char *FileName ){

wInterwavelet inter;

wfunctionMatrix();

pOut<<_block<<endl;



do{
if ( !_pRoot ){
_pRoot = wAlocateNode();
_pRoot->_ind = 0;
_pRoot->_mat = _block;
wresizeVect(0.0);
```

```
wresizenewVect(0.0);
for(_i=0;_i<_MAX;_i++){
_pRoot->_pNext[_i] = NULL;
}
winsertVector(_pRoot);
wsetRoot(_pRoot);

}
else{
_i=1;
_j=0;
_newvect = wgetVector();
_vect.resize(1);

while(_i<_newvect.size() ){
_block = _newvect[_i]->_mat;
if ( inter.checkMatrix ( _block, 0.5, 2 ) !=0 ){
for(_j=1;_j<=_MAX;_j++){
_newvect[_i]->_pNext[_j] = wAlocateNode();
_newvect[_i]->_mat = 0;
_newvect[_i]->_pNext[_j]->_ind = _newvect[_i]->_ind*4+_N;
_newvect[_i]->_pNext[_j]->_mat = inter.getArray ( _j );
for(_k=1;_k<_MAX;_k++){
_newvect[_i]->_pNext[_j]->_pNext[_k] = NULL;
_newvect[_i]->_pPrevious = _newvect[_i];
}
/*cout<<_newvect[_i]->_pNext[_j]->_mat<<endl;
cout<<_vect.size()<<endl;*/
winsertVector(_newvect[_i]->_pNext[_j]);

if(_count==level-1) wsetVectorRefine(_newvect[_i]->_pNext[_j]);
pOut<<_newvect[_i]->_pNext[_j]->_mat;
}
}
_i++;
_N++;
}
```

```
_newvect.resize(1);
}
_count =_count + 1.0;
}while ( _count <= level );
pOut.close();
}
```

A.2.6 *Método wRefine()*

Este método realiza o refinamento da árvore para que se possa verificar se existe a necessidade de adicionar mais um nível na árvore.

```
void wTree::wRefine(){

_i=1;
_j=0;
_newvect = wgetRefineVector();
wInterwavelet inter;

_refinevect.resize(1);
while(_i<_newvect.size() ){
_block = _newvect[_i]->_mat;
if ( inter.checkMatrix ( _block, 0.5, 2 ) !=0 ){
for(_j=1;_j<=_MAX;_j++){
_newvect[_i]->_pNext[_j] = wAlocateNode();
_newvect[_i]->_mat = 0;
_newvect[_i]->_pNext[_j]->_ind = _newvect[_i]->_ind*4+_N;
_newvect[_i]->_pNext[_j]->_mat = inter.getArray ( _j );
for(_k=1;_k<_MAX;_k++){
_newvect[_i]->_pNext[_j]->_pNext[_k] = NULL;
_newvect[_i]->_pPrevious = _newvect[_i];
}
/*cout<<_newvect[_i]->_pNext[_j]->_mat<<endl;
cout<<_vect.size()<<endl;*/
//pOut<<_newvect[_i]->_pNext[_j]->_mat;
}

}else{
```

```
wRemove(_newvect[_i]);

}
_i++;
_N++;
}
_newvect.resize(1);
```

A.2.7 *Método wRemoveTree()*

Este método remove os nós folhas da árvore caso não precise realizar o refinamento no dermi-nado nó.

```
void wTree::wRemove(wNode *pointer){

wInterwavelet inter;

for(_j=1;_j<=_MAX;_j++){
inter.wMountmatrix(pointer->_pNext[_j]->_mat);
pointer->_pNext[_j] = NULL;
}

pointer->_pPrevious->_mat = inter.wGetmountedmatrix();
}
```

A.2.8 *Método wfunctionMatrix()*

Este método gera uma matrix de entrada de dados por meio de uma equação diferencial parcial.

```
void wTree::wfunctionMatrix(){

float euler = 2.71828, pi = 3.14159, x0 = 0.5, y0 = 0.5, fxy, alfa = 3.0;
TinyVector<int, 2> index;

for(_i=-2; _i<_numRaw-2;_i++){
for(_j=-2; _j<_numCol-2;_j++){
index = _i,_j;
_block(index) = 0.2*sin(2*pi*_i)*sin(2*pi*_j)*pow(euler,((-1)*alfa*(pow(_
}
}
}
```

A.3 Classe wInterwavelet()

Esta classe possui os métodos responsáveis pela interpolação e subdivisão da matriz de dados.

A.3.1 Método $setArray(Array2D_mat)$

Este método realiza uma subdivisão temporária da matriz para que se possa avaliar se realmente é necessária efetuar a subdivisão da matriz em questão.

```
void wInterwavelet::setArray(Array2D _mat){

TinyVector<int, 2> index0, index1;
int valor;

//first square

for(_i=-2, _k=-2; _i< numRow-2;_i=_i+2, _k++){
for(_j=-2, _w=-2; _j< numCol-2 && _w<numCol/2-1;_j=_j+2, _w++){
index0 = _i, _j;
index1 = _k, _w;
_mat1(index0) = _mat(index1);

}
}

//second square
for(_i=-2, _k=-2; _i< numRow-2;_i=_i+2, _k++){
for(_j=-2, _w=numCol/2-2; _j< numCol-2 && _w<numCol-1;_j=_j+2, _w++){
index0 = _i, _j;
index1 = _k, _w;
_mat2(index0) = _mat(index1);

}
}

//third square
for(_i=-2, _k=numRow/2 - 2; _i< numRow-2;_i=_i+2, _k++){
for(_j=-2, _w=-2; _j< numCol-2 && _w<numCol/2-1;_j=_j+2, _w++){
index0 = _i, _j;
index1 = _k, _w;
```

```
_mat3(index0) = _mat(index1);

}

}

//forth square
_w=numCol/2 - 1;
_k=numRow/2 - 1;
for(_i=-2, _k=numRow/2 - 2; _i< numRow-2;_i=_i+2, _k++){
for(_j=-2, _w=numCol/2 - 2; _j< numCol-2 && _w<numCol-1;_j=_j+2, _w++){
index0 = _i, _j;
index1 = _k, _w;
_mat4(index0) = _mat(index1);
}
}
```

À.3.2 Método $functionInterpolation(Array2D_mataux, inti, intj, intinterpolationorder)$

O método *functionInterpolation()* efetua a interpolação dos dados da matriz de acordo com a ordem de interpolação escolhida. O método realiza a interpolação de forma linear e cúbica.

```
double wInterwavelet::functionInterpolation(Array2D _mataux, int i, int j

order = interpolationorder;
TinyVector<int, 2> index0, index1, index2, index3, index4;

switch(order){

case 2: //Linear Interpolation

//Odd Column
if((i%2==0)&&(j%2!=0)){
index1 = i, j-1;
index2 = i, j+1;
fc = (_mataux(index1)+_mataux(index2))/2;
return setFloatPoint(fc);
}

//Odd Row
```

```
if((i%2!=0)&&(j%2==0)){
index1 = i-1, j;
index2 = i+1, j;
fc = (_mataux(index1)+_mataux(index2))/2;
return setFloatPoint(fc);
}

//Odd Row and Column
if((i%2!=0)&&(j%2!=0)){
index1 = i, j-1;
index2 = i, j+1;
fc = (_mataux(index1)+_mataux(index2))/2;
// cout<<"7: "<<fc<<endl;
//numbers after comma iqual zero
return setFloatPoint(fc);
}
break;

case 4://Cubic Interpolation

if((i%2==0)&&(j%2!=0)){

if((j-2)<-2){
index1 = i, numCol-1;
index2 = i, j+1;
index3 = i, j-1;
index4 = i, j+2;
fc = ((-1.0/16.0)*(_mataux(index1)+_mataux(index2)))- ((9.0/16.0)*_mataux

return fc;
} else if ((j+2)>numCol-1){
index1 = i, j-2;
index2 = i, j-1;
index3 = i, j+1;
index4 = i, 0;
fc = (-1.0/16.0*(_mataux(index1)))- (9.0/16.0*_mataux(index2))+ (9.0/16.0
```

```
return fc;
}else {
index1 = i, j-2;
index2 = i, j-1;
index3 = i, j+1;
index4 = i, j+2;
fc = (-1.0/16.0)*(_mataux(index1) )- (9.0/16.0)*_mataux(index2)+ 9.0/16.0

return fc;
}
}

if((i%2!=0)&&(j%2==0)){
if((i-2)<-2){

index1 = numRow-1, j;
index2 = i-1, j;
index3 = i+1, j;
index4 = i+2, j;
fc = (-1.0/16.0)*(_mataux(index1)+_mataux(index2))- (9.0/16.0)*_mataux(in

return fc;
}else if((i+2)<numRow-1){
index1 = i-2, j;
index2 = i-1, j;
index3 = i+1, j;
index4 = 0, j;
fc = (-1.0/16.0)*(_mataux(index1) )- (9.0/16.0)*_mataux(index2)+ 9.0/16.0

return fc;
}else {
index1 = i-2, j;
index2 = i-1, j;
index3 = i+1, j;
index4 = i+2, j;
fc = (-1.0/16.0)*(_mataux(index1) )- (9.0/16.0)*_mataux(index2)+ 9.0/16.0
return fc;
```

```
}
}

if((i%2!=0)&&(j%2!=0)){
index1 = i, j-1;
index2 = i, j-1;
index3 = i, j+1;
index4 = i, j+2;
fc = (-1.0/6.0)*(_mataux(index1) )- (9.0/16.0)*_mataux(index2)+ 9.0/16.0*
return fc;
}
break;

default:
            cout<<"Sorry!!This order of interpolation is not implemented
break;
}
```

À.3.3 Método $checkMatrix(Array2D_mat, double epslon, int interpolationorder)$

O método *checkMatrix()* verifica se a interpolação da matriz pode ser realizada de acordo com a comparação com um valor *epslon*.

```
int wInterwavelet::checkMatrix(Array2D _mat, double epslon, int interpola

int cont=0;
static int order = interpolationorder;
TinyVector<int, 2> index;

int flag = 1;

setArray(_mat);

for(int _i=0; _i<5; _i++) _square[_i]=0;


//first square
while (flag == 1){
for(int i=-2; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
```

```
if(epslon > functionInterpolation(_mat1, i, j, interpolationorder)){
 _square[0]=1;
index = i, j;
_mat1(index)= functionInterpolation(_mat1, i, j, interpolationorder);
}
}
}


for(int i=-1; i< numRow-2;i=i+2){
for(int j=-2; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat1, i, j, interpolationorder)){
 _square[0]=1;
index = i, j;
_mat1(index)= functionInterpolation(_mat1, i, j, interpolationorder);
}

}
}

for(int i=-1; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat1, i, j, interpolationorder)){
 _square[0]=1;
index = i, j;
_mat1(index)= functionInterpolation(_mat1, i, j, interpolationorder);
}

}
if(i== numRow-4){
 flag=0;

}
}
}
if (_square[0]!=1) _mat1=0;
```

```
flag=1;
//second square
while (flag == 1){
for(int i=-2; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat2, i, j, interpolationorder)){
 _square[1]=2;
index = i, j;
_mat2(index)= functionInterpolation(_mat2, i, j, interpolationorder);
}

}
}
for(int i=-1; i< numRow-2;i=i+2){
for(int j=-2; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat2, i, j, interpolationorder)){
 _square[1]=2;
index = i, j;
_mat2(index)= functionInterpolation(_mat2, i, j, interpolationorder);
}

}
}

for(int i=-1; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat2, i, j, interpolationorder)){
 _square[1]=2;
index = i, j;
_mat2(index)= functionInterpolation(_mat2, i, j, interpolationorder);
}

}
if(i== numRow-4) flag=0;
}

if (_square[1]!=2) _mat2=0;
```

```
}

if (_square[1]!=2) _mat2=0;
flag=1;
//third square
while (flag == 1){
for(int i=-2; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat3, i, j, interpolationorder)){
 _square[2]=3;
index = i, j;
_mat3(index)= functionInterpolation(_mat3, i, j, interpolationorder);
}

}
}
for(int i=-1; i< numRow-2;i=i+2){
for(int j=-2; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat3, i, j, interpolationorder)){
 _square[2]=3;
index = i, j;
_mat3(index)= functionInterpolation(_mat3, i, j, interpolationorder);
}

}
}

for(int i=-1; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat3, i, j, interpolationorder)){
 _square[2]=3;
index = i, j;
_mat3(index)= functionInterpolation(_mat3, i, j, interpolationorder);
}

}
```

```
if(i== numRow-4) flag=0;
}


}
if (_square[2]!=3) _mat3=0;

flag=1;
//fourth square
while (flag == 1){
for(int i=-2; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat4, i, j, interpolationorder)){
 _square[3]=4;
index = i, j;
_mat4(index)= functionInterpolation(_mat4, i, j, interpolationorder);
}

}
}
for(int i=-1; i< numRow-2;i=i+2){
for(int j=-2; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat4, i, j, interpolationorder)){
 _square[3]=4;
index = i, j;
_mat4(index)= functionInterpolation(_mat4, i, j, interpolationorder);
}

}
}

for(int i=-1; i< numRow-2;i=i+2){
for(int j=-1; j< numCol-2;j=j+2){
if(epslon > functionInterpolation(_mat4, i, j, interpolationorder)){
 _square[3]=4;
index = i, j;
_mat4(index)= functionInterpolation(_mat4, i, j, interpolationorder);
```

```
}

}
if(i== numRow-4) flag=0;
}

}
if (_square[3]!=4) _mat4=0;

//cheking vector lenght: if equal zero don't devide matrix
for(int _i=0; _i<5; _i++) {

if(_square[_i]!=0)cont++;

}
return cont;
}
```

A.3.4 Método $getArray(int_cn)$

Este método retorna a subdivisão da matriz de acordo com o quadrante particonado.

```
Array2D wInterwavelet::getArray(int _cn){

if(_cn == 1){

//cout<<_mat1<<endl;
return _mat1;
}
if(_cn == 2){

//cout<<_mat2<<endl;
return _mat2;
}
if(_cn == 3){

//cout<<_mat3<<endl;
return _mat3;
}
if(_cn == 4){
```

```
//cout<<_mat4<<endl;
return _mat4;
}
```

À.3.5 Método $wMountmatrix(Array2Dmatrix)$

O método *wMountmatrix()* remonta a nova matriz utilizando os valores da interpolação e dos quadrantes da matriz pai.

```
void wInterwavelet::wMountmatrix(Array2D matrix){

static int mi, mj;
int mk, mw;
TinyVector<int, 2> index, index1;

for(mi=-2, mk=-2; mi< numRow && mk<numRow; mi++, mk=mk+2){
for(mj=-2, mw=-2; mj< numRow && mw<numRow; mj++, mw=mw+2){

index = mi, mj;
index1 = mk, mw;
_mat3(index)= matrix(index1);
}
}
}
```