UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Estudo de Quadtrees para Uso em Dinâmica de Fluidos Computacional

por

Elisângela Pinto Francisquetti

Dissertação submetida como requisito parcial
para a obtenção do grau de

Mestre em Matemática Aplicada

Prof. Dagoberto Adriano Rizzotto Justo, Ph.D.
Orientador

Porto Alegre, 18 de março de 2010.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Francisquetti, Elisângela Pinto

Estudo de Quadtrees para Uso em Dinâmica de Fluidos Computacional / Elisângela Pinto Francisquetti.—Porto Alegre: PPGMAp da UFRGS, 2010.

94 p.: il.

Dissertação (mestrado) —Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2010.
Orientador: Rizzotto Justo, Ph.D., Dagoberto Adriano

Dissertação: Matemática Aplicada
Quadtree, Dinâmica de fluidos

Estudo de Quadtrees para Uso em Dinâmica de Fluidos Computacional

por

Elisângela Pinto Francisquetti

Dissertação submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Análise Numérica e Computação Científica

Orientador: Prof. Dagoberto Adriano Rizzotto Justo, Ph.D.

Banca examinadora:

Profa. Dra. Cybele Tavares Maia Vinagre
IM/UFF

Prof. Dr. João Comba
PPGC/UFRGS

Prof. Dr. Vilmar Trevisan
PPGMAp/UFRGS

Dissertação apresentada e aprovada em
18 de março de 2010.

Prof. Dr. Waldir Leite Roque
Coordenador

AGRADECIMENTOS

Em primeiro lugar agradeço à Deus que com grande graça, amor, fidelidade me conduziu em todo o caminho da minha vida, me fez sonhar e alcançar lugares que pareciam altos demais para mim, mas afirmo que sem Ele nada seria possível.

Agradeço aos meus pais, em especial, à minha mãe que é um exemplo de mulher para mim e que sem o seu apoio, confiança, dedicação eu não poderia ter realizado este projeto, pois abriu mão de muitas coisas em meu favor e nunca conseguirei retribuí-la, pois o que fizeste por mim não tem preço. Te amo muito mãe.

Agradeço com muito carinho e amor à minha grande líder, pastora Nadia Zatti, que me ensinou a andar mais pertinho de Deus e tem sido grande inspiradora na minha vida. Obrigada por acreditar e investir em mim. Obrigada por entender minhas ausências em alguns momentos importantes. A senhora é um presente de Deus na minha vida.

Também agradeço aos professores que tive ao longo da minha vida, principalmente ao professor Dr. Oclide José Dotto que acreditou em mim e me apoiou durante a minha vida acadêmica e foi de grande inspiração pra mim. Agradeço ao meu orientador Dr. Dagoberto Adriano Rizzotto Justo que acreditou em mim, me ensinou muitas coisas com sua grande paciência e por isso este trabalho se tornou possível e que com toda certeza é um exemplo de profissional para mim.

Não poderia deixar de agradecer aos colegas e amigos que fiz nesta minha caminhada. Quero agradecer o companheirismo, os momentos de reflexão, trocas de idéias, as risadas, os momentos de estudo que tive com meu amigo Manoel da Rosa Paiva Filho. Agradeço a grande paciência do meu amigo Daniel Generali Tartari, pois muitas vezes tiveste que me ensinar e repetir aqueles exercícios

complicados para mim de Análise e de outras disciplinas, poucas vezes na minha vida encontrei pessoas tão inteligentes e humildes como este meu amigo. Claro, que não posso esquecer e registrar meu agradecimento ao colega João Prolo, pois sua ajuda no estudo do `FORTRAN` foi essencial, além das inúmeras ajudas com problemas diversos que tive no LICC.

Agradeço à Greice da Silva Lorenzzetti por sua alegria, pelos materiais que me emprestaste, por sua companhia, enfim por tudo. Agradeço à Katia Arcaro que no nivelamento ia até a minha casa levar o material das aulas as quais eu não podia frequentar, muito obrigada por tudo! Também quero registrar meu agradecimento à colega Francieli Aparecida Vaz e a grande amiga Valdirene Roxo!

Agradeço ao Programa de Pós Graduação em Matemática Aplicada da UFRGS pela oportunidade e à CAPES pelo apoio financeiro.

Enfim, agradeço a todos os que tornaram este projeto possível!

Sumário

Lista de Figuras

Lista de Tabelas

RESUMO

Neste trabalho desenvolvemos um algoritmo para a geração de malhas *quadtree* com o objetivo de utilizá-las na simulação de escoamento de fluidos, onde muitas vezes faz-se necessário o uso de malhas finas. A idéia central da geração das malhas *quadtree* está baseada na estrutura de árvore quaternária em que cada nodo possui quatro filhos. Assim uma estrutura de árvore está associada a uma malha que pode ter espaçamento uniforme ou refinamento em regiões específicas.

Para evitar problemas de discretização, uma malha *quadtree* deve satisfazer um critério chamado de balanceamento e este é tratado de forma detalhada no desenvolvimento do trabalho. Podemos destacar também outro ponto importantíssimo na implementação de malhas *quadtree*, que é a busca de vizinhos dos nodos. Além disso apresentamos a discretização dos operadores em diferenças finitas, que é feita a partir do conhecimento dos vizinhos dos quadrantes da malha e de técnicas de interpolação.

A ordem do método é verificada a partir de testes com a equação do calor, tanto para malhas uniformes quanto para malhas com níveis de refinamento diferenciados e concluímos, assim, que o método desenvolvido e apresentado é satisfatório.

ABSTRACT

In this work we developed an algorithm to generate meshes *quadtree* in order to use them to simulate fluid flow, which often makes it necessary to use fine meshes. The central idea of the generation of quadtree meshes is based on the quaternary structure of the tree, where each node has four children, and thus a tree structure is associated with a mesh that can be evenly or locally refined.

To avoid problems of discretization, a quadtree mesh must satisfy a criterion called the balancing and this is addressed in detail in the development of this work. We also highlight another important point in the implementation of quadtree meshes, which is the search for neighboring nodes. Additionally, we present the finite differences discretization of operators, which uses the knowledge of the mesh quadrant neighboors and interpolation techniques.

The order of the method is checked by tests with the heat equation for both uniform meshes and for meshes with different levels of refinement and we conclude that method here presented is satisfactory.

1 INTRODUÇÃO

Os métodos de diferenças finitas e volumes finitos são os mais utilizados na busca da solução numérica de equações diferenciais parciais, pois são de fácil entendimento [5, 7, 11] e a discretização dos operadores é diretamente aplicada em domínios computacionais que apresentam contornos verticais e horizontais e cujas intersecções são ângulos retos, como: domínios quadrados, fluxo em torno de obstáculos quadrados, dutos, cavidades.

Alguns problemas computacionais surgem na simulação do escoamento de um fluxo em torno de um obstáculo circular. E por isso algumas técnicas surgiram com o intuito de sanar estes problemas. Dentre elas podemos citar:

- diferenças finitas generalizadas: com o objetivo de suavizar o contorno do obstáculo uma malha ortogonal é gerada se ajustando ao objeto. A geração das malhas pode tornar-se complicada e é necessária uma formulação diferente para o problema [19, 21, 33].

- malhas triangulares: utilizando este tipo de malhas podemos discretizar o domínio mais facilmente de forma a contornar o objeto. Entretanto o método dos elementos finitos se adapta melhor à formulação dessas malhas [12, 2, 20].

- método dos contornos virtuais: utiliza-se uma malha cartesiana e simula-se a presença do obstáculo através do correto dimensionamento de termos fonte. Se a malha não for refinada suficientemente próxima do objeto, a aproximação pode ser inadequada, não simulando de forma correta o problema [26, 22, 30].

- aproximações livres de malha ("*meshfree*"): a malha não é gerada, porém discretizações específicas devem ser obtidas para uma nuvem de pontos [23, 25, 24].

- refinamento adaptativo com malhas estruturadas: malhas cartesianas (ou triangulares) são obtidas e próximo do objeto a malha é refinada de forma a suavizar o contorno do objeto (que continuará de certa forma como um reticulado quadriculado).

Os métodos apresentados acima apresentam vantagens e desvantagens. O objetivo deste trabalho é estudar e explorar o refinamento localizado com malhas estruturadas, para num trabalho futuro desenvolvermos malhas adaptativas.

1.1 Malha *Quadtree*

O termo *quadtree* é usado para descrever a classe de estrutura de dados cuja propriedade comum é o princípio de decomposição recursiva do espaço. Primeiramente, os algoritmos *quadtree* foram utilizados no processamento de imagem e recentemente seu potencial como geradores de malha tem sido reconhecido. A abordagem mais estudada da representação de uma região em *quadtree* é baseada na subdivisão sucessiva do domínio da malha (ou limites da imagem) em quatro quadrantes de mesmo tamanho [32].

Assim, para a geração de uma malha adaptativa bidimensional ("*quadtree*"), primeiramente subdivide-se o domínio em uma malha cartesiana com espaçamentos uniformes. O obstáculo deve ser definido através de uma curva suave por partes, e em seguida a malha inicial é refinada num processo iterativo, de tal forma a subdividir os quadriláteros onde a curva está presente. Para um domínio tridimensional ("*octree*"), a subdivisão é feita em cubos.

No presente trabalho, estamos interessados em domínios bidimensionais, assim a cada varredura da malha, cada quadrilátero, que contém a curva, é subdividido em quatro quadrados iguais, sucessivamente. Tal forma de subdivisão resultará em uma malha estruturada que será armazenada numa estrutura de dados chamada *árvore quaternária* ou *quadtree*. Assim a primeira questão relevante é a armazenagem de forma dinâmica e eficiente de tal estrutura, que na prática deve seguir alguns critérios como, por exemplo, a eficiência no acesso a células vizinhas de qualquer célula da malha e de qualquer nível [27].

O critério de refinamento pode tornar-se complicado, pois ao refinar um certo quadrilátero podemos ter que refinar os seus vizinhos e assim sucessivamente, por todo o domínio. O refinamento é feito como pré-processamento para a simulação porém tal técnica pode ser utilizada para refinar partes do domínio onde a solução possuir gradiente alto, obtendo assim uma melhor aproximação.

Existem alguns métodos de decomposição planar, em [32] encontramos uma justificativa para a decomposição *quadtree* em quadrados. Quadrados são usados porque a decomposição resultante satisfaz duas propriedades:

- ela produz uma partição que é um padrão repetido infinitamente para que ela possa ser usada em domínios de qualquer tamanho;
- ela gera uma partição que é infinitamente decomposta em padrões cada vez mais finos o que resulta numa região mais refinada e portanto, curvas por parte cada vez mais suaves.

Uma decomposição em *quadtree* em quatro triângulos equiláteros também satisfaz estes critérios. No entanto, ao contrário da decomposição em quadrados, eles não têm uma orientação uniforme. Já a decomposição em hexágonos, por exemplo, possui uma orientação uniforme, mas não satisfaz a segunda propriedade.

Algumas das vantagens de utilização de uma malha *quadtree*, são as seguintes: são geradas automaticamente sobre qualquer domínio, não importando a complexibilidade e irregularidade, fornecendo regiões de alta resolução onde estas forem necessárias; por exemplo, a malha é fina em regiões onde o gradiente tem grandes variações. A estrutura hierárquica permite obter informação de forma estruturada de qualquer célula, além de ser robusta e de eficiente implementação [14, 34].

Veja a figura 1.1 que mostra um exemplo de uma malha *quadtree* com obstáculos circulares, onde próximo aos círculos a malha é mais refinada e conforme os quadrados afastam-se desta região, os espaçamentos aumentam. Desta forma é possível obter uma malha com uma boa resolução sem refinamento completo do domínio.

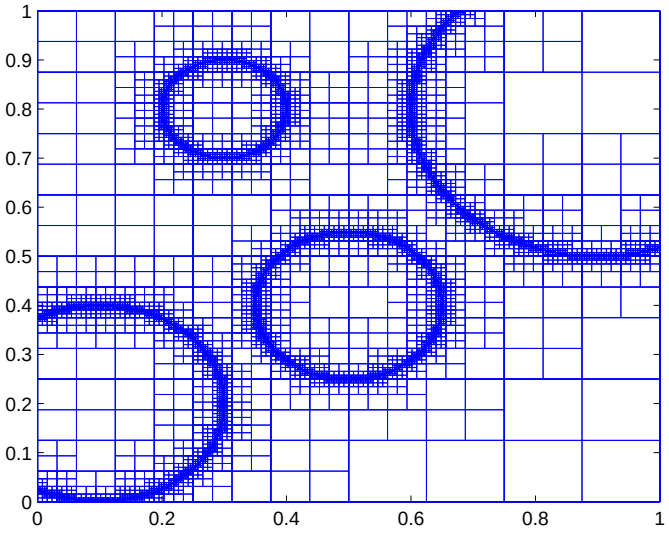

Figura 1.1: *Exemplo de uma malha quadtree.*

As malhas *quadtree* possuem grande aplicação nos problemas de dinâmica de fluidos, onde são necessárias malhas muito finas que sejam eficientes na captação

de vórtices, pois geralmente nestes problemas há grandes variações dos gradientes envolvidos.

1.2 Estrutura da Dissertação

Neste trabalho desenvolvemos uma estrutura de dados "quaternária" para a geração de malhas *quadtree*. Para isto usamos `FORTRAN 90` como linguagem de programação e os programas `MATLAB` e `Visual` [18] para o pós-processamento de imagens.

No capítulo 2 iniciamos com algumas definições básicas sobre estrutura de dados e apresentamos os termos usados ao longo da dissertação. No que se refere especificamente à estrutura da árvore quaternária implementada, apresentamos o sistema de numeração de nodos adotado no desenvolvimento do trabalho e um dos pontos altos na implementação deste tipo de árvore, que é a busca de vizinhos dos nodos. Os algoritmos de inicialização da árvore, subdivisão de um nodo e busca de vizinhos são apresentados de forma resumida.

Em seguida, diferentes processos de refinamentos são apresentados no capítulo 3. Diferentes tipos de malhas são geradas através do desenvolvimento de subrotinas que refinam ao longo de retas, círculos, rosáceas, regiões. A geração das malhas *quadtree* deve satisfazer o critério de "balanceamento" que indica a relação de subdivisão que existe entre nodos vizinhos, de forma a evitar problemas de instabilidade e discretização de operadores. A metodologia empregada na implementação do algoritmo que realiza o balanceamento da malha é explicada com alguns detalhes. Ressaltamos que não estamos apresentando, talvez, a forma mais eficiente de implementação, pois o nosso interesse, num primeiro momento, está focalizado no desenvolvimento de um algoritmo que seja capaz de gerar malhas *quadtree* e indicar as discretizações de operadores nestas malhas.

Um ponto crucial no desenvolvimento das malhas *quadtree* são as interpolações das variáveis entre quadrantes com espaçamentos diferentes. Dois tipos de interpolações são descritas no capítulo 4, bem como as discretizações em diferenças finitas para derivadas de primeira e segunda ordem são apresentadas.

Finalmente no capítulo 5 apresentamos os resultados obtidos na simulação da equação do calor e de escoamento de fluidos através da equação de Navier-Stokes. Inicialmente os modelos matemáticos e as metodologias adotadas na resolução e discretização das equações são apresentadas, juntamente com as condições de contorno e condições iniciais dos problemas propostos. Logo em seguida, a ordem do método é verificada, primeiro para o caso da malha *quadtree* uniforme e depois para malhas com diferentes espaçamentos entre os quadrantes.

2 ESTRUTURA DE DADOS

Neste capítulo apresentaremos algumas definições importantes para a compreensão dos algoritmos que serão tratados ao longo dos capítulos. Estas definições são baseadas em [15], [17] e [8]. Primeiramente definiremos o conceito de "árvore", uma das estruturas de dados mais importantes em computação, bem como a terminologia utilizada no tratamento do modelo.

2.1 Definições

Uma árvore pode ser representada graficamente como mostra a figura 2.1, onde temos um conjunto de *nodos* representados na figura pelas letras **K, L, M, N, O, P, Q, R** e **S** ligados através de linhas que indicam como os nodos estão relacionados. A terminologia para o tipo de relacionamento entre os nodos não é padronizada, portanto apresentaremos a seguir a terminologia adotada neste trabalho.

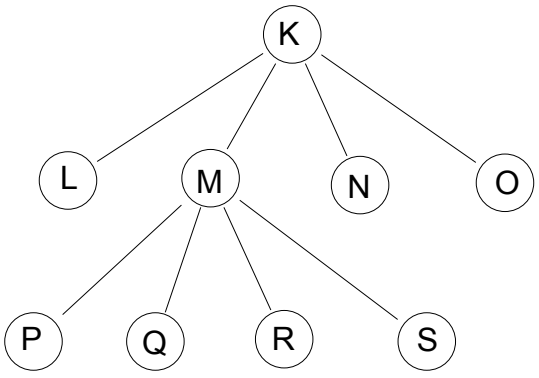

Figura 2.1: *Exemplo da representação de uma árvore.*

A *raiz* da árvore é representada no topo e cresce de cima para baixo, ou seja, na figura 2.1 a raiz é indicada pela letra **K**. É importante observar que todos os

nodos da árvore estão associados a sua raiz, portanto o acesso a qualquer nodo pode ser feito a partir da raiz. Na figura 2.1 vemos que a forma de relacionamento entre os nodos **K** e **M** é diferente do existente entre os nodos **K** e **Q**. A primeira relação é dita direta e a segunda indireta e os nodos relacionados são chamados de *nodos descendentes*. Costuma-se chamar o descendente direto de um nodo de *nodo filho*, e o nodo em questão de *nodo pai*. Assim, os nodos de mesmo pai são chamados de *nodos irmãos*. Por exemplo, na figura 2.1, os nodos **P, Q, R** e **S** são irmãos entre si e filhos do nodo pai **M**, que por sua vez é nodo filho do nodo **K**, que também é avô dos nodos **P, Q, R** e **S**.

É necessário fazer distinção entre os nodos da árvore com relação à existência de filhos: os nodos que possuem filhos são chamados de *nodos interiores* e os que não possuem, de *nodos folha* ou *externos*. Quando um conjunto de nodos está associado a um nodo folha, chamamos este conjunto de *subárvore* e o número de subárvores associadas a um nodo é chamado de *grau de um nodo*, ou podemos definir o número de filhos de um nodo como o seu grau. No exemplo da árvore representada na figura 2.2 os nodos **2, 5, 8, 9** e **10** formam uma subárvore da raiz que tem grau **3**.

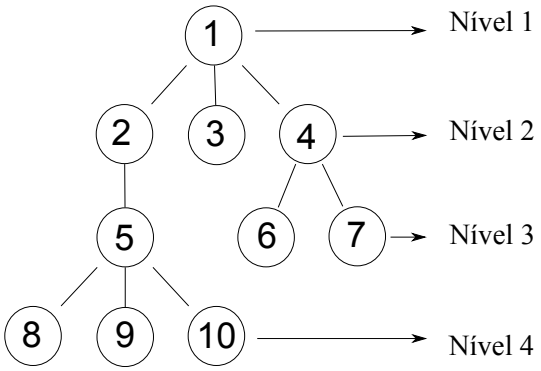

Figura 2.2: *Exemplo da representação de uma árvore e subárvores.*

Assim, também é possível definir o grau de uma árvore, que é o número máximo de filhos dentre os nodos da árvore. O grau da árvore representada na

figura 2.2 é, portanto, **3**. Outra definição muito importante é o *nível* de um nodo. Sabemos que os nodos estão ligados por linhas, o número de conecções de um nodo com outro até a raiz acrescido de 1 é dito o nível deste nodo. Usando ainda a árvore da figura 2.2 como exemplo, temos que os nodos folha **8, 9** e **10** estão no nível 4, já que existe um caminho de cada um desses nodos até a raiz constituído de 3 linhas e, portanto, acrescido 1, obtemos o nível do nodo. Desta forma, a raiz será sempre o primeiro nível.

Este trabalho está focado num tipo de árvore chamada ***quadtree***, ou seja, uma árvore onde cada nodo pai possui quatro filhos. Como o objetivo aqui, é construir malhas *quadtree*, podemos associar uma árvore com um quadrante no plano cartesiano, onde a raiz, inicialmente, será um quadrado de lado 1 com os vértices nos pontos $(0,0), (1,0), (1,1)$ e $(0,1)$. Assim podemos construir malhas uniformes ou não, com o número de nodos desejado.

É importante numerar os quadrantes gerados (nodos filhos e folhas) e associá-los com a árvore. Antes disso, cada novo quadrante é associado com um rótulo que representa uma direção dependendo de sua localização. Esta representação é mostrada na figura 2.3. Ao conjunto dos quatro quadrantes chamaremos de *bloco*.

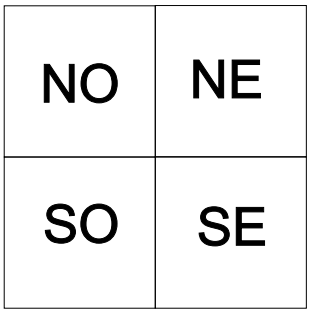

Figura 2.3: *Cada quadrante é associado com um rótulo: NO para os que estão na região noroeste do referido bloco, NE para os da região nordeste, SO para sudoeste e SE para os se encontram na região sudeste.*

O sistema de numeração adotado consiste em numerar os nodos da seguinte forma: 1 para a raiz e cada quadrante gerado, ou seja, os filhos são numerados na ordem **NO-NE-SO-SE** em cada nível. Este esquema de indexação é conhecido como *Z-Ordering*. Veja o exemplo, mostrado na figura 2.4, onde ocorre a numeração dos nodos começando pela raiz (nível 1) até a geração de nível 3. Outra forma de numeração pode ser encontrada em [15].

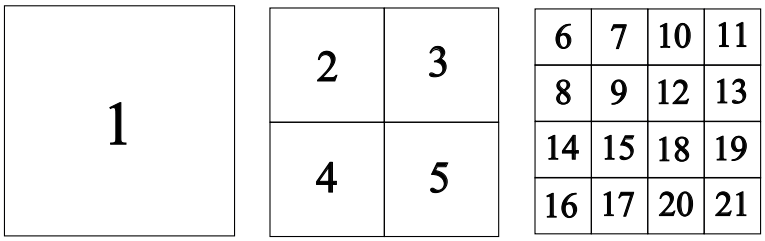

Figura 2.4: *Exemplo do sistema de numeração iniciando com a raiz até o nível 3 para o caso de uma malha uniforme.*

Agora, podemos associar o quadrante no plano cartesiano com a representação de uma árvore, seguindo o sistema de numeração acima. Desta forma, toda malha gerada pode ser associada a uma árvore e, assim, representada graficamente, como mostra a figura 2.5.

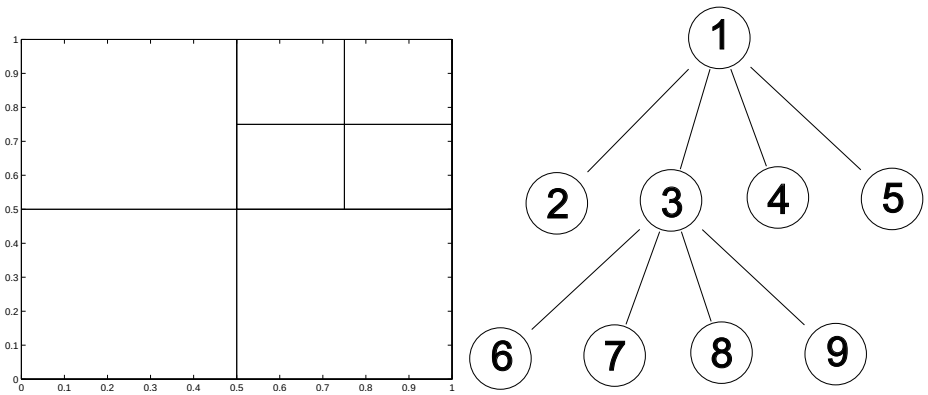

Figura 2.5: *Malha não uniforme de nível 3 (à esquerda) e sua representação gráfica através da estrutura de uma árvore (à direita).*

Para o caso de uma malha quadrada uniforme de nível n teremos 4^{n-1} nodos gerados com um número total de $\sum_{i=1}^{n} 4^{i-1} = \frac{4^n-1}{3}$ nodos na árvore, gerando assim uma malha $n \times n$ com espaçamento $dx = dy = \frac{1}{2^{n-1}}$.

Segundo [14], [15] e [29] é desejável que o nível máximo entre nodos adjacentes (vizinhos) não exceda 2 (para vizinhos de aresta a diferença é 1 e de vértice 2), caso contrário a discretização dos operadores torna-se muito complexa, o que pode dificultar a convergência. Quando uma malha satisfaz este critério dizemos então, que se trata de uma *quadtree balanceada*. A figura 2.6 apresenta um exemplo onde o critério de balanceamento é satisfeito. Os detalhes deste critério serão apresentados no capítulo 3.

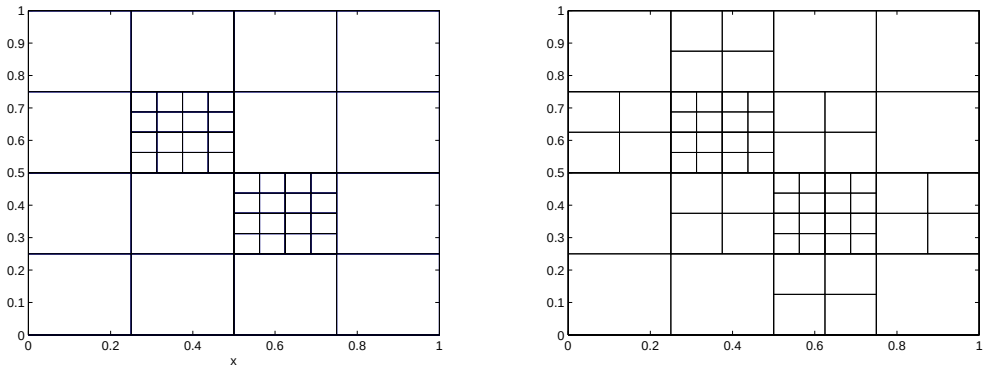

Figura 2.6: *Quadtree balanceada: à esquerda malha com níveis 2 e 4, à direita quadtree balanceada com níveis 2, 3 e 4.*

2.2 Encontrando os Vizinhos

Cada nodo ou quadrante possui, no máximo, quatro *vizinhos de aresta* e quatro *vizinhos de vértice*, onde os de aresta são chamados *Norte, Sul, Leste* ou *Oeste* e os de vértice são chamados *Noroeste, Sudoeste, Nordeste* e *Sudeste*, veja a figura 2.7. Temos interesse em saber quais são os vizinhos de aresta (observe na figura 2.7), pois estes serão usados na discretização das equações.

Figura 2.7: *Possíveis vizinhos de um quadrante: vizinhos de aresta (negrito) e vizinhos de vértice (itálico).*

Este é um ponto crucial na construção das malhas *quadtree*. Em pesquisa realizada encontramos basicamente duas maneiras para encontrar os vizinhos de aresta, que podem ser conferidas em [32, 31, 14, 15, 3, 29]. As duas primeiras referências mostram a mesma técnica, apresentada com mais detalhes no capítulo 3 em [32] que por isso nos serviu de base para o desenvolvimento do algoritmo. O grande diferencial é que esta técnica não depende do sistema de numeração adotado, nem das coordenadas dos nodos, nem do tamanho dos quadrantes, sendo assim, um método mais geral.

O método para encontrar vizinhos é baseado na localização do *ancestral comum mais próximo*. Primeiramente precisamos definir algumas funções importantes que envolvem operações entre blocos de quadrantes com suas arestas e vértices, bem como algumas notações.

Função `ADJ(D,Q)`: retorna verdadeiro (V) se, e somente se, a aresta ou vértice do **bloco** é adjacente (ou vizinho) a outro **bloco**, tomando como referência o quadrante Q na direção D, e retorna falso (F) para o caso contrário. A tabela 2.1 mostra estas relações. Outra maneira de entender esta relação é tomando por base a direção e o quadrante, ou seja, quando eles forem do mesmo "tipo" de rótulo

usamos verdadeiro. Por exemplo, `ADJ(N, NE)=V`, pois **N** e **NE** são do mesmo "tipo", já que ambos possuem a direção norte no rótulo.

Direção	Quadrante			
	NO	**NE**	**SO**	**SE**
N	V	V	F	F
E	F	V	F	V
S	F	F	V	V
O	V	F	V	F
NO	V	F	F	F
NE	F	V	F	F
SO	F	F	V	F
SE	F	F	F	V

Tabela 2.1: Tabela de adjacência

Função `PAI(i)`: retorna o pai de cada nodo i.

Função `TIPO_FILHO(i)`: retorna o tipo de filho ao qual o nodo i está associado. Sendo assim, pode ser: *NO, NE, SO* ou *SE*.

Função `FILHO(i, D)`: retorna o filho de i na direção D.

Função `REFLETE(D,Q)`: espelha o quadrante Q na direção D, isto vale para blocos de mesmo tamanho, ou seja, os blocos podem ser irmãos ou não. Este espelhamento pode ser através das arestas ou dos vértices dos quadrantes. A tabela 2.2 mostra estas relações.

Função `ARESTA_COMUM(D, Q)`: retorna a aresta comum entre Q e seu vizinho contido no bloco de mesmo tamanho na direção D, onde D pode ser entendido como o vértice do quadrante. Veja a definição na tabela 2.3.

Direção	Quadrante			
	NO	**NE**	**SO**	**SE**
N	SO	SE	NO	NE
L	NE	NO	SE	SO
S	SO	SE	NO	NE
W	NE	NO	SE	SO
NO	SE	SO	NE	NO
NE	SE	SO	NE	NO
SO	SE	SO	NE	NO
SE	SE	SO	NE	NO

Tabela 2.2: Tabela de reflexão

Direção	Quadrante			
	NO	**NE**	**SO**	**SE**
NO	ID	N	O	ID
NE	N	ID	ID	L
SO	O	ID	ID	S
SE	ID	L	S	ID

Tabela 2.3: Tabela de aresta comum, onde `ID` significa valor indefinido.

Vejamos um exemplo do uso das funções apresentadas. Consideremos uma malha de nível 2 e a árvore associada a ela, como mostra a figura 2.8.

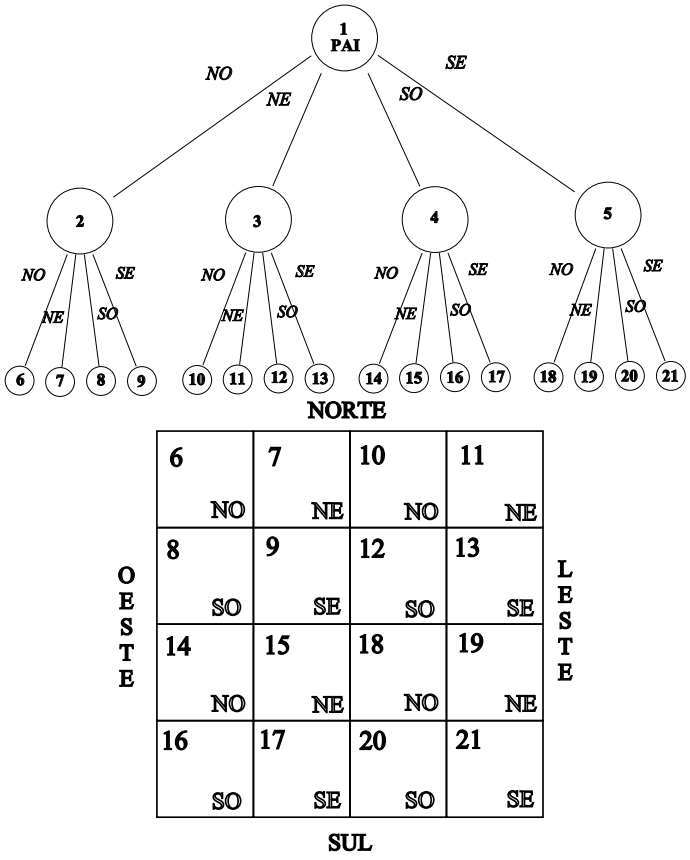

Figura 2.8: *Malha e árvore associada para exemplificar a aplicação de algumas funções importantes na busca de vizinhos.*

- `ADJ(SE, SE) = V`. Veja na figura 2.8, que o bloco que contém os nodos $6, 7, 8$ e 9 correspondente ao nodo 2 como pai. Queremos saber se o quadrante SE deste bloco (é importante observar que isto acontece em todos os blocos) é adjacente a outro bloco na direção SE. Que neste caso é verdadeiro, pois na direção especificada existe o bloco de quadrantes $18, 19, 20$ e 21. Mas se mudarmos a direção para NO, temos que `ADJ(NO, SE) = F`, pois na direção especificada estamos tratando

de um quadrante dentro de um mesmo bloco, no caso, o nodo 6 que não pertence a um bloco vizinho, portanto não há adjacência.

- `PAI(21) = 5`, significa que o nodo pai do nodo folha 21 é o nodo 5. Para observar esta relação, veja a árvore na figura 2.8, basta tomar o nodo folha 21 e subir um nível na árvore, e logo, o pai é encontrado.

- `FILHO(4, SE) = 17`, ou seja, o filho do nodo 4 na direção SE é o nodo folha 17.

- `TIPO_FILHO(5) = SE`, que significa que o nodo 5 é um filho do tipo SE já que está na direção SE do bloco ao qual pertence.

- `REFLETE(S, SO) = NO`. Observe o nodo 8 na figura 2.8 que é um quadrante SO. Agora tome a direção S. Encontramos o nodo 14 que pertence a outro bloco, e é do tipo NO. Portanto a função reflete o nodo SO na direção S e encontra um nodo do tipo NO.

- `ARESTA_COMUM(NO, SO)= O`. Veja a figura 2.8, consideremos o nodo 12 que é um quadrante rotulado por SO e que pertence ao bloco constituído pelos nodos $10, 11, 12$ e 13. Na direção NO encontramos o quadrante 7 que pertence a outro bloco (constituído pelos nodos $6, 7, 8$ e 9), portanto a aresta comum a estes blocos é a aresta da direção oeste com relação ao nodo e na direção especificados.

2.3 Implementação Básica

Nesta seção detalharemos a implementação em `FORTRAN 90` da inicialização da árvore, inserção de nodos e a procura de vizinhos utilizando as funções definidas anteriormente.

2.3.1 Inicializando a Árvore

A implementação foi feita utilizando a declaração de um objeto chamado de No através do comando `type`. As propriedades de cada No são:

`no%Nivel`: nível do nodo.

`no%NE, no%SE, no%NO, no%SO`: define-se os nodos filhos em cada direção. No inicio, atribui-se *zero* para cada um destes nodos.

`no%pai`: pai do nodo em questão. Como o nodo raiz não possui pai, então atribui-se *zero*.

`no%x, no%y`: valor das coordenadas x e y do canto inferior esquerdo do nodo pai.

`no%dx, no%dy`: comprimento dos lados do quadrante definido pelo nodo pai nas direções x e y. Na subrotina `INICIALIZA_ARVORE` apresentada logo a seguir, temos $dx = dy = 1$, portanto o nodo pai é um quadrado de lado 1.

`no%cx, no%cy`: valor das coordenadas x e y no centro do quadrante.

Foram criadas várias subrotinas arquivadas em um `module`. Iniciemos com a subrotina que inicializa a árvore, `INICIALIZA_ARVORE`, ou seja, cria o primeiro quadrante com todas as informações necessárias ao longo da implementação.

ALGORITMO: INICIALIZA_ARVORE

1 Define $no.pai = 0$

2 $no.nivel = 1$

3 $no.NE = no.NW = no.SE = no.SW = 0$

4 $no.x = no.y = 0$

5 $no.dx = no.dy = 1$

6 $no.cx = no.cy = 0.5$

A subrotina `SUBDIVIDE_NO` é responsável por inserir novos nodos na árvore, seguindo a ordem apresentada na seção anterior, bem como para refinar um nodo ou todos os nodos que derivam da subárvore gerada por esse nodo. Ela também será usada por outras rotinas que refinarão ao longo de uma reta, por exemplo. Isto será mostrado no capítulo 3 que trata do refinamento. Os passos resumidos desta subrotina são:

ALGORITMO: SUBDIVIDE_NO

Entrada: $pai = NO_i$

1 **SE** o nodo pai não possui filhos, **ENTÃO**

2 **PARA** cada filho $F \in \{NO, NE, SO, SE\}$

3 Define $F.x$ e $F.y$

4 $F.dx = \frac{pai.dx}{2}$ e $F.dy = \frac{pai.dy}{2}$

5 $F.nivel = pai.nivel + 1$

6 $F.pai = i$

7 $F.NE = F.NW = F.SE = F.SW = 0$

8 Define $F.tipofilho$

2.3.2 Procurando Vizinhos

O algoritmo abaixo foi reescrito de [32] e as funções auxiliares foram implementadas conforme definições apresentadas na seção 2.2. Algumas correções foram feitas, como por exemplo: quando procura-se o vizinho de um nodo i da fronteira, o que significa que, em pelo menos uma das direções, digamos D, não há vizinho, o algoritmo retorna 0 quando for solicitado o vizinho do nodo i na direção D.

ALGORITMO: VIZINHO

Entrada: NO_i, $Dir = Direcao$

1 **SE** $ADJ(Dir, NO_i)$, **ENTÃO**

2 Define E como o vizinho do pai do No na direcao Dir

3 **SENÃO**

4 Define E como o pai do nodo.

5 **SE** E possui nodos filhos, **ENTÃO**

6 Define $saida = FILHO(E, REFLETE(Dir, NO_i))$

7 **SENÃO**

8 Define $saida = 0$

É importante salientar que o algoritmo `VIZINHO(i, dir)`, onde i denota o nodo e *dir* a direção, encontra vizinhos considerando duas situações com relação a diferença de níveis:

1. Procura-se o vizinho de um nodo que está num nível superior. Por exemplo, veja a malha da figura 2.9. Deseja-se determinar o vizinho na direção *leste* do nodo 23, ou seja, `VIZINHO(23, 'E ')`, que é o nodo 7.

2. Procura-se o vizinho de um nodo que está num nível inferior. Um exemplo, é considerar o nodo 12 da malha representada na figura 2.9 e

procurar seu vizinho na direção *oeste*. Neste caso, o algoritmo "sobe" na árvore e procura um nodo de mesmo tamanho. Então `VIZINHO(12, 'O ')= 9`, pois 9 é o nodo pai dos nodos $26, 27, 28$ *e* 29.

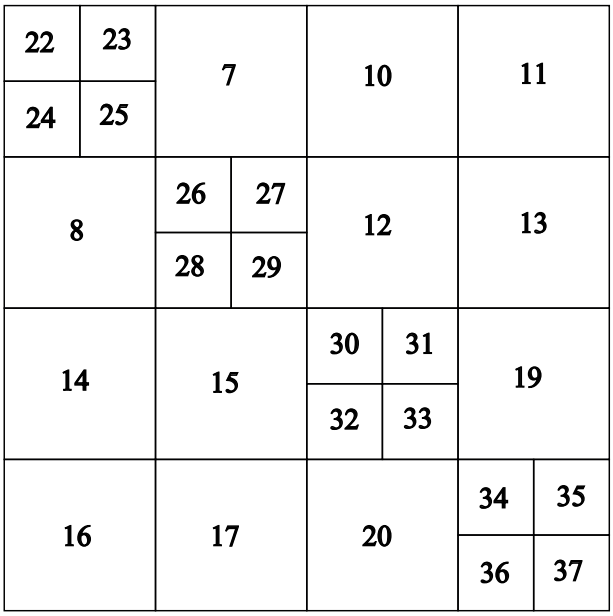

Figura 2.9: *Malha de níveis* 3 *e* 4 *com a numeração de seus respectivos quadrantes.*

O vizinho de um nodo da fronteira será, por convenção, 0, quando não houver vizinho na direção especificada. Por exemplo, `VIZINHO(11, 'E ')= 0`.

3 REFINAMENTO E BALANCEAMENTO

Este capítulo trata da implementação das subrotinas utilizadas para a geração de malhas quadtree com níveis diferentes, e ainda apresenta o critério de balanceamento da *quadtree* e sua implementação.

3.1 Refinamento

Após gerarmos a malha quadtree e até conseguirmos refiná-la em diferentes quadrantes através da subrotina `SUBDIVIDE_NO`, tornou-se necessário fazer o refinamento de forma mais rápida e mais específica, como por exemplo refinar os quadrantes ao longo de uma reta dada. Com esta idéia, o próximo objetivo do trabalho foi criar subrotinas que refinam ao longo de um círculo e de uma região para assim obtermos diferentes tipos de malhas.

ALGORITMO: REFINA

1 Calcula-se as coordenadas (x_j, y_j) utilizando a função especificada.
2 Para cada ponto (x_j, y_j)
3 Encontra-se as folhas
4 **SE** o ponto (x_j, y_j) está dentro do nodo, **ENTÃO**
5 executa $SUBDIVIDE\_NO$
6 Faça o mesmo para os nodos *NO, NE, SO e SE*

3.1.1 Refinando ao Longo de uma Reta

A subrotina `REFINA_RETA(i,a,b)` refina ao longo de uma reta de inclinação a e coeficiente linear b, começando do nodo i. A subrotina percorre a árvore iniciando do nodo pai até folhas onde é feito o refinamento. Veja na figura 3.1 uma malha gerada a partir desta subrotina, com $a = -2$ e $b = 0.9$.

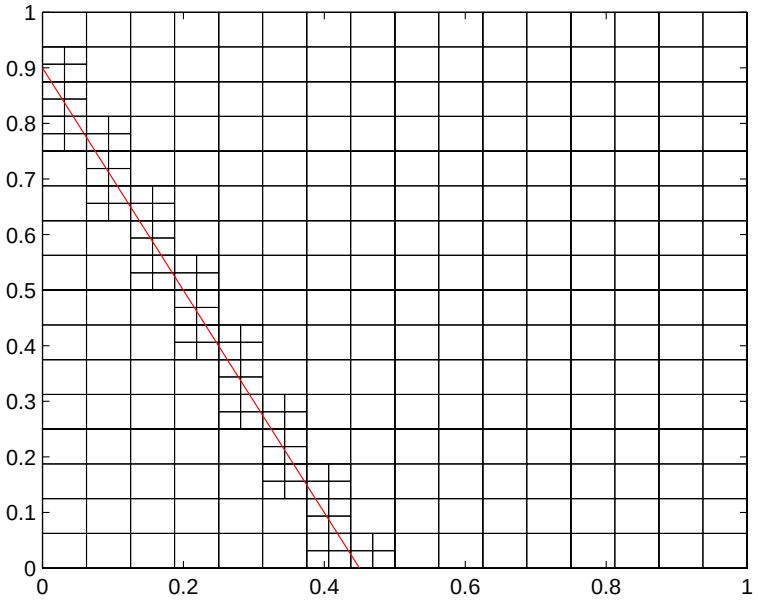

Figura 3.1: *Malha de nível* 5 *com refinamento ao longo da reta* $y = -2x + 0.9$.

É importante ressaltar que as subrotinas de refinamento podem ser aplicadas mais de uma vez na mesma região ou em regiões diferentes, obtendo assim diferentes tipos de malha. A figura 3.2 mostra uma malha refinada ao longo de retas com inclinações diferentes. Já a malha apresentada na figura 3.3, possui refinamento duplo ao longo da reta de inclinação 2.5 e coeficiente linear 1, ou seja, a subrotina `REFINA_RETA(i,a,b)` foi executada duas vezes, com isso é possível obtermos malhas com maior resolução. A malha da figura 3.4 apresenta regiões ainda mais refinadas.

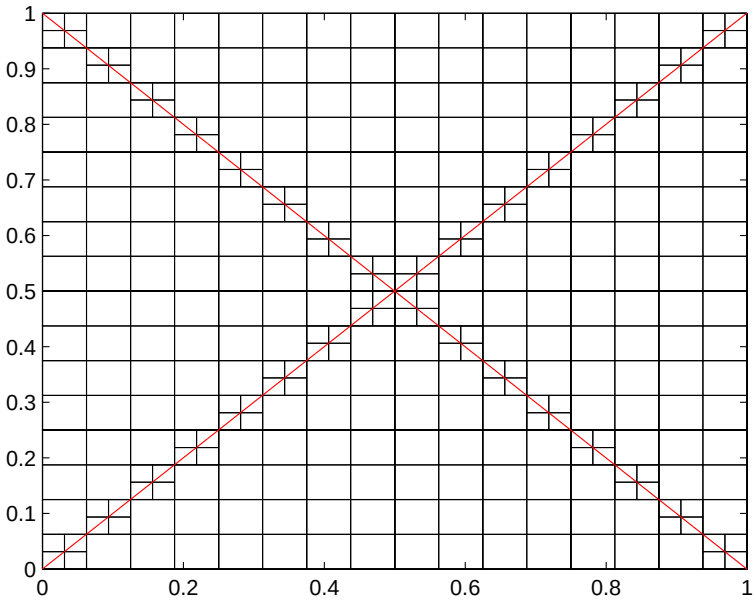

Figura 3.2: *Malha de nível* 5 *com refinamento ao longo das retas* $y = x$ *e* $y = -1x + 1$.

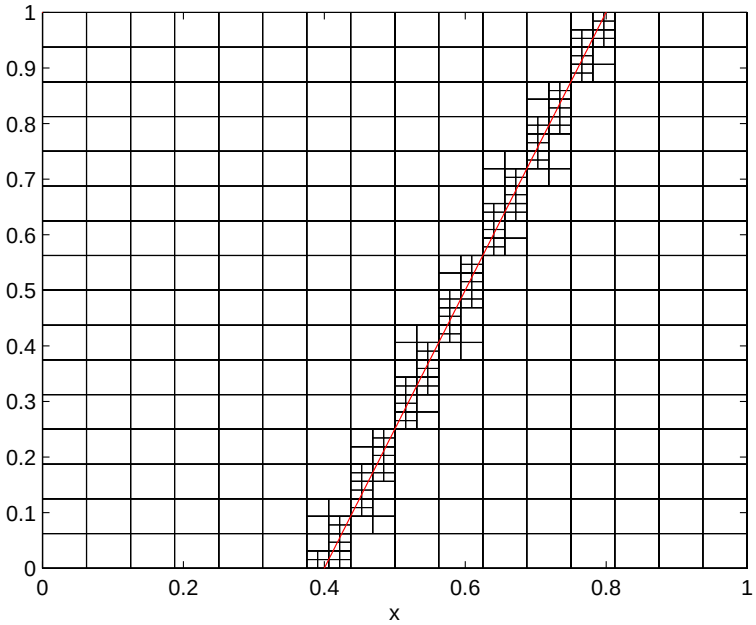

Figura 3.3: *Malha de nível* 5 *com duplo refinamento ao longo da reta* $y = 2.5x + 1$.

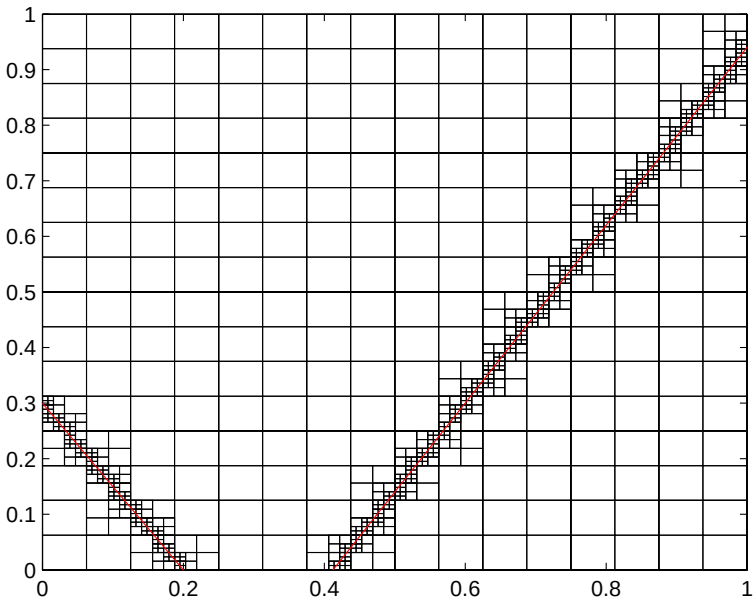

Figura 3.4: *Malha de nível* 5 *com triplo refinamento ao longo das retas* $y = -1.5x + 0.3$ *e* $y = 1.6x - 0.66$.

3.1.2 Refinando a Partir de Equações Polares

Com o auxílio da subrotina que subdivide um nodo é possível gerar malhas com refinamento na forma de círculos, rosáceas, cardióides, entre outros. Para isto basta utilizar as equações apropriadas que geram estas figuras. Para rosáceas, por exemplo, as equações são definidas pelo raio:

$$r = a\cos(n\theta) \text{ e } r = a\sin(n\theta) \tag{3.1}$$

onde $n = 1, 2, 3, ...$ e que possuem

- $2n$ pétalas, se n é par,
- n pétalas, se n é ímpar.

A figura 3.5 mostra uma malha uniforme de nível 7 com refinamento ao longo de uma rosácea de 5 pétalas, já na figura 3.6 temos uma malha uniforme de nível 6 com um refinamento na forma de um cardióide.

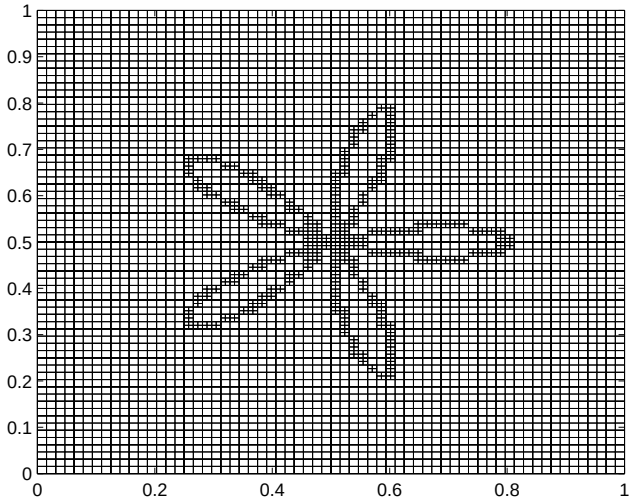

Figura 3.5: *Malha de nível* 7 *com refinamento em forma de uma rosácea de* 5 *pétalas.*

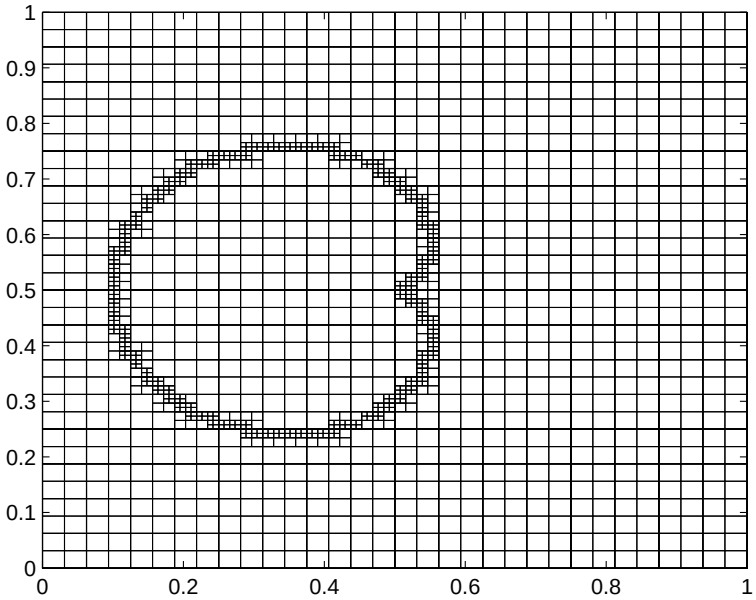

Figura 3.6: *Malha de nível* 6 *com refinamento em forma de um cardióide.*

Se desejarmos regiões bem definidas, basta utilizarmos a subrotina repetidamente até alcançarmos a resolução desejada. Um exemplo disto pode ser visto na figura 3.7 que apresenta uma malha com um refinamento triplo na forma de um limaçon. Nas figuras 3.8 e 3.9 temos exemplos de malhas com refinamentos ao longo de círculos.

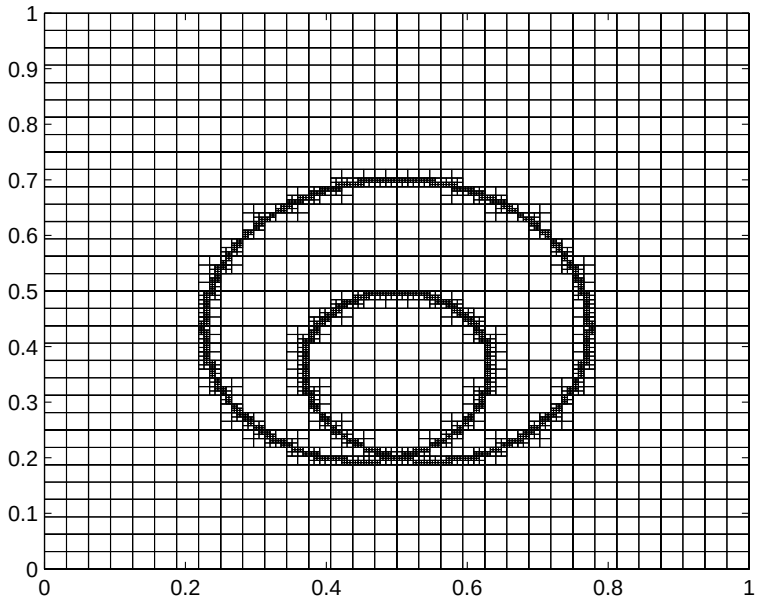

Figura 3.7: *Malha de nível* 6 *com triplo refinamento em forma de um limaçon.*

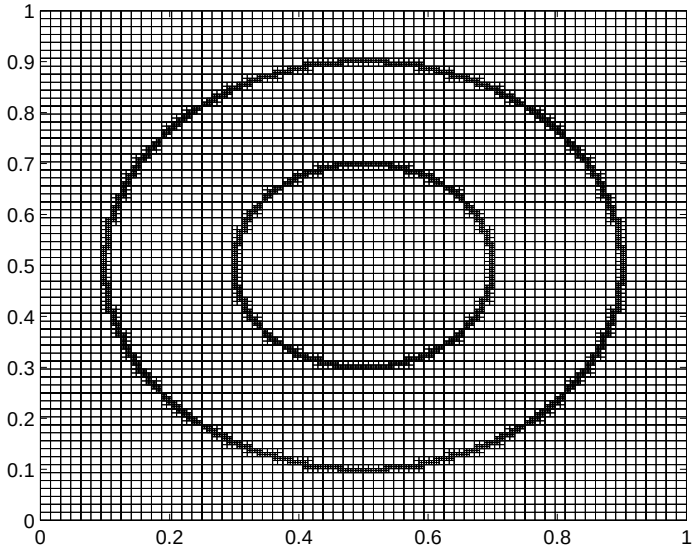

Figura 3.8: *Malha de nível* 7 *com duplo refinamento ao longo dos círculos de centro* $(0.5, 0.5)$ *e raios* 0.2 *e* 0.4.

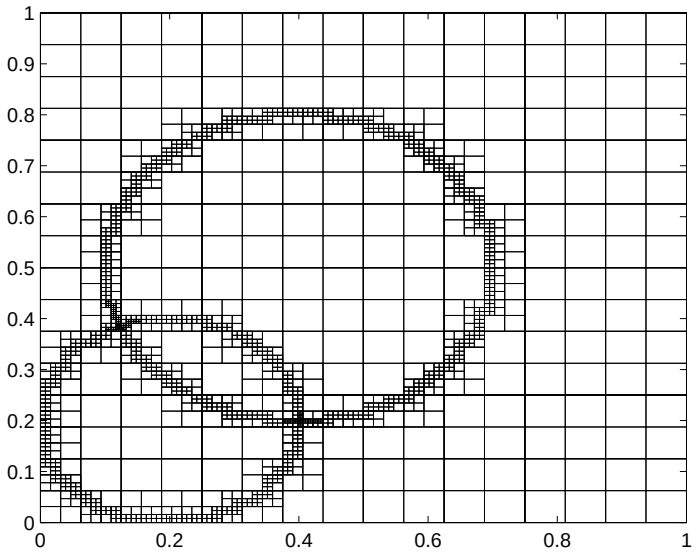

Figura 3.9: *Malha de nível* 5 *com triplo refinamento ao longo dos círculos: raio* 0.2 *e centro* $(0.2; 0.2)$ *e raio* 0.3 *e centro* $(0.4; 0.5)$.

Com os exemplos apresentados até o momento já é possível entendermos e identificarmos o poder de uma malha *quadtree*. Em outras palavras, com o uso das subrotinas apresentadas até aqui podemos gerar malhas com uma boa resolução contendo obstáculos circulares, o que torna o método uma ótima alternativa na geração de malhas.

3.1.3 Refinando uma Região

Para o refinamento de uma determinada região, como um quadrado ou retângulo, basta informarmos os pontos que determinam os extremos da diagonal do quadrilátero que define a região de interesse, denotados por (xi, yi) e (xf, yf) para a subrotina `REFINA_REGIAO(i,xi,xf,yi,yf)` e, então toda a região conforme a figura 3.10 é refinada.

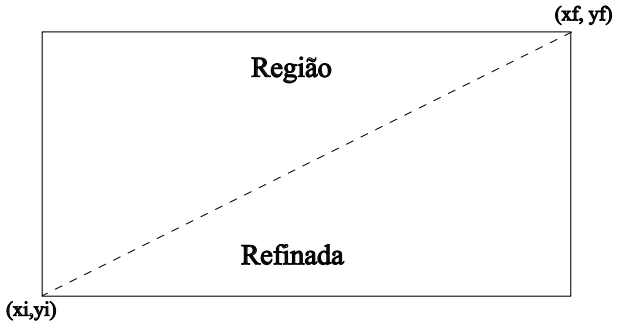

Figura 3.10: *Esquema da delimitação da região a ser refinada.*

Um exemplo da aplicação desta subrotina pode ser vista na figura 3.11 que apresenta uma malha, inicialmente, de nível 6 com refinamento em duas regiões, gerando, assim uma malha com dois níveis de refinamento.

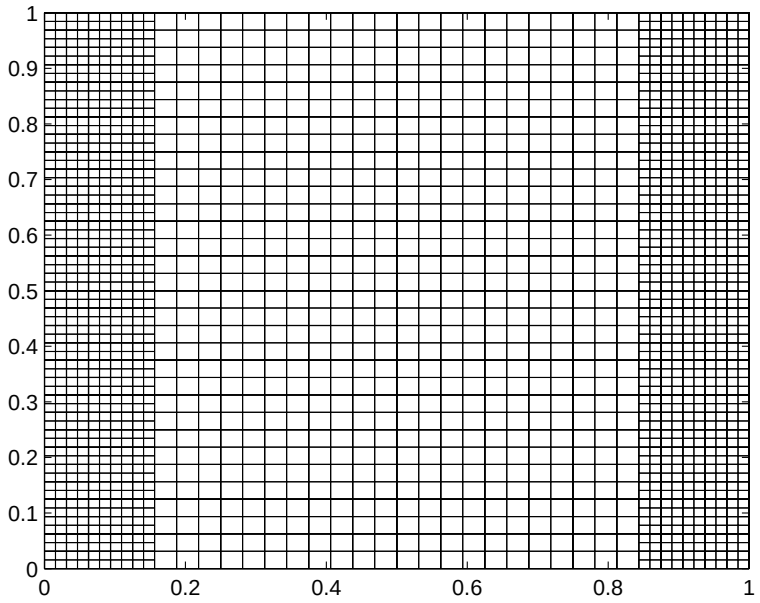

Figura 3.11: *Malha de nível* 6 *com refinamento das regiões com diagonais definidas pelos pontos:* $(0, 0)$, $(0.15, 1)$ *e* $(0.85, 1)$, $(0.15, 1)$, *gerando assim, uma malha com dois níveis de refinamento,* 6 *e* 7.

3.2 *Quadtree* Balanceada

Como foi dito anteriormente no capítulo 2, que tratou da estrutura de dados na árvore, o nível entre nodos adjacentes não pode exceder 2 [14, 15, 29], mais especificamente, o nível máximo entre vizinhos de aresta é 1 e entre vizinhos de vértice é 2. Quando uma malha satisfaz esta condição, dizemos que se trata de uma *quadtree balanceada*. Esta condição proporciona um equilíbrio entre a flexibilidade da grade e a precisão. Por exemplo, na interface entre células de diferentes tamanhos os **vizinhos de aresta** devem ter diferença de até 1 nível, pois isto reduz os possíveis erros de interpolação através das discretizações, enquanto permite que a densidade da grade varie significativamente em pequenas distâncias.

A primeira ideia que surgiu no início da implementação do algoritmo que tem por função balancear a quadtree, foi percorrer a árvore comparando o nível dos vizinhos, mas isto não foi suficiente. Para entendermos melhor, consideremos a malha da figura 3.12 que necessita ser balanceada, pois há nodos adjacentes de nível 3 e 6 o que não pode acontecer.

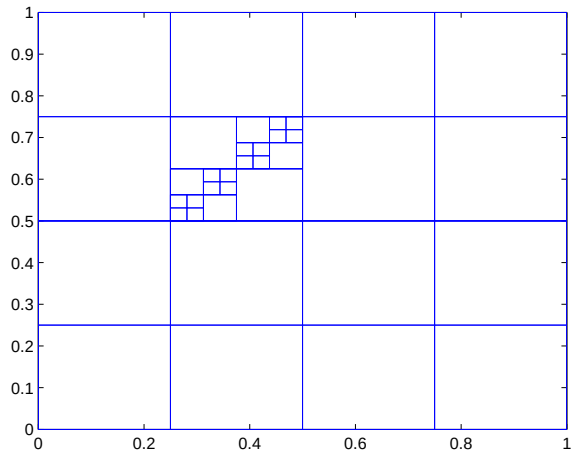

Figura 3.12: *Malha de níveis* 3, 4, 5 *e* 6 *não balanceada.*

A primeira tentativa de balanceamento pode ser verificada na malha da figura 3.13. O que acontece é que os nodos marcados pela letra "A" também deveriam ter sido subdivididos, mas isto não acontece, pois apenas os nodos pais desses quadrantes são verificados pela subrotina de balanceamento. Logo, não é suficiente compararmos apenas os níveis dos vizinhos, mas devemos também verificar através da subrotina de balanceamento os novos nodos que são gerados a partir da necessidade de balanceamento da malha.

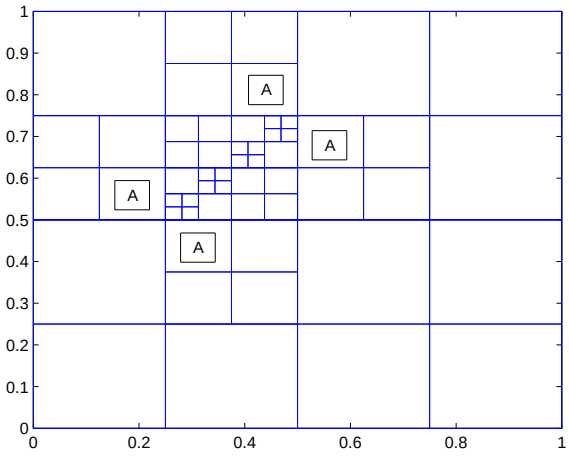

Figura 3.13: *Resultado da primeira tentativa de balanceamento. Os nodos marcados com a letra "A" também deveriam ter sido refinados.*

Portanto, para corrigir este problema devemos incluir a procura dos vizinhos dos novos nodos durante a execução da subrotina e incluí-los na árvore imediatamente após a subdivisão de um quadrante, e então aplicar a subrotina de balanceamento para os quatro nodos mais recentes. De forma resumida apresentamos os passos para esta subrotina.

ALGORITMO: BALANCEAMENTO_QUADTREE

Entrada: NO_i

1 **SE** NO_i possui filhos, **ENTÃO**

2 **PARA** cada $direcao \in \{leste, oeste, norte, sul\}$

3 **SE** o nível do vizinho na *direcao* é maior do que o nível do NO_i em uma unidade, **ENTÃO**

4 Subdivide o No vizinho.

5 Recalcula os vizinhos de cada NO da árvore.

6 Faça o balanceamento dos quatro novos nodos.

E o resultado obtido pode ser verificado na malha da figura 3.14, onde os níveis dos vizinhos de arestas satisfazem a condição de 1 nível de diferença e entre vizinhos de vértice de no máximo 2 níveis.

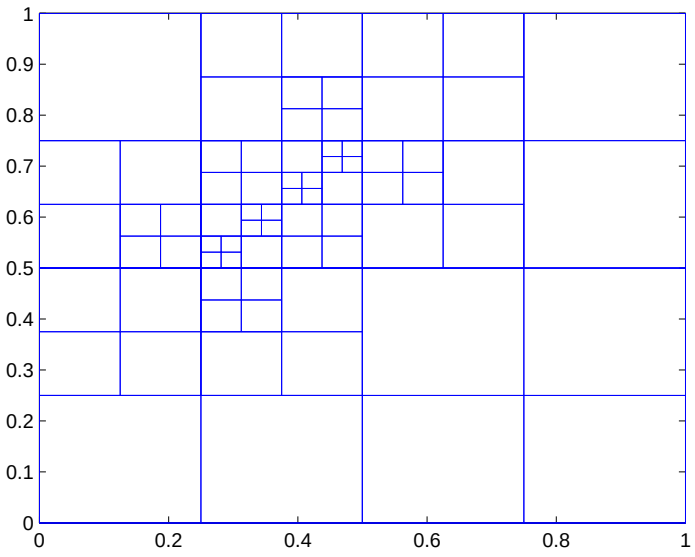

Figura 3.14: *Malha da figura* (3.12) *balanceada.*

Na figura 3.15 apresentamos duas malhas com refinamento ao longo de uma determinada reta, sendo uma na forma balanceada através da subrotina apresentada nesta seção.

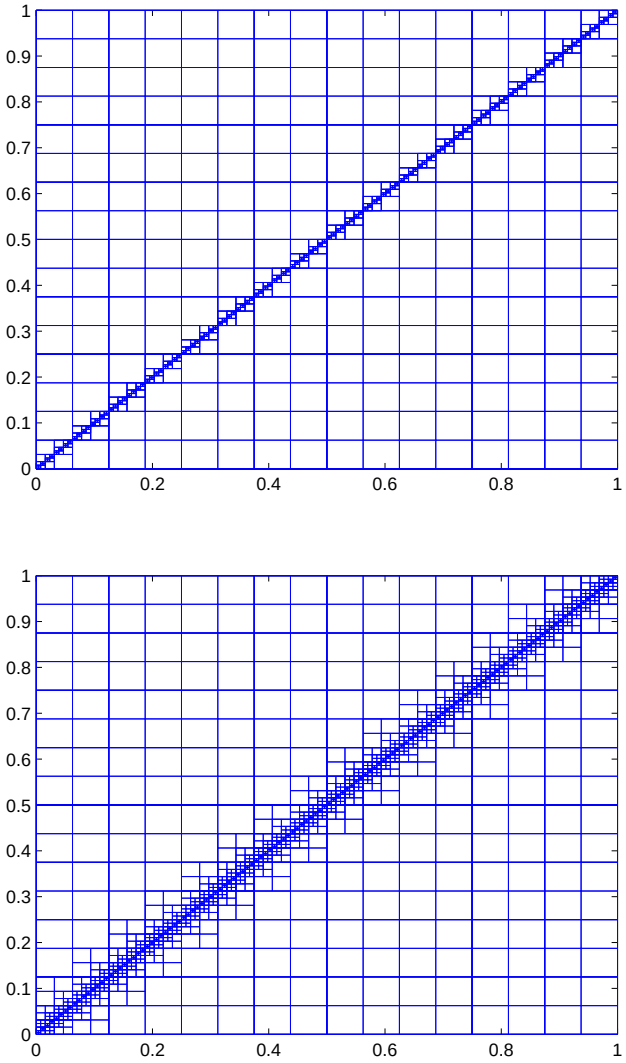

Figura 3.15: *Malha de níveis* 4 *a* 8 *não balanceada (acima) e quadtree balanceada (abaixo).*

De forma semelhante, a malha da figura 3.16 mostra a diferença entre duas malhas com refinamento ao longo de duas retas, sendo que uma das malhas, mais exatamente a malha inferior está satisfazendo o critério de balanceamento.

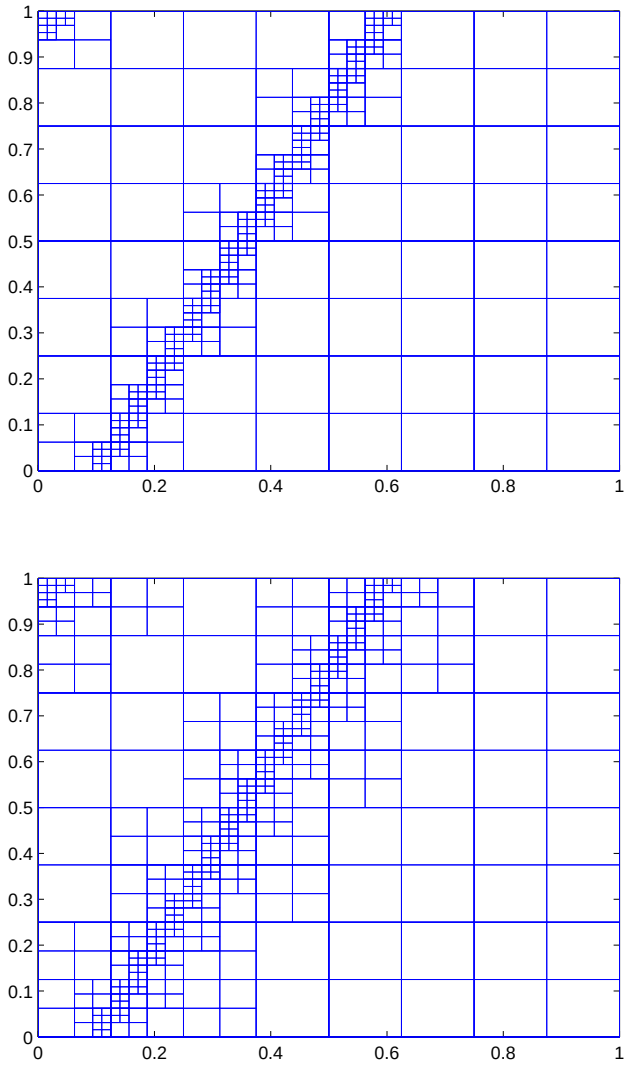

Figura 3.16: *Malha de níveis* 4 *a* 7 *não balanceada (acima) e quadtree balanceada (abaixo).*

Na figura 3.17 temos uma malha com 5 níveis de refinamento numa região retangular próxima a borda da direção norte e também o resultado obtido após o balanceamento da *quadtree*, gerando assim uma malha com os padrões esperados de balanceamento.

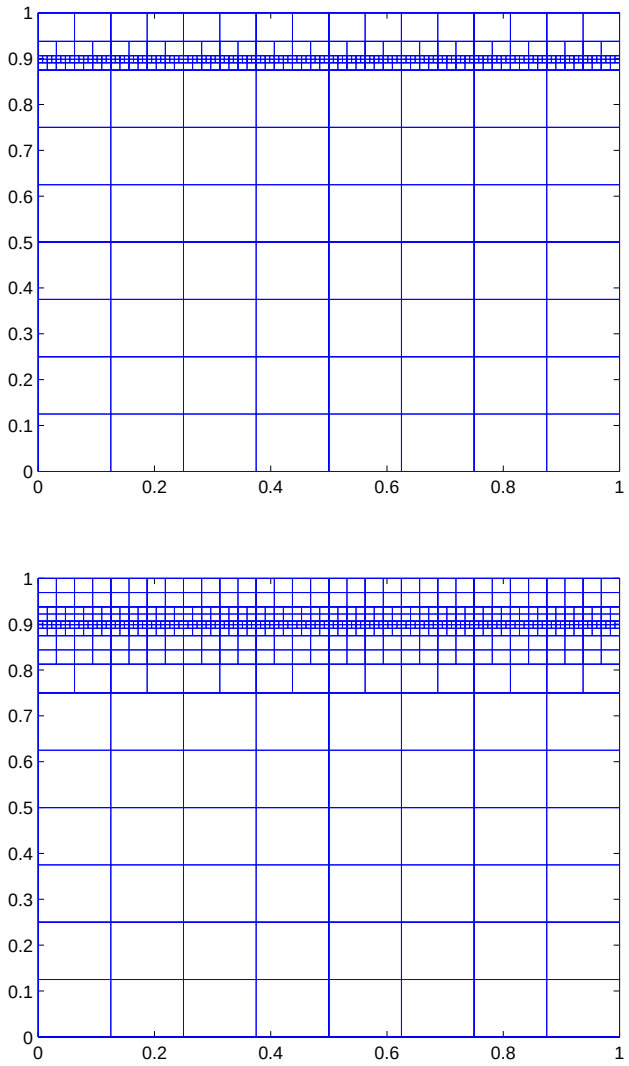

Figura 3.17: *Malha de níveis* 4 *a* 8 *não balanceada (acima) e quadtree balanceada (abaixo).*

Confirmando a eficiência da subrotina que realiza o balanceamento da *quadtree*, temos na figura 3.18 a diferença entre duas malhas com refinamento ao longo de um círculo de raio 0.2 e centro $(0.5, 0.5)$ após a verificação feita através da subrotina `BALANCEAMENTO_QUADTREE`.

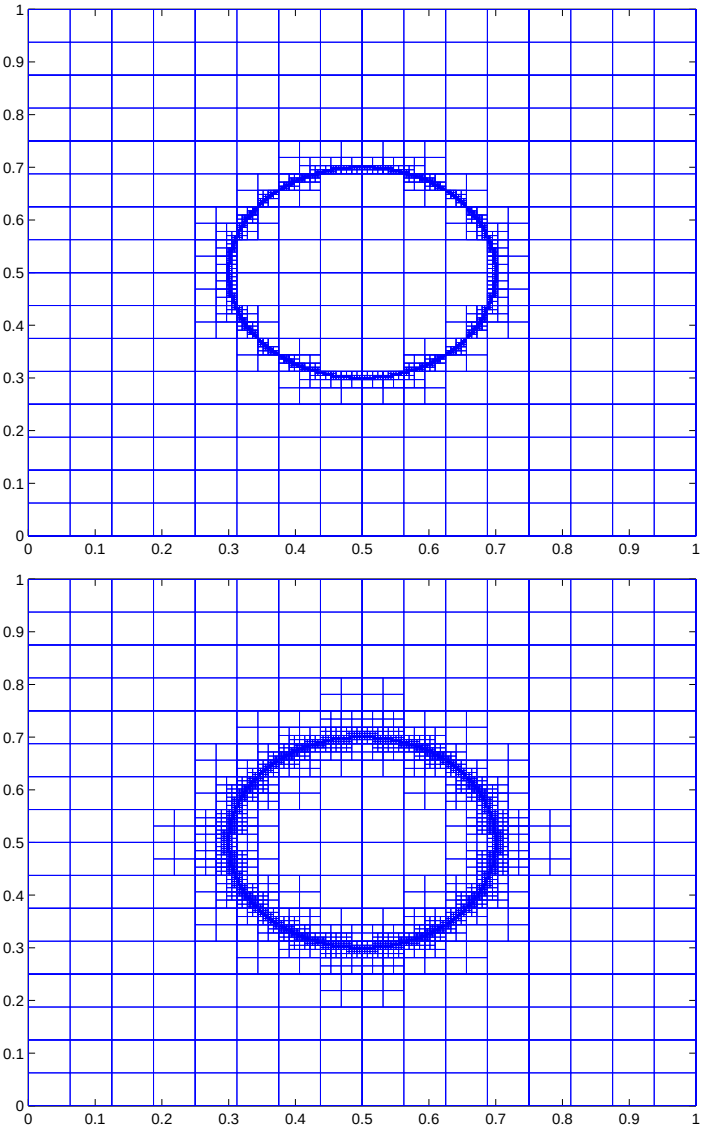

Figura 3.18: *Malha de níveis* 5 *a* 9 *não balanceada (acima) e quadtree balanceada (abaixo).*

Aumentando a complexidade da malha, temos na figura 3.19 uma malha com refinamento na forma de uma rosácea de 4 pétalas fora do critério de balanceamento, e também o resultado obtido quando esta satisfaz o critério de balanceamento.

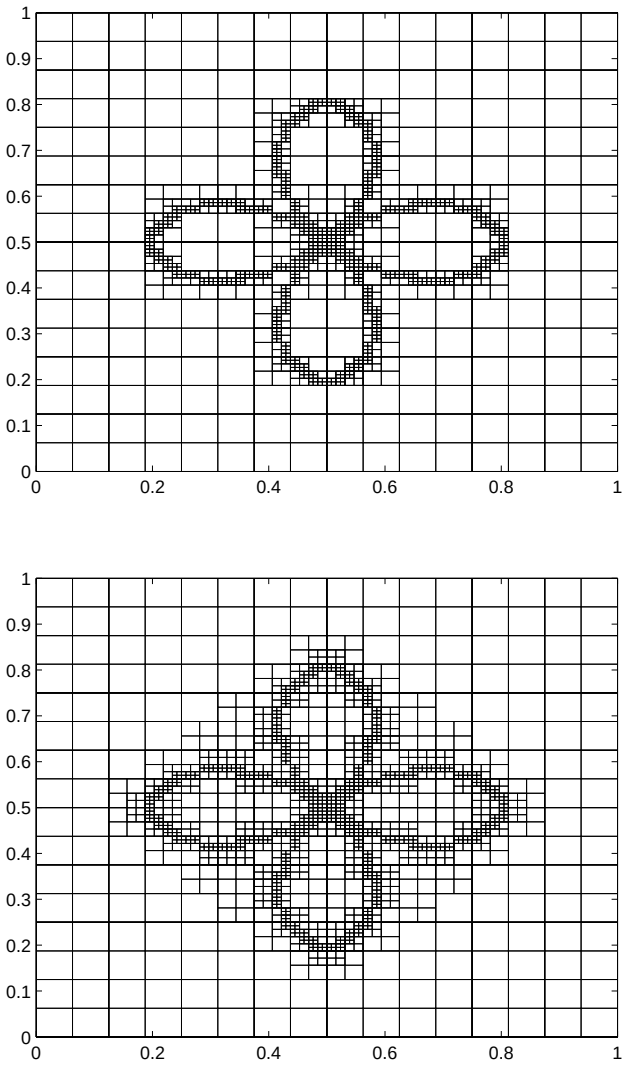

Figura 3.19: *Malha de níveis* 5 *a* 8 *não balanceada (acima) e quadtree balanceada (abaixo).*

E, para finalizar com estes exemplos, na figura 3.20 apresentamos uma *quadtree* balanceada com refinamentos mistos, mais especificamente, refinamentos ao longo de uma rosácea de 8 pétalas contida na interior de um círculo.

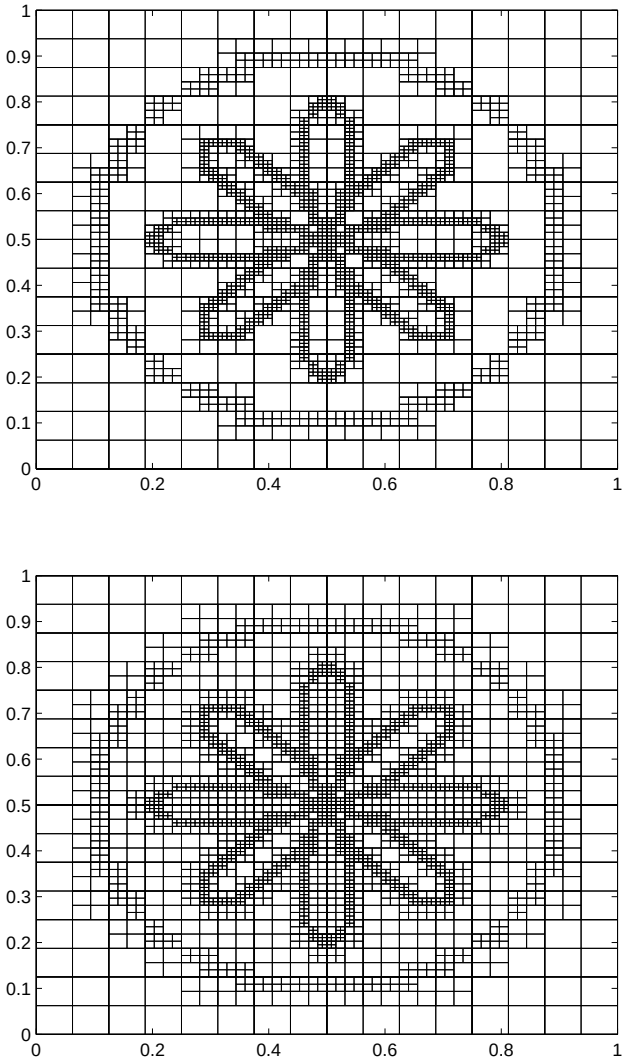

Figura 3.20: *Malha de níveis* 5 *a* 8 *não balanceada (acima) e quadtree balanceada (abaixo).*

4 INTERPOLAÇÃO E DIFERENÇAS FINITAS NA *QUADTREE*

Como estamos trabalhando em malhas não uniformes, faz-se necessário o uso de interpolações dos valores das funções utilizadas numa simulação. Na bibliografia pesquisada encontramos algumas maneiras de fazer isso, como em [34, 14, 27, 1, 13]. Além disso, neste capítulo também trataremos da discretização das derivadas em diferenças finitas para o caso das malhas *quadtree*.

4.1 Interpolação

Trabalhamos com dois tipos de interpolação para o caso de vizinhos em níveis diferentes:

Tipo 1 - Interpolação para um nível inferior - faz-se a média dos valores da função nos nodos filhos e guarda-se no nodo pai.

Tipo 2 - Interpolação para um nível superior - usa-se interpolação linear conforme [34].

Lembramos que os valores das variáveis são guardados no centro de cada quadrante.

4.1.1 Interpolação do Tipo 1

Neste caso pretendemos interpolar valores de vizinhos de um nível superior para um nível inferior conforme esquema representado na figura 4.1. Desta forma os valores que serão usados na interpolação estão num quadrante mais refinado que o seu vizinho e são utilizados para encontrar o valor da função num quadrante de um nível inferior através da média aritmética. Na figura 4.1 podemos observar

que os pontos marcados com (•) representam os valores da função nos nodos filhos, e portanto através da média desses quatro pontos obtemos o ponto representado por (∘) no quadrante de um nível inferior.

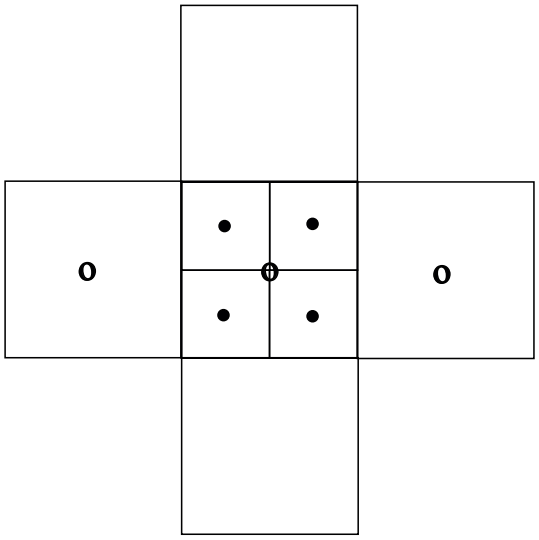

Figura 4.1: *Esquema da interpolação do tipo 1.*

Assim é possível obter a derivada de uma função em diferenças finitas em relação a x, por exemplo, utilizando os pontos indicados por (∘) na figura 4.1, pois estes encontram-se num mesmo nível. Para implementar o algoritmo com esta primeira interpolação são necessárias duas subrotinas: uma que contenha a fórmula da média dos quatro pontos e outra que percorra a árvore comparando os vizinhos e aplicando a média onde for necessária (*subrotina recursiva*).

4.1.2 Interpolação do Tipo 2

Para interpolar os valores das variáveis que estão num nível inferior em comparação a um dos seus vizinhos, utilizando 3 pontos, usamos interpolação linear conforme [34]. A figura 4.2 mostra que os pontos $PE1$ e $PE2$ são interpolados

utilizando os pontos $E1$, $E2$ e P. A vantagem desta interpolação é que são usados os valores dos vizinhos mais próximos aos pontos dos quais deseja-se interpolar.

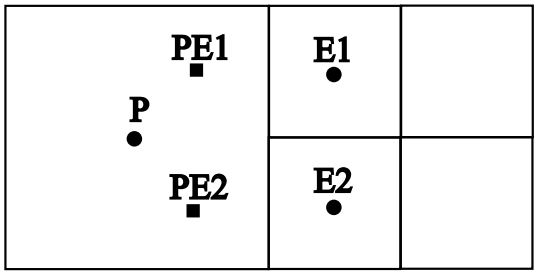

Figura 4.2: *Esquema da interpolação do tipo 2.*

A interpolação linear consiste em determinar a equação da reta que passa por dois pontos conhecidos. Em outras palavras, se dois pontos de coordenadas $(x_0, f(x_0))$ e $(x_1, f(x_1))$ são conhecidos, então para $x_0 < x < x_1$ temos que $f(x)$ ao longo da reta é dado pela equação

$$\frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{4.1}$$

ou seja,

$$f(x) = f(x_0) + (x - x_0)\frac{f(x_1) - f(x_0)}{x_1 - x_0}. \tag{4.2}$$

Passo 1: consideremos inicialmente um ponto K como mostra a figura 4.3 de coordenadas $(x_k, f(x_k))$ tal que

$$f(x_k) = \frac{f(x_{E1})}{2} + \frac{f(x_{E2})}{2}, \tag{4.3}$$

onde as coordenadas de E_1 e E_2 são, respectivamente, $(x_{E1}, f(x_{E1}))$ e $(x_{E2}, f(x_{E2}))$.

Passo 2: considere um ponto $(x_{mf}, f(x_{mf}))$ localizado entre os pontos P e K de tal forma que esteja à $\frac{2}{3}$ do ponto P e à $\frac{1}{3}$ do ponto K, veja a figura 4.3. Sejam $(x_0, f(x_0))$, $(x_1, f(x_1))$ e $(x, f(x))$ dados por $(0, f(x_P))$, $(1, f(x_K))$ e $(\frac{2}{3}, f(x_{mf}))$, respectivamente. Portanto pela relação 4.2 o valor de $f(x_{mf})$ é dado

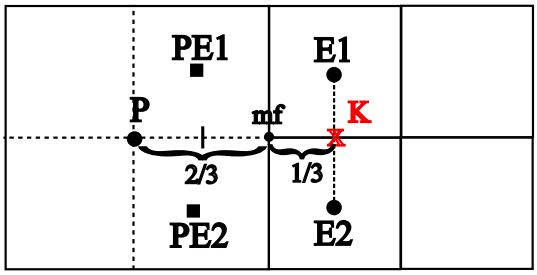

Figura 4.3: *Esquema de pontos auxiliares para a obtenção das fórmulas de interpolação.*

por

$$f(x_{mf}) = \frac{2}{3}f(x_K) + \frac{1}{3}f(x_P) \tag{4.4}$$

e substituindo (4.3) em (4.4) obtemos

$$f(x_{mf}) = \frac{1}{3}f(x_{E1}) + \frac{1}{3}f(x_{E2}) + \frac{1}{3}f(x_P). \tag{4.5}$$

Observe a figura 4.4, onde é possível observarmos que ligando os pontos $PE1$ e $E2$ de coordenadas $(x_{PE1}, f(x_{PE1}))$ e $(x_{E2}, f(x_{E2}))$ através de uma reta, esta passa por m_f, sendo este o ponto médio entre $PE1$ e $E2$.

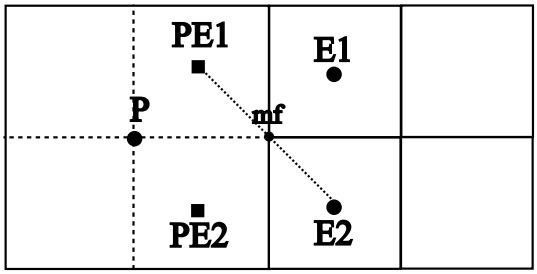

Figura 4.4: *Representação da uma reta auxiliar para a obtenção das fórmulas de interpolação.*

Assim, podemos obter outra relação para o valor de $f(x_{mf})$ que é dada por

$$f(x_{mf}) = \frac{1}{2}f(x_{PE1}) + \frac{1}{2}f(x_{E2}). \tag{4.6}$$

Substituindo (4.5) em (4.6) e fazendo as simplificações necessárias, obtemos a relação de interpolação do ponto $PE1$ que depende dos pontos P, $E1$ e $E2$ como desejávamos, e é dada por

$$f(x_{PE1}) = \frac{2}{3}f(x_P) + \frac{2}{3}f(x_{E1}) - \frac{1}{3}f(x_{E2}). \tag{4.7}$$

Através de um procedimento análogo, obtemos o valor de $PE2$

$$f(x_{PE2}) = \frac{2}{3}f(x_P) + \frac{2}{3}f(x_{E2}) - \frac{1}{3}f(x_{E1}). \tag{4.8}$$

Os pontos $PE1$ e $PE2$ chamaremos de *pontos fictícios*, pois eles serão usados apenas como pontos auxiliares na discretização das derivadas. Como vimos, o quadrante que contém o ponto P no centro pode ser dividido de tal forma que obtemos 4 quadrantes *fictícios* de mesmo tamanho que os quadrantes com os pontos $E1$ e $E2$. Desta forma os quadrantes que contêm os pontos $PE1$ e $PE2$ são do mesmo tamanho que seus vizinhos na direção leste.

Como cada quadrante possui 4 vizinhos de lado podemos ter 8 possíveis pontos fictícios. Para facilitar a discretização dos operadores na direção x e y utilizamos pontos fictícios nas duas direções, dependendo do tipo de vizinho do quadrante como mostra o esquema 4.5.

Na direção horizontal (vizinhos leste e oeste) temos 4 possíveis pontos fictícios que denominaremos por: $pe1x$, $pe2x$, $pe3x$ e $pe4x$. Analogamente, na direção vertical (vizinhos norte e sul) temos $pe1y$, $pe2y$, $pe3y$ e $pe4y$ como os possíveis pontos fictícios. Para estas interpolações, primeiramente foram criadas duas subrotinas, uma para as interpolações na direção x e outra para a direção y. A idéia é verificar os três possíveis casos.

Por exemplo, na direção x temos os vizinhos leste e oeste, sendo assim consideramos três possibilidades:

1. os dois vizinhos estão no mesmo nível;

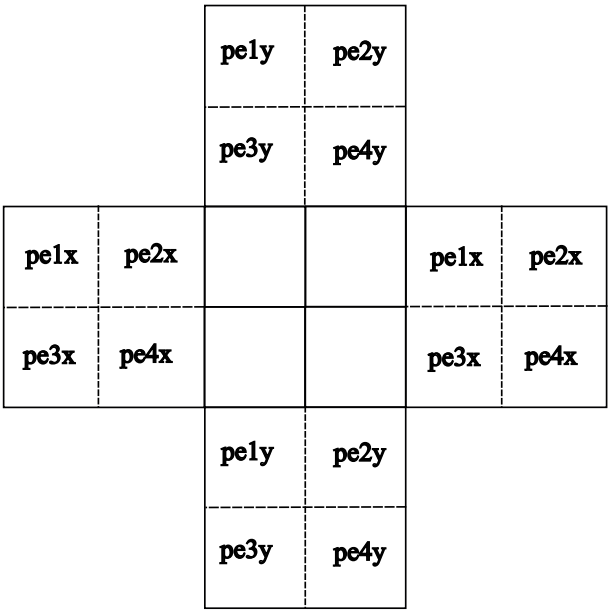

Figura 4.5: *Esquema da distribuição dos pontos fictícios nas direções x e y.*

2. apenas o vizinho leste está refinado;

3. apenas o vizinho oeste está refinado.

Considerando estes três casos, apresentamos o resumo da implementação dos algoritmos de interpolação.

ALGORITMO: INTERPOLA_HORIZONTAL

Entrada: $pai = NO$

1 **SE** os vizinhos leste e oeste estão num nível maior **ENTÃO**

2 Calcule $pe1x$, $pe2x$, $pe3x$ e $pe4x$.

3 **SENÃO**, **SE** o vizinho leste está num nível maior**ENTÃO**

4 Calcule $pe2x$ e $pe4x$.

5 Defina $pe1x = 0$ e $pe3x = 0$

6 **SENÃO**, **SE** o vizinho oeste está num nível maior**ENTÃO**

7 Calcule $pe1x$ e $pe3x$.

8 Defina $pe2x = 0$ e $pe4x = 0$

ALGORITMO: INTERPOLA_VERTICAL

Entrada: $pai = NO$

1 **SE** os vizinhos norte e sul estão num nível maior **ENTÃO**

2 Calcule $pe1y$, $pe2y$, $pe3y$ e $pe4y$.

3 **SENÃO**, **SE** o vizinho norte está num nível maior**ENTÃO**

4 Calcule $pe1y$ e $pe2y$.

5 Defina $pe3y = 0$ e $pe4y = 0$

6 **SENÃO**, **SE** o vizinho sul está num nível maior**ENTÃO**

7 Calcule $pe3y$ e $pe4y$.

8 Defina $pe1y = 0$ e $pe2y = 0$

Há a necessidade de que os nodos vizinhos sejam comparados com relação a diferença de níveis. Assim a subrotina `INTERPOLA_ARVORE` executa esta operação aplicando as interpolações verticais e horizontais quando forem necessárias, e ainda guarda os valores interpolados na árvore ao longo das simulações. Quando os pontos fictícios não são necessários, atribui-se 0.

ALGORITMO: INTERPOLA_ARVORE

Entrada: $pai = NO$

1 **SE** um dos vizinhos está num nível maior **ENTÃO**

2 Chame $INTERPOLA\_HORIZONTAL$ e $INTERPOLA\_VERTICAL$.

3 **SENÃO**

4 Defina os pontos fictícios como 0.

4.2 Diferenças Finitas

O grande diferencial da discretização das derivadas na malha *quadtree* é que necessitamos conhecer os vizinhos de cada quadrante e utilizar as interpolações adequadas. Para tornar o algoritmo mais rápido, os vizinhos de cada quadrante são armazenados na árvore logo após a geração da malha. Esta afirmação é verificada através de um teste no capítulo 5.

Da maneira como tratamos as interpolações, não há maiores dificuldades para discretizar as derivadas, pois estaremos sempre utilizando quadrantes de mesmo tamanho. Como exemplo, consideremos a malha mostrada na figura 4.6, onde deseja-se calcular as derivadas de primeira e segunda ordem nos pontos • e ∘, ou seja, nos nodos folhas da árvore. É importante ressaltar que, para que tenhamos os valores das derivadas nos nodos folhas é necessário percorrer toda a árvore. Portanto as derivadas são calculadas em todos os nodos de todos os níveis.

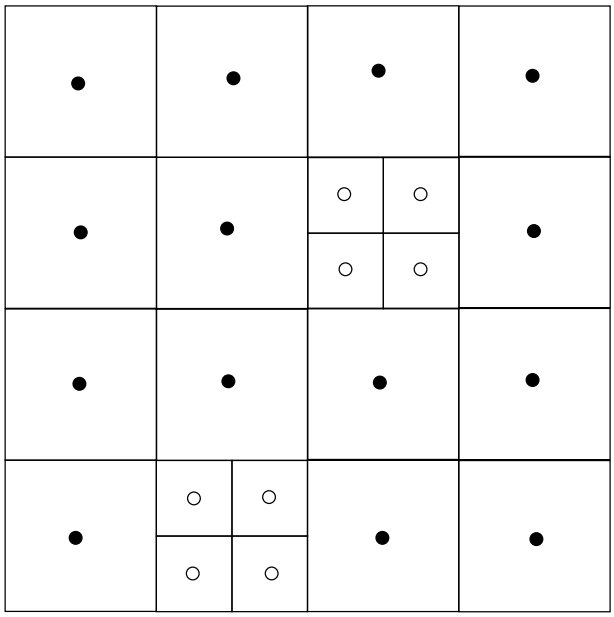

Figura 4.6: *Malha quadtree com representação dos pontos centrais de cada quadrante.*

Primeiramente vamos considerar os pontos •. Note que os pontos marcados pelas letras A, B, C, D, E, F e G na figura 4.7, têm pelos menos um dos vizinhos num nível diferente. Por exemplo, o ponto G possui vizinhos na direção oeste que estão num nível superior. Logo para o cálculo das derivadas em alguma das direções x ou y nesses pontos, faz-se necessário utilizar a interpolação do tipo 1. Veja na figura 4.7 a representação dos pontos interpolados a partir dos pontos representados por ∘ na figura 4.6.

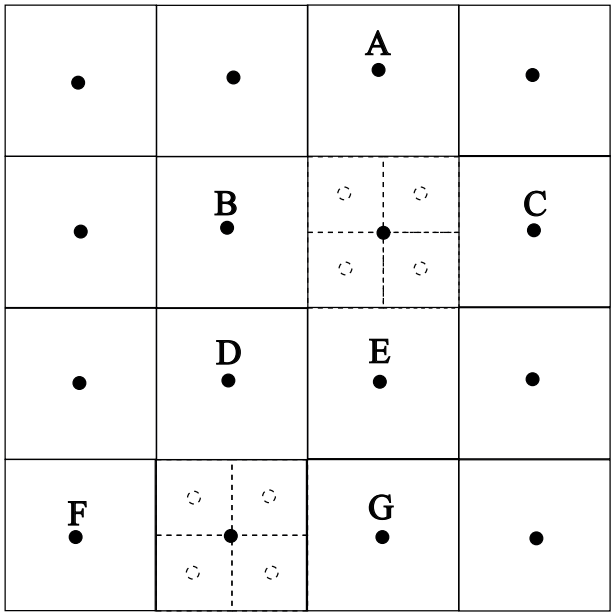

Figura 4.7: *Malha quadtree com representação dos pontos interpolados.*

Desta forma, em qualquer ponto P (•) da malha da figura 4.7, as derivadas de primeira e segunda ordem de uma função f são dadas por:

$$\frac{\partial f_P}{\partial x} \approx \frac{f_l - f_o}{2dx} \tag{4.9}$$

$$\frac{\partial f_P}{\partial y} \approx \frac{f_n - f_s}{2dy} \tag{4.10}$$

$$\frac{\partial^2 f_P}{\partial x^2} \approx \frac{f_l - 2f_P + f_o}{dx^2} \tag{4.11}$$

$$\frac{\partial^2 f_P}{\partial y^2} \approx \frac{f_n - 2f_P + f_s}{dy^2} \tag{4.12}$$

onde

f_P é o valor de f no quadrante que contém P.

f_l é o valor de f no quadrante vizinho ao que contém P na direção leste.

f_o é o valor de f no quadrante vizinho ao que contém P na direção oeste.

f_n é o valor de f no quadrante vizinho ao que contém P na direção norte.

f_s é o valor de f no quadrante vizinho ao que contém P na direção sul.

$\frac{\partial f_P}{\partial x}$ é a aproximação da derivada de primeira ordem de f em P na direção x.

$\frac{\partial f_P}{\partial y}$ é a aproximação da derivada de primeira ordem de f em P na direção y.

$\frac{\partial^2 f_P}{\partial x^2}$ é a aproximação da derivada segunda de f em P na direção x.

$\frac{\partial^2 f_P}{\partial y^2}$ é a aproximação da derivada segunda de f em P na direção y.

Como nosso objetivo é calcular as derivadas nos nodos folhas, ainda resta obtê-las nos pontos ∘ da malha da figura 4.6. Neste caso é utilizada a interpolação do tipo 2 e os pontos fictícios são gerados como mostra a figura 4.8.

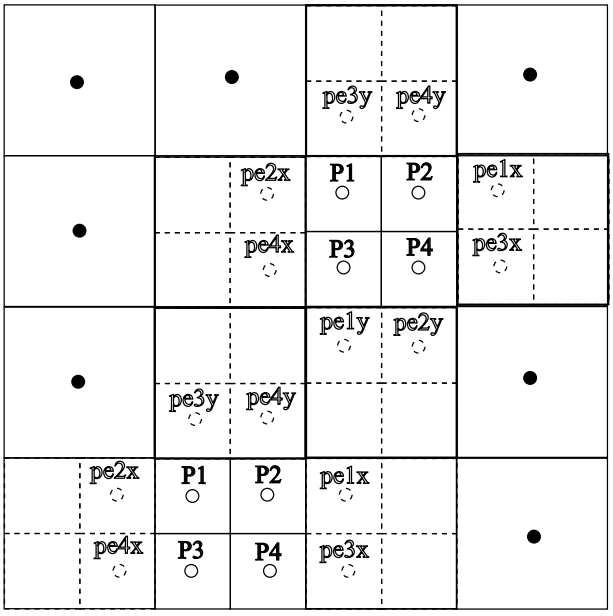

Figura 4.8: *Malha quadtree com representação dos pontos fictícios.*

Assim, a discretização das derivadas nos pontos $P1$, $P2$, $P3$ e $P4$ utilizando os pontos fictícios é dada pelas aproximações:

$$\frac{\partial f_{P1}}{\partial x} \approx \frac{f_l - f_{pe2x}}{2dx} \tag{4.13}$$
$$\frac{\partial f_{P2}}{\partial x} \approx \frac{f_{pe1x} - f_o}{2dx} \tag{4.14}$$
$$\frac{\partial f_{P3}}{\partial x} \approx \frac{f_l - f_{pe4x}}{2dx} \tag{4.15}$$
$$\frac{\partial f_{P4}}{\partial x} \approx \frac{f_{pe3x} - f_o}{2dx} \tag{4.16}$$
$$\frac{\partial f_{P1}}{\partial y} \approx \frac{f_{pe3y} - f_s}{2dy} \tag{4.17}$$
$$\frac{\partial f_{P2}}{\partial y} \approx \frac{f_{pe4y} - f_s}{2dy} \tag{4.18}$$
$$\frac{\partial f_{P3}}{\partial y} \approx \frac{f_n - f_{pe1y}}{2dy} \tag{4.19}$$
$$\frac{\partial f_{P4}}{\partial y} \approx \frac{f_n - f_{pe2y}}{2dy} \tag{4.20}$$

$$\frac{\partial^2 f_{P1}}{\partial x^2} \approx \frac{f_l - 2f_{P1} + f_{pe2x}}{dx^2} \tag{4.21}$$
$$\frac{\partial^2 f_{P2}}{\partial x^2} \approx \frac{f_{pe1x} - 2f_{P2} + f_o}{dx^2} \tag{4.22}$$
$$\frac{\partial^2 f_{P3}}{\partial x^2} \approx \frac{f_l - 2f_{P3} + f_{pe4x}}{dx^2} \tag{4.23}$$
$$\frac{\partial^2 f_{P4}}{\partial x^2} \approx \frac{f_{pe3x} - 2f_{P4} + f_o}{dx^2} \tag{4.24}$$
$$\frac{\partial^2 f_{P1}}{\partial y^2} \approx \frac{f_{pe3y} - 2f_{P1} + f_s}{dy^2} \tag{4.25}$$
$$\frac{\partial^2 f_{P2}}{\partial y^2} \approx \frac{f_{pe4y} - 2f_{P2} + f_s}{dy^2} \tag{4.26}$$
$$\frac{\partial^2 f_{P3}}{\partial y^2} \approx \frac{f_n - 2f_{P3} + f_{pe1y}}{dy^2} \tag{4.27}$$
$$\frac{\partial^2 f_{P4}}{\partial y^2} \approx \frac{f_n - 2f_{P4} + f_{pe2y}}{dy^2} \tag{4.28}$$

onde

f_{Pi} é o valor de f no quadrante que contém Pi, com $i = 1, 2, 3$ e 4.

f_{peix} é o valor de f no quadrante que contém o ponto $peix$, com $i = 1, 2, 3$ e 4 .

f_{peiy} é o valor de f no quadrante que contém o ponto $peiy$, com $i = 1, 2, 3$ e 4 .

$\frac{\partial f_{Pi}}{\partial x}$ é a aproximação da derivada de primeira ordem de f em Pi na direção x, com $i = 1, 2, 3$ e 4.

$\frac{\partial f_{Pi}}{\partial y}$ é a aproximação da derivada de primeira ordem de f em Pi na direção y, com $i = 1, 2, 3$ e 4.

$\frac{\partial^2 f_{Pi}}{\partial x^2}$ é a aproximação da derivada segunda de f em Pi na direção x, com $i = 1, 2, 3$ e 4.

$\frac{\partial^2 f_{Pi}}{\partial y^2}$ é a aproximação da derivada segunda de f em Pi na direção y, com $i = 1, 2, 3$ e 4.

Os passos básicos para a implementação das derivadas nas direções x e y são os seguintes:

ALGORITMO: DERIVADAS_DIRECAOX

Entrada: NO_i

1 **SE** o nível de NO_i é maior que o nível do seu vizinho leste **ENTÃO**
2 Use as fórmulas adequadas.
3 Defina as derivadas nos nodos da fronteira como 0.
4 **SENÃO, SE** o nível de NO_i é maior que o do seu vizinho oeste **ENTÃO**
5 Use as fórmulas adequadas.
6 Defina as derivadas nos nodos da fronteira como 0.
7 **SENÃO, SE** o nível de NO_i é igual ao dos seus vizinhos **ENTÃO**
8 Use as fórmulas adequadas.
8 Defina as derivadas nos nodos da fronteira como 0.

ALGORITMO: DERIVADAS_DIRECAOY

Entrada: NO_i

1 **SE** o nível de NO_i é maior que o nível do seu vizinho norte **ENTÃO**
2 Use as fórmulas adequadas.
3 Defina as derivadas nos nodos da fronteira como 0.

4 **SENÃO, SE** o nível de NO_i é maior que o do seu vizinho sul **ENTÃO**

5 Use as fórmulas adequadas.

6 Defina as derivadas nos nodos da fronteira como 0.

7 **SENÃO, SE** o nível de NO_i é igual ao dos seus vizinhos **ENTÃO**

8 Use as fórmulas adequadas.

8 Defina as derivadas nos nodos da fronteira como 0.

Nos passos 1, 4 e 7 dos algoritmos apresentados acima, é necessário garantir que os vizinhos em alguma das direções leste, oeste, norte ou sul sejam diferentes de zero, pois no caso em que algum dos vizinhos é 0 temos nodos que estão na fronteira do domínio e por convenção as derivadas nestes pontos devem ser especificadas por 0.

5 TESTES E RESULTADOS COMPUTACIONAIS

Após o desenvolvimento da malha *quadtree* torna-se necessário testarmos sua eficiência. Portanto neste capítulo estudaremos alguns aspectos importantes, como a ordem do método, utilizando a equação do calor. Também faremos algumas comparações com uma malha cartesiana em relação ao tempo de execução. Em seguida apresentaremos alguns resultados obtidos na simulação do problema da cavidade quadrada e de um duto reto. Para tais testes, inicialmente, definiremos de forma direta as equações dos problemas simulados: condução de calor e Navier-Stokes. Para um estudo mais detalhado dos fenômenos e dos termos das equações indicamos [28, 6, 9, 4].

5.1 Equação do Calor

Apresentaremos aqui nesta seção a metodologia adotada para a simulação da condução de calor.

5.1.1 Modelo Matemático

A equação do calor, também conhecida como equação de difusão, descreve a densidade de u ao longo do tempo de alguma quantidade como calor, concentração química, etc. A equação homogênea é dada por

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{5.1}$$

e a equação do calor não homogênea por

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f. \tag{5.2}$$

Temos $t > 0$, $u \in \mathbb{R}^2$. Sendo Ω como a região $0 \leq x, y \leq 1$ e $\partial\Omega$ como o contorno dessa região. Na realização dos testes foram consideradas 6 condições de contorno com suas respectivas condições iniciais. São elas:

Caso 1

$$u(x,0,t) = 0, \quad u(x,1,t) = 1, \qquad \text{para } 0 < x < 1, t > 0 \tag{5.3}$$

$$u(0,y,t) = 0, \quad u(1,y,t) = 0, \qquad \text{para } 0 < y < 1, t > 0 \tag{5.4}$$

$$u(x,y,0) = 0 \qquad \text{para } 0 < x < 1, 0 < y < 1 \tag{5.5}$$

Caso 2

$$u(x,y,t) = x + y, \quad \text{para } (x,y) \in \partial\Omega, t > 0 \tag{5.6}$$

$$u(x,y,0) = 0, \qquad \text{para } (x,y) \in \Omega \tag{5.7}$$

Caso 3

$$u(x,y,t) = x^2 + y^2, \quad \text{para } (x,y) \in \partial\Omega, t > 0 \tag{5.8}$$

$$u(x,y,0) = x + y, \qquad \text{para } (x,y) \in \Omega \tag{5.9}$$

Caso 4

$$u(x,y,t) = x^3 + y^3, \quad \text{para } (x,y) \in \partial\Omega, t > 0 \tag{5.10}$$

$$u(x,y,0) = x^2 + y^2, \quad \text{para } (x,y) \in \Omega \tag{5.11}$$

Caso 5

$$u(x,y,t) = x^4 + y^4, \quad \text{para } (x,y) \in \partial\Omega, t > 0 \tag{5.12}$$

$$u(x,y,0) = x^3 + y^3, \quad \text{para } (x,y) \in \Omega \tag{5.13}$$

Caso 6

$$u(x,y,t) = 0, \qquad \text{para } (x,y) \in \partial\Omega, t > 0 \tag{5.14}$$

$$u(x,y,0) = 0, \qquad \text{para } (x,y) \in \Omega \tag{5.15}$$

Com condições interiores: se $(x - x_c)^2 + (y - y_c)^2 \leq r$ então $u(x,y) = 1$, onde (x_c, y_c) são das coordenadas do centro de um círculo de raio r.

5.1.2 Algoritmo

O *método de Euler* que emprega diferenças de primeira ordem é utilizado para a discretização da derivada temporal e é representado por

$$\left[\frac{\partial u}{\partial t}\right]_{i,j}^{n+1} = \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\partial t}. \tag{5.16}$$

Aplicando 5.16 em 5.2 obtemos

$$u_{i,j}^{n+1} = \partial t(\Delta u + f) + u_{i,j}^{n}. \tag{5.17}$$

A primeira subrotina necessária contém a equação a ser iterada e as condições de contorno e é resumida pelos passos apresentados a seguir.

ALGORITMO: EQ_CALOR

Entrada: $pai = NO$

1 Para cada no folha da árvore
2 Calcule u usando 5.17
3 Use as condições de contorno para u.
4 Calcule a norma residual de u.
5 Atualiza o valor de u^n.

A próxima subrotina gerencia a `EQ_CALOR` e chama outras subrotinas necessárias para esta simulação.

ALGORITMO: ITERA_CALOR

Entrada: $pai = NO$, $kmax$, tol

1 $t = 0$, $res = 0$
2 **ENQUANTO** $k < kmax$ e $||res|| > tol$
3 Chame $INTERPOLA\_ARVORE$.
4 Chame $INTERPOLA\_MEDIA$.

5 Chame $DERIVADAxx$.

6 Chame $DERIVADAyy$.

7 Chame $LAPLACIANO$.

8 Chame EQ_CALOR.

9 Calcule o novo res.

10 $k = k + 1$.

As subrotinas `DERIVADAxx`, `DERIVADAyy` calculam as derivadas de segunda ordem de u conforme discretizações apresentadas no capítulo anterior. Para calcular Δu é utilizada a subrotina `LAPLACIANO`.

5.2 Equação de Navier-Stokes

Nesta seção são apresentadas as equações de Navier-Stokes para a simulação de escoamento de fluidos e também o algoritmo **PRIME** utilizado para encontrar a solução do problema.

5.2.1 Modelo Matemático

Em coordenadas cartesianas bidimensionais, para fluxos laminares de fluidos incompressíveis e isotérmicos, as equações de Navier-Stokes são representadas pela equação da continuidade que reflete o princípio físico da conservação de massa, e pelas equações de momento nas direções x e y, que representam a aplicação da segunda lei de Newton ao fluido, que são respectivamente

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{5.18}$$

$$\frac{\partial u}{\partial t} + \frac{\partial (uu)}{\partial x} + \frac{\partial (uv)}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{5.19}$$

$$\frac{\partial v}{\partial t} + \frac{\partial (uv)}{\partial x} + \frac{\partial (vv)}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{5.20}$$

onde u e v são as velocidades nas direções x e y, respectivamente e Re é um número adimensional conhecido como *número de Reynolds*, que indica razão entre as forças inerciais e as forças viscosas na camada limite fluidodinâmica, e é dado por

$$Re = \frac{\rho_\infty u_\infty L}{\mu}, \tag{5.21}$$

onde ρ_∞ e u_∞ representam a densidade e a velocidade característica do escoamento, L é o comprimento característico e μ é a viscosidade dinâmica. Para valores de Re grandes, há a prevalência das forças inerciais, já para valores pequenos, as forças viscosas são dominantes.

Seja Ω a região $0 \leq x, y \leq 1$ e $\partial\Omega$ o contorno dessa região. As condições de contorno para as componentes da velocidade no caso da simulação do problema da cavidade são condições de Dirichlet, ou seja, nos pontos de contorno as soluções são especificadas por

$$u(x,0,t) = 0, \quad u(x,1,t) = 1, \qquad \text{para } 0 < x < 1, t > 0 \tag{5.22}$$

$$u(0,y,t) = 0, \quad u(1,y,t) = 0, \qquad \text{para } 0 < y < 1, t > 0 \tag{5.23}$$

$$v(x,0,t) = 0, \quad v(x,1,t) = 0, \qquad \text{para } 0 < x < 1, t > 0 \tag{5.24}$$

$$v(0,y,t) = 0, \quad v(1,y,t) = 0, \qquad \text{para } 0 < y < 1, t > 0. \tag{5.25}$$

No caso do problema do duto, temos tanto condições de Dirichlet como condições de Neumann para as componentes da velocidade, mais especificamente definidas por

$$u(x,0,t) = 0, \quad u(x,1,t) = 0, \qquad \text{para } 0 < x < 1, t > 0 \tag{5.26}$$

$$u(0,y,t) = 1, \quad \frac{\partial u}{\partial x}(1,y,t) = 0, \qquad \text{para } 0 < y < 1, t > 0 \tag{5.27}$$

$$v(x,0,t) = 0, \quad v(x,1,t) = 0, \qquad \text{para } 0 < x < 1, t > 0 \tag{5.28}$$

$$v(0,y,t) = 0, \quad \frac{\partial v}{\partial x}(1,y,t) = 0, \qquad \text{para } 0 < y < 1, t > 0. \tag{5.29}$$

As condições de contorno para a pressão são condições de Neumann tanto para o problema da cavidade quanto para o caso do duto. E são definidas por

$$\frac{\partial p}{\partial y}(x,0,t) = 0, \quad \frac{\partial p}{\partial y}(x,1,t) = 0, \quad \text{para } 0 < x < 1, t > 0 \tag{5.30}$$

$$\frac{\partial p}{\partial x}(0,y,t) = 1, \quad \frac{\partial u}{\partial x}(1,y,t) = 0, \quad \text{para } 0 < y < 1, t > 0. \tag{5.31}$$

As condições iniciais para tais problemas são as seguintes

$$u(x,y,0) = 1 \text{ para } (x,y) \in \Omega \tag{5.32}$$

$$v(x,y,0) = 0 \text{ para } (x,y) \in \Omega \tag{5.33}$$

$$p(x,y,0) = 1 \text{ para } (x,y) \in \Omega. \tag{5.34}$$

5.2.2 Algoritmo

A solução das equações de Navier-Stokes é encontrada com a utilização do algoritmo **PRIME - (PRessão Implícita, Momento Explícito)** conforme [19], [16].

Iniciamos discretizando as derivadas temporais nas equações do momento através da equação (5.16) e obtemos

$$u^{n+1} = u^n + \partial t \left[\frac{1}{Re} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial (uu)}{\partial x} - \frac{\partial (uv)}{\partial y} - \frac{\partial p}{\partial x} \right] \tag{5.35}$$

$$v^{n+1} = v^n + \partial t \left[\frac{1}{Re} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial (vv)}{\partial y} - \frac{\partial p}{\partial y} \right]. \tag{5.36}$$

Em seguida utilizamos as velocidades intermediárias

$$\hat{u}^n = u^n + \partial t \left[\frac{1}{Re} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial (uu)}{\partial x} - \frac{\partial (uv)}{\partial y} \right] \tag{5.37}$$

$$\hat{v}^n = v^n + \partial t \left[\frac{1}{Re} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial (vv)}{\partial y} \right] \tag{5.38}$$

e, desta forma podemos escrever as equações (5.35) e (5.36) como

$$u^{n+1} = \hat{u}^n - \partial t \frac{\partial p}{\partial x}^{n+1} \tag{5.39}$$

$$v^{n+1} = \hat{v}^n - \partial t \frac{\partial p}{\partial y}^{n+1} \tag{5.40}$$

Derivando as equações (5.39) e (5.40) e substituindo na equação (5.18) obtemos

$$\frac{\partial^2 p}{\partial x^2}^{n+1} + \frac{\partial^2 p}{\partial y^2}^{n+1} = \frac{1}{\partial t}\left(\frac{\partial \hat{u}^n}{\partial x} - \frac{\partial \hat{v}^n}{\partial y}\right). \tag{5.41}$$

Assim, obtemos o acoplamento pressão-velocidade representado pela equação (5.41). Note que a pressão é dada implicitamente em (5.39) e (5.40), enquanto que os componentes da velocidade são calculados explicitamente, sendo corrigidos através do gradiente de pressão.

Para a implementação do algoritmo **PRIME** na malha *quadtree* são necessárias mais algumas subrotinas além das que foram apresentadas ao longo deste trabalho.

ALGORITMO: PRIME

1 $t = 0$, $n = 0$
2 Defina as condições iniciais para u, v e p.
3 **ENQUANTO** $t < t_f$
4 De acordo com o critério de convergência, escolha Δt.
5 Calcule $\hat{u}$ e $\hat{v}$.
6 Use as condições de contorno para $\hat{u}$ e $\hat{v}$.
7 **ENQUANTO** $it < itmax$ e $||res^{it} > \epsilon||$.
8 Use o método do ponto fixo.
9 Calcule $||res^{it}||$ da pressão.
10 $it = it + 1$
11 Use as condições de contorno para a pressão.
12 Calcule u^{n+1} e v^{n+1}.
13 Use as condições de contorno para u^{n+1} e v^{n+1}.
14 $t = t + \Delta t$
15 $n = n + 1$

De acordo com [19], temos que escolher Δt de modo a garantir a estabilidade e evitar oscilações na solução. Para isto são usadas três condições para estimar o valor de Δt, que são

$$|u_{max}|\Delta t < \Delta x, \quad |v_{max}|\Delta t < \Delta y \quad e \quad \frac{2\Delta t}{Re} = \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)^{-1} \tag{5.42}$$

onde as duas primeiras são conhecidas como *condições de Courant-Friedrichs-Lewy (CFL)*, com $|u_{max}|$ e $|v_{max}|$ representando os valores máximos absolutos da velocidade na malha. Estas condições significam que uma partícula não pode se mover a uma distância maior que o espaçamento da malha em um intervalo de tempo Δt .

Desta forma, Δt deve ser escolhido de maneira a satisfazer as três condições. Colocamos

$$\Delta t = \tau \cdot min\left(\frac{Re}{2}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right), \frac{\Delta x}{|u_{max}|}, \frac{\Delta y}{|v_{max}|}\right), \tag{5.43}$$

onde $0 < \tau \leq 1$ é chamado de *fator de segurança*. O valor de Δt é fixado no início das iterações.

5.3 Tempo de Execução com os Vizinhos na Árvore

O módulo com as subrotinas para o caso da simulação da equação do calor tem em torno de 1500 linhas e em torno de $\frac{1}{3}$ das subrotinas utilizam os vizinhos dos nodos. Sendo assim, duas perguntas básicas surgiram: o tempo de execução muda se os vizinhos forem guardados na árvore ao invés de serem procurados quando necessários? Tudo indica que sim, mas diminui de forma significativa?

Com o intuito de responder estas questões fizemos um pequeno teste. Utilizamos um computador Intel(R) Celeron(R) CPU 2.13 GHz com memoria RAM de 2 GB. O tempo médio de execução de 5 simulações foi de $232.260633s$ para o caso de procura dos vizinhos dos nodos em cada subrotina em que eram necessários e de $64.672180s$ utilizando a subrotina que procura e guarda os vizinhos logo após a geração da malha.

Isto confirma que a resposta para a nossa primeira questão realmente foi positiva, o tempo realmente diminui, e pelos resultados podemos afirmar que diminui significativamente, pois há uma diferença de cerca de 72%.

Mas precisamos, ressalta que no primeiro caso, a memória necessária é de 56 bytes para cada nó e de 72 bytes para o segundo caso. Este cálculo está baseado no armazenamento dos valores das variáveis básicas como: pai, nível, dx, dy, cx, cy, x e y, sendo as primeiras variáveis definidas por números inteiros e as demais por números reais. Portanto para uma malha uniforme de nível n há um aumento da quantidade de memória necessária de $16 \cdot \frac{4^n-1}{3}$ bytes. Sendo assim, no caso de uma malha de nível 5, por exemplo, teremos um aumento de 5456 bytes. Portanto, mesmo que tenhamos que utilizar mais memória, o ganho de tempo é significativo para o caso apresentado.

5.4 Comparando Malha Cartesiana com Malha *Quadtree*

Durante o desenvolvimento do trabalho surgiram certas perguntas, tais como: "Em que situações vale a pena pagar o preço por uma malha quadtree?" Neste seção buscaremos resposta a esta pergunta ou pelo menos, algumas pistas sobre esta questão.

Iniciamos simulando o problema da equação do calor com condições de contorno 5.3 e 5.4, e condição inicial definida por 5.5 da subseção 5.1.1 deste capítulo. O problema foi iterado até que se obtivesse a solução permanente, ou seja, até que $\frac{\partial u}{\partial t} = 0$. Desta forma estamos obtendo a solução da equação de Poisson sem dependência no tempo. Na figura 5.1 temos o resultado com 10423 iterações com tempo de execução de $24.13s$ para a malha cartesiana de 64×64 pontos e na figura 5.2 temos o resultado para uma malha *quadtree* de nível 7, onde foram necessários $119.74s$ com o mesmo número de iterações da malha cartesiana. Para ambas as malhas a tolerância especificada foi de 1×10^{-5}.

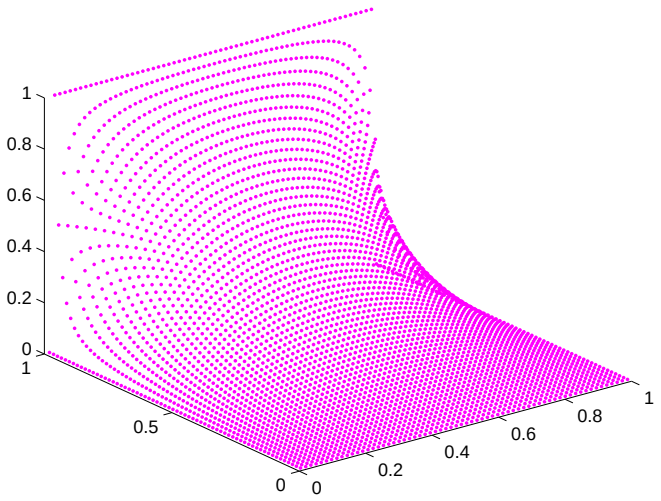

Figura 5.1: *Solução estacionária do problema de condução de calor através de uma malha cartesiana uniforme.*

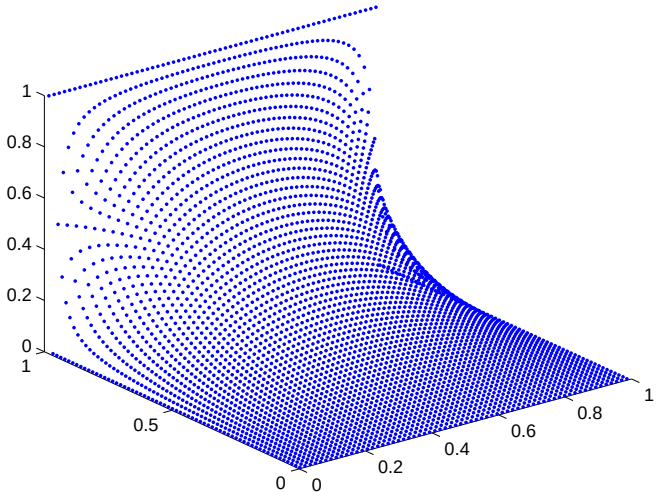

Figura 5.2: *Solução estacionária do problema de condução de calor através de uma malha quadtree de nível* 7.

A figura 5.3 mostra uma malha *quadtree* com 3 níveis, sendo que aproximadamente 50% da malha está no nível 5 e os outros 50% nos níveis 6 e 7 e na figura 5.4 temos a solução do problema de condução de calor simulada utilizando a malha 5.3 com a mesma tolerância do teste anterior. Neste caso, foram necessárias 10211 iterações com tempo de execução de 32.95s.

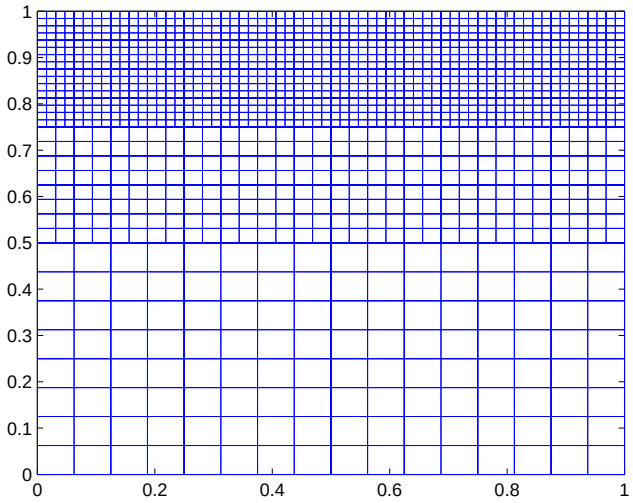

Figura 5.3: *Primeira malha quadtree de níveis* 5,6 *e* 7.

No próximo teste foi utilizada uma malha com 3 níveis de refinamento nas proporções indicadas na figura 5.5. Com 10112 iterações num tempo de execução de 32.26s obtivemos a solução, veja na figura 5.6.

Diminuindo um pouco mais a proporção do nível 7 e aumentando a do nível 5 gerou-se a malha conforme a figura 5.7 e a solução apresentada na figura 5.8 foi obtida após 18.97s num total de 9920 iterações.

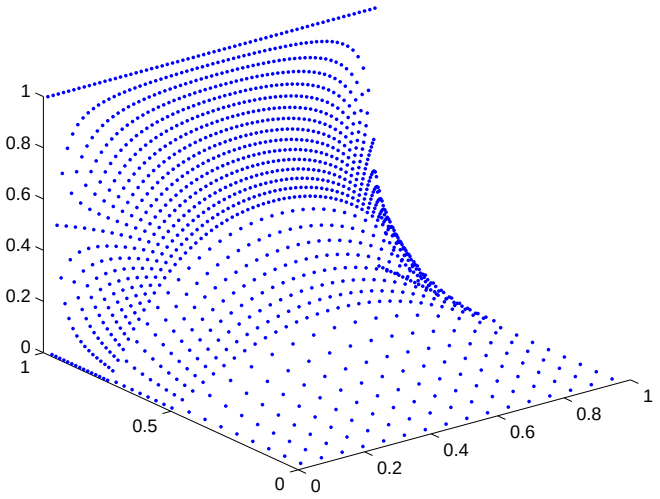

Figura 5.4: *Solução estacionária do problema de condução de calor utilizando a malha da figura 5.3.*

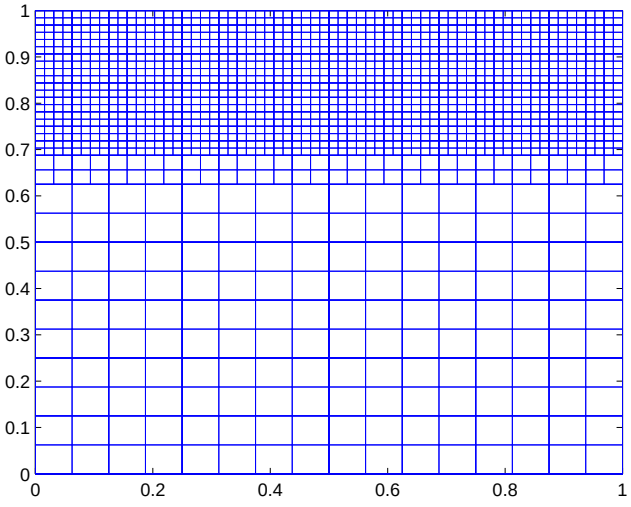

Figura 5.5: *Segunda malha quadtree de níveis* 5,6 *e* 7.

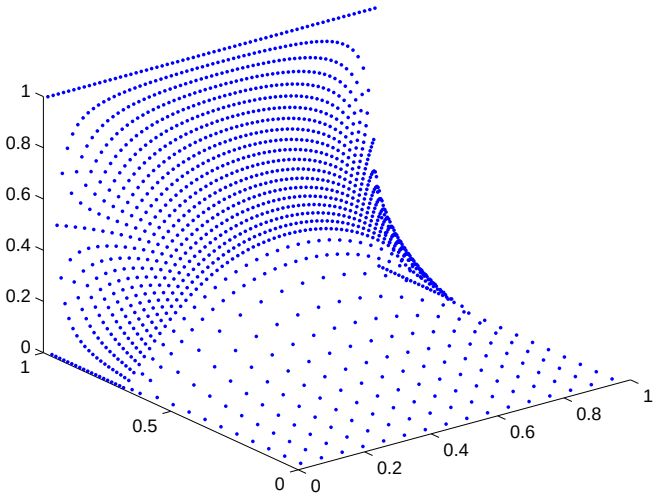

Figura 5.6: *Solução estacionária do problema de condução de calor utilizando a malha da figura 5.5.*

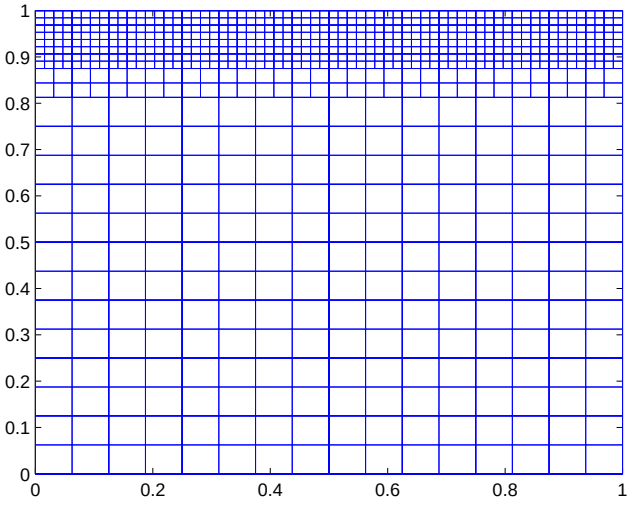

Figura 5.7: *Terceira malha quadtree* 3 *de níveis* 5,6 *e* 7.

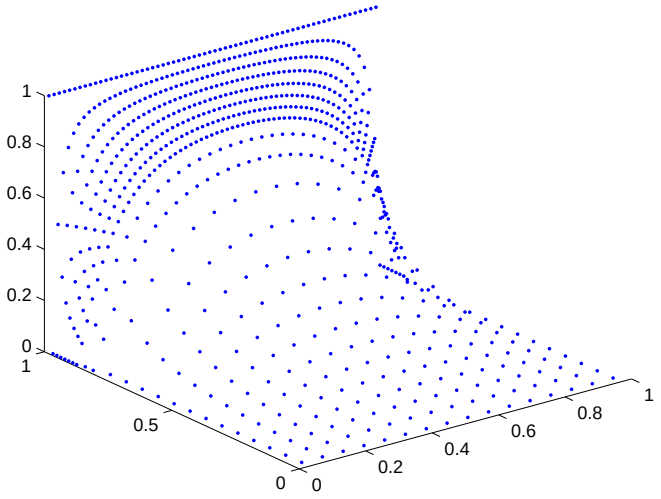

Figura 5.8: *Solução estacionária do problema de condução de calor utilizando a malha da figura 5.7.*

Podemos concluir que quando a malha *quadtree* uniforme 64×64 é comparada com uma malha cartesiana, não há nenhuma vantagem. Pelo contrário com a malha cartesiana obtemos o mesmo resultado em um tempo menor. Este teste também nos ajudou a verificar que o número de iterações nos dois casos foi o mesmo. Isto mostra que estamos obtendo exatamente a mesma solução apesar de estarmos usando estruturas matemáticas diferentes (malha *quadtree* e malha cartesiana).

Já quando utilizamos malhas com níveis diferentes e portanto, menos nodos folha, podemos observar dois aspectos. Primeiro, não há grandes perturbações, pelo menos visíveis, próximo à mudança de níveis. Isto indica que as interpolações estão funcionando de forma desejável. Claro que não podemos concluir algo definitivo apenas com este teste, mas ele é um bom indicador. O segundo aspecto é que nos 3 casos apresentados a malha *quadtree* foi mais eficiente que a malha cartesiana uniforme no que se refere ao número de iterações necessárias para a obtenção da solução. No último caso, a malha *quadtree* superou inclusive o tempo de execução.

Com este pequeno teste percebemos que a malha *quadtree* nem sempre é vantajosa, pois uma malha cartesiana é simples de ser gerada, enquanto que uma malha quadtree requer um pouco mais de tempo de trabalho. Mas se torna um bom instrumento quando queremos uma solução utilizando uma malha não uniforme.

Podemos salientar também que uma das grandes vantagens na utilização de malhas *quadtree* é que, com muito menos pontos na malha podemos obter a mesma solução, com o mesmo nível de acurácia. Para isto devemos concentrar pequenos quadrados próximos da região do domínio onde a solução possuir maior gradiente e quadrados maiores onde a solução não tiver grandes variações.

5.5 Verificando a Ordem

Pelos testes apresentados até aqui, tudo indica que a malha *quadtree* funciona de forma aceitável. Agora, vamos verificar o que acontece com a ordem do método quando utilizamos a malha *quadtree* construída neste trabalho. Conforme definido no capítulo anterior, o problema de condução de calor é usado nesta seção, portanto os gráficos que serão mostrados referem-se à solução deste problema.

5.5.1 Malha *Quadtree* Uniforme

Inicialmente apresentaremos os resultados para a malha uniforme, pois neste caso não há a necessidade de utilizar as subrotinas de interpolações. A ideia é obter a solução da equação do calor através de malhas uniformes com 32×32, 64×64 e 128×128 pontos partindo da solução exata do problema. Sendo assim, se conhecemos a solução u da equação (5.2), podemos obter f e usando u como condição de contorno podemos verificar o quão boa está a aproximação obtida numericamente.

Seja $u = x+y$ a solução da equação (5.2), então $f = 0$ e as condições de contorno e iniciais são especificadas por 5.6 e 5.7, respectivamente. Assim, é possível

calcular o erro e gerar gráficos que mostrem a evolução do erro com o passar das iterações. Observe os gráficos 5.9, 5.10 e 5.11 que mostram como o erro se comporta ao longo das iterações para o caso em que a solução de (5.2) é $u = x + y$.

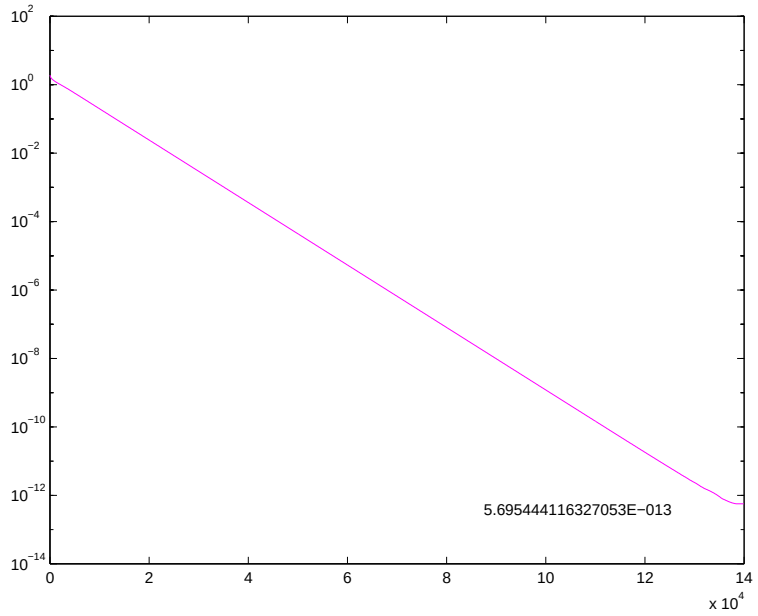

Figura 5.9: *Gráfico do erro versus o número de iterações com* $f = 0$ *e malha com espaçamento* $dx = dy = \frac{1}{32}$*, com a indicação do valor do erro quando a solução estacionária é obtida.*

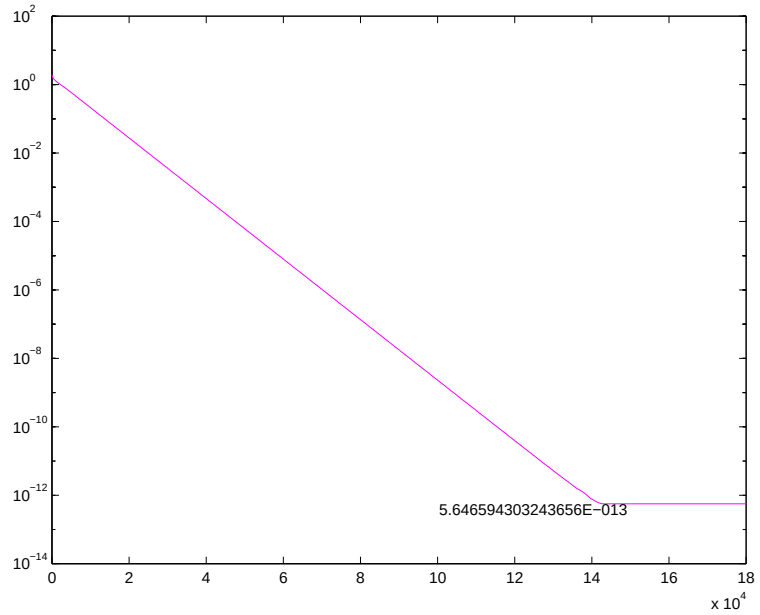

Figura 5.10: *Gráfico do erro versus o número de iterações com $f = 0$ e malha com espaçamento $dx = dy = \frac{1}{64}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

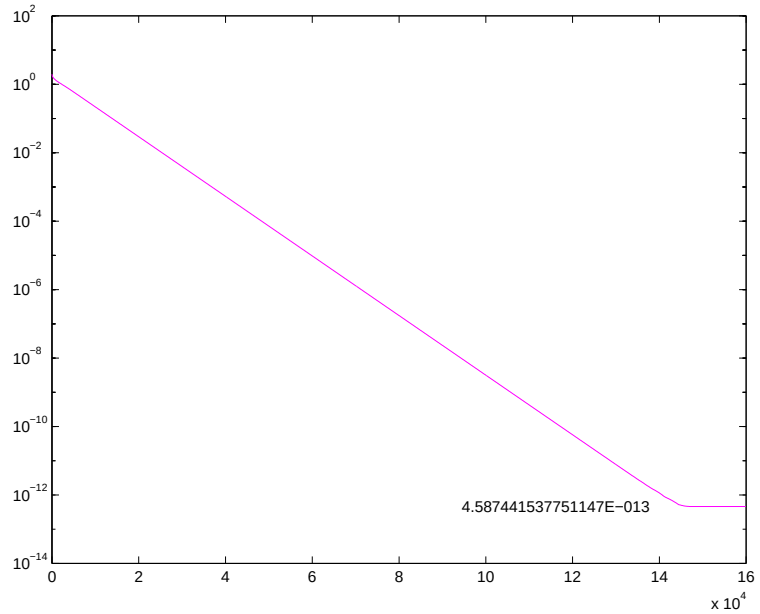

Figura 5.11: *Gráfico do erro versus o número de iterações com $f = 0$ e malha com espaçamento $dx = dy = \frac{1}{128}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

Agora consideremos a solução da equação 5.2 como $u = x^2 + y^2$. Logo $f = 4$ e as condições de contorno e iniciais são dadas por 5.8 e 5.9, respectivamente. Da mesma forma como no caso anterior, fizemos a simulação utilizando malhas com espaçamentos $dx = dy = \frac{1}{32}$, $dx = dy = \frac{1}{64}$ e $dx = dy = \frac{1}{128}$. Veja a figura 5.12 que mostra o decaimento do erro com o passar das iterações para o caso em que o espaçamento da malha é $dx = dy = \frac{1}{128}$.

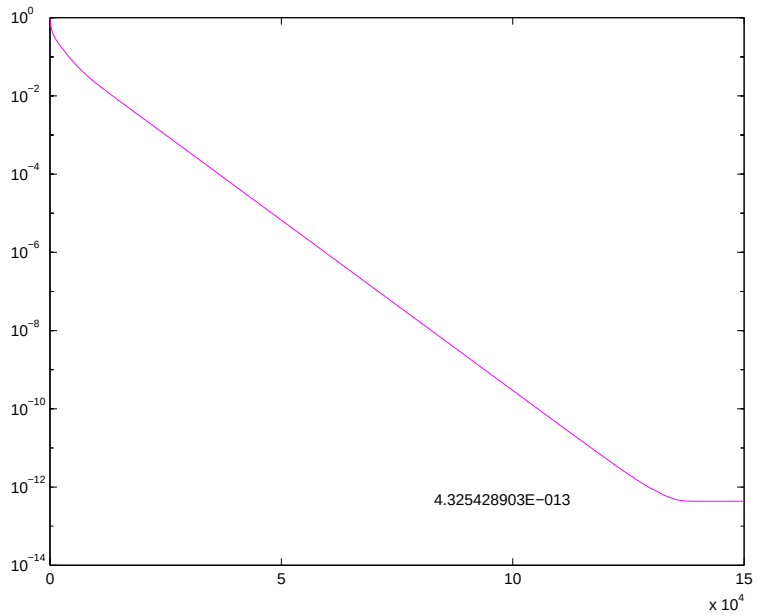

Figura 5.12: *Gráfico do erro versus o número de iterações com $f = 4$ e malha com espaçamento $dx = dy = \frac{1}{128}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

O gráfico da figura 5.13 representa a evolução do erro ao longo das iterações para o caso em que o espaçamento da malha é $dx = dy = \frac{1}{128}$ e a solução da equação de calor é $u = x^3 + y^3$, com condições de contorno definidas por 5.10 e condição inicial 5.11.

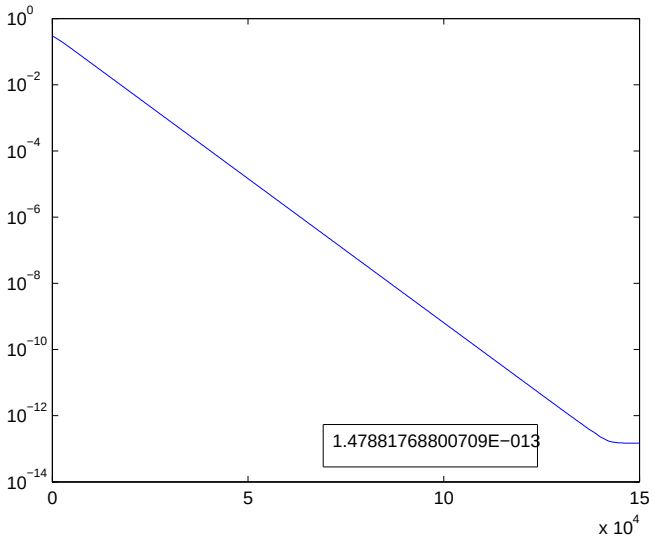

Figura 5.13: *Gráfico do erro versus o número de iterações com $f = 6x + 6y$ e malha com espaçamento $dx = dy = \frac{1}{128}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

Os erros obtidos até o momento são da ordem do erro de máquina, portanto é interessante ainda verificarmos o que acontece quando a solução é $u = x^4 + y^4$. Agora vemos pelos gráficos 5.14, 5.15 e 5.16 que o erro diminui conforme as iterações aumentam, mas é maior que o erro de máquina, portanto neste caso há presente também erro da discretização do método.

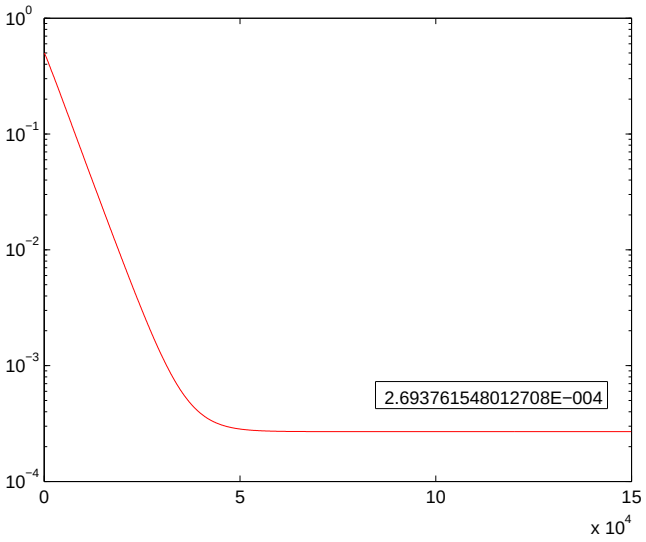

Figura 5.14: *Gráfico do erro versus o número de iterações com* $f = 12x^2 + 6y^2$ *e malha com espaçamento* $dx = dy = \frac{1}{32}$*, com a indicação do valor do erro quando a solução estacionária é obtida.*

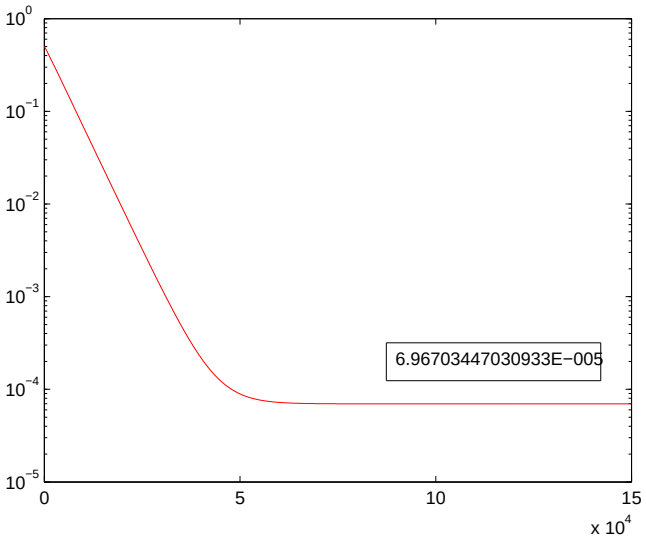

Figura 5.15: *Gráfico do erro versus o número de iterações com* $f = 12x^2 + 12y^2$ *e malha com espaçamento* $dx = dy = \frac{1}{64}$*, com a indicação do valor do erro quando a solução estacionária é obtida.*

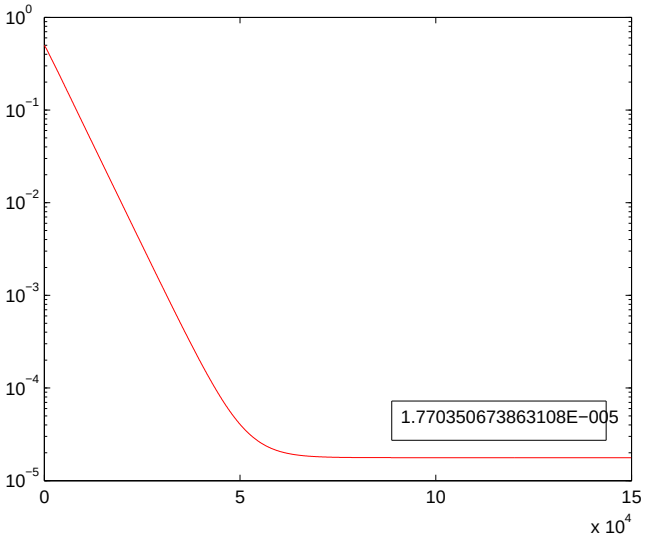

Figura 5.16: *Gráfico do erro versus o número de iterações com* $f = 12x^2 + 12y^2$ *e malha com espaçamento* $dx = dy = \frac{1}{128}$*, com a indicação do valor do erro quando a solução estacionária é obtida.*

Segundo [10], um método apresenta a ordem p desejada na região $h_{min} < h < h_{max}$, que varia com o problema estudado. Desta forma, quando estamos nesta região a ordem do método pode ser aproximada da seguinte maneira. Para um determinado $t = T^*$ fixo, considere:

- a solução $u(T^*)$,
- a solução obtida com espaçamento h, denotada por u^h,
- a solução obtida com espaçamento $\frac{h}{2}$, denotada por $u^{\frac{h}{2}}$,
- a solução obtida com espaçamento $\frac{h}{4}$, denotada por $u^{\frac{h}{4}}$,
- e assim por diante.

Então podemos estimar p diretamente de

$$p \approx \frac{\log\left(\frac{\|u^{\frac{h}{2}} - u^h\|}{\|u^{\frac{h}{4}} - u^{\frac{h}{2}}\|}\right)}{\log 2} \tag{5.44}$$

Neste caso, através de (5.44) e os dados obtidos nas simulações obtemos

$$p \approx \frac{\log(2.6937 \times 10^{-4}) - \log(1.7703 \times 10^{-5})}{\log(\frac{1}{32}) - \log(\frac{1}{128})} \approx 1.96. \tag{5.45}$$

Isto significa que o método é de ordem 2.

5.5.2 Malha *Quadtree* com Níveis Diferentes

Procedendo da mesma maneira como nos casos da malha uniforme, aqui utilizaremos as malhas conforme as figuras 5.17, 5.18 e 5.19, onde estas apresentam 3 níveis de refinamento, sendo que os níveis mais refinados apresentam $dx = dy = \frac{1}{32}$, $dx = dy = \frac{1}{64}$ e $dx = dy = \frac{1}{128}$, respectivamente, ou seja, os espaçamentos dessas malhas diminuem em $\frac{dx}{2}$ de uma para outra, como fizemos no caso anterior.

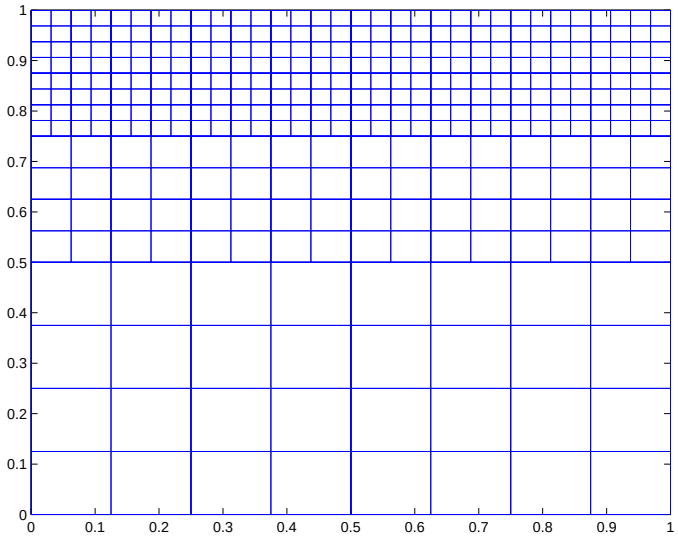

Figura 5.17: *Malha utilizadas para testar a ordem do método com o menor espaçamento* $dx = dy = \frac{1}{32}$.

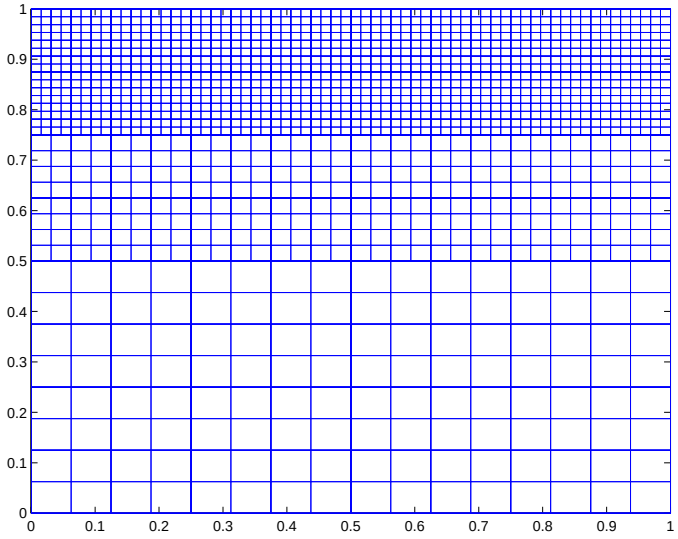

Figura 5.18: *Malhas utilizadas para testar a ordem do método com o menor espaçamento* $dx = dy = \frac{1}{64}$.

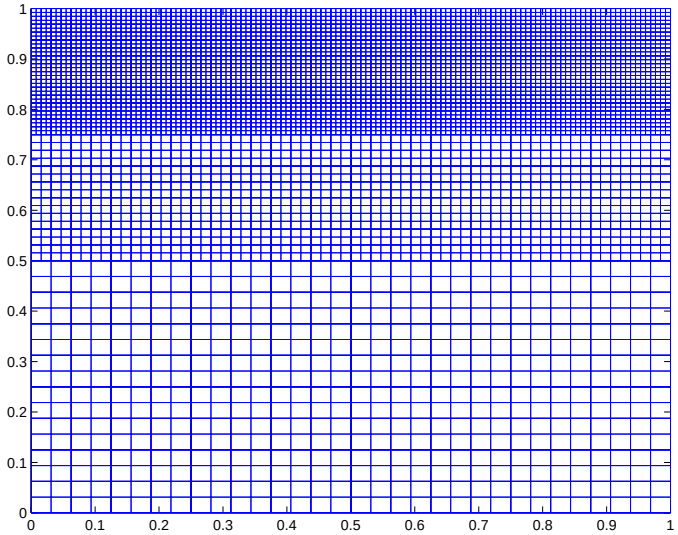

Figura 5.19: *Malhas utilizadas para testar a ordem do método com o menor espaçamento* $dx = dy = \frac{1}{128}$.

Para o caso em que a solução é $u = x + y$ as condições de contorno são conforme 5.6 e condição inicial 5.7. O erro apresentado nos resultados é da ordem de máquina. Veja para o caso mais refinado apresentado na figura 5.20 como o erro se comporta ao longo das iterações.

Procedendo da mesma maneira como no caso anterior, mas agora considerando as condições de contorno definidas por 5.8 com condição inicial dada por 5.9, percebemos pelo gráfico da figura 5.21 que o erro diminui até certo ponto e depois começa a aumentar, mas se estabiliza. E comparando com o caso uniforme, percebemos que agora o erro do método já aparece para o polinômio de grau 2.

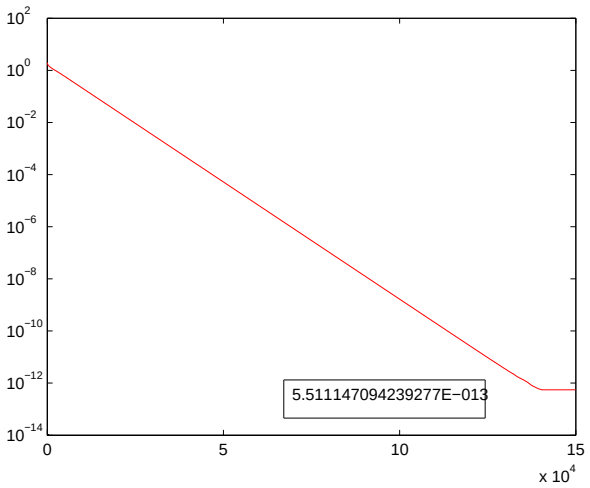

Figura 5.20: *Gráfico do do erro versus o número de iterações com $f = 0$ e malha com espaçamentos $\frac{1}{32}$, $\frac{1}{64}$ e $\frac{1}{128}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

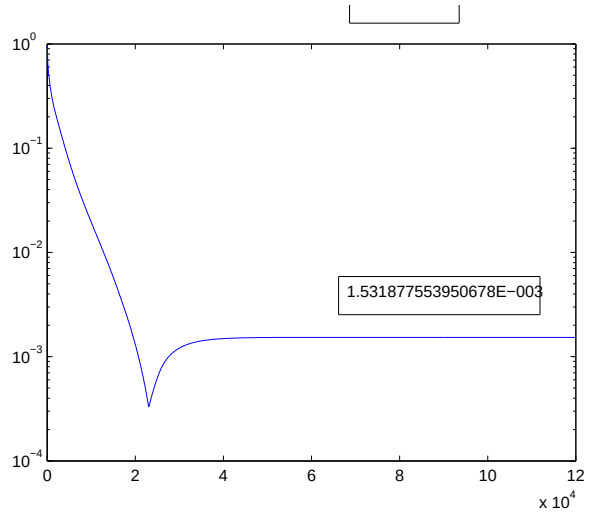

Figura 5.21: *Gráfico do erro versus o número de iterações com $f = 4$ e malha com espaçamentos $\frac{1}{32}$, $\frac{1}{64}$ e $\frac{1}{128}$, com a indicação do valor do erro quando a solução estacionária é obtida.*

Usando a relação 5.44 com os dados obtidos nas simulações obtemos

$$p \approx \frac{\log(5.4605 \times 10^{-3}) - \log(1.5318 \times 10^{-3})}{\log(\frac{1}{32}) - \log(\frac{1}{128})} \approx 0.9169 \tag{5.46}$$

o que mostra que o erro para este caso é de ordem 1. Para os casos $u = x^3 + y^3$ e $u = x^4 + y^4$ a ordem obtida também é próxima a 1.

5.5.3 Conclusão dos Testes

Retomando, temos que para o caso da malha uniforme obtivemos apenas o erro de máquina para $u = x+y$, $u = x^2+y^2$ e $u = x^3+y^3$. Já para $u = x^4+y^4$ o erro é de ordem 2. Para entendermos melhor isso, vamos lembrar como obtemos a discretização de uma função para a derivada de segunda ordem em diferença central através da série de Taylor. Considere as expansões

$$p(x+h) = p(x) + hp'(x) + \frac{h^2}{2!}p''(x) + \frac{h^3}{3!}p'''(x) + \frac{h^4}{4!}p^{(4)}(x) + \mathcal{O}(h^4) \quad (5.47)$$

$$p(x-h) = p(x) - hp'(x) + \frac{h^2}{2!}p''(x) - \frac{h^3}{3!}p'''(x) + \frac{h^4}{4!}p^{(4)}(x) + \mathcal{O}(h^4) \quad (5.48)$$

Somando 5.47 e 5.48 obtemos

$$p(x+h) + p(x-h) = 2p(x) + 2\frac{h^2}{2!}p''(x) + 2\frac{h^4}{4!}p^{(4)}(x) + \mathcal{O}(h^4) \quad (5.49)$$

isolando a derivada de segunda ordem em 5.49 temos

$$p''(x) = \frac{p(x+h) - 2p(x) + p(x-h)}{h^2} + \mathcal{O}(h^2)p^{(4)}(x) \quad (5.50)$$

que representa a discretização utilizada nos testes. Lembramos que o termo $\mathcal{O}(h^2)$ refere-se à constante da derivada de quarta ordem.

Voltando à nossa análise, temos que as derivadas de quarta ordem das funções $u = x + y$, $u = x^2 + y^2$ e $u = x^3 + y^3$ são iguais a zero, portanto todos os termos restantes da série são nulos e por isso nos testes encontramos apenas o erro de máquina. Já para o caso de $u = x^4 + y^4$, a derivada quarta é diferente de 0 e, portanto existe um erro, que conforme 5.50 é de ordem 2 como constatado nos testes.

Para a malha não uniforme estamos utilizando interpolações de primeira ordem, e por isso o erro aparece para o caso $u = x^2 + y^2$ que é de ordem 1. Logo a ordem do método diminui devido ao tipo de interpolação utilizada.

5.6 Exemplos

Nesta seção são apresentados alguns exemplos de resolução de problemas utilizando malhas *quadtree* diversas. No primeiro exemplo temos a solução da equação de calor 5.2 para o caso de uma malha uniforme 64×64 e com condição de contorno especificadas por 5.14 e condição inicial dada por 5.15, com $r = 0.2$ e coordenadas do centro do círculo dadas por $(0.5, 0.5)$. Veja a figura 5.22.

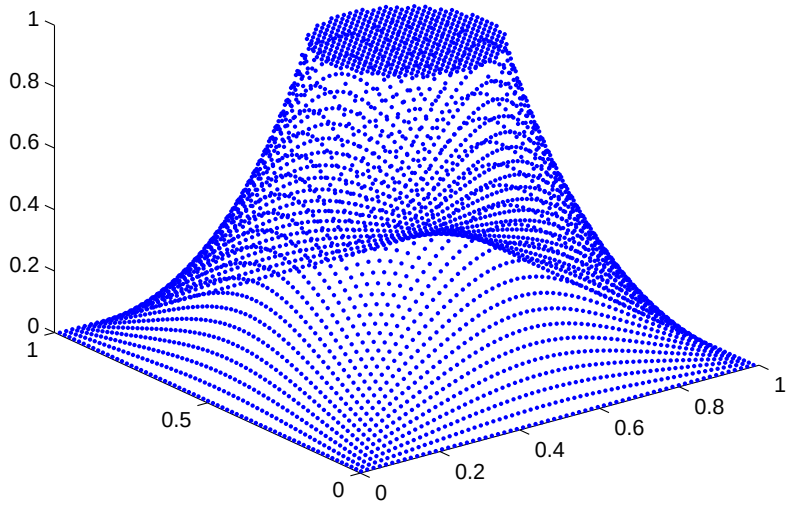

Figura 5.22: *Solução para a condução de calor.*

Agora, resolvendo o caso anterior numa malha conforme a figura 5.23 de nível 2 e aplicando 4 vezes a subrotina que refina ao longo de círculo de raio 0.2 e com centro $(0.5, 0.5)$, obtemos a solução conforme a figura 5.24.

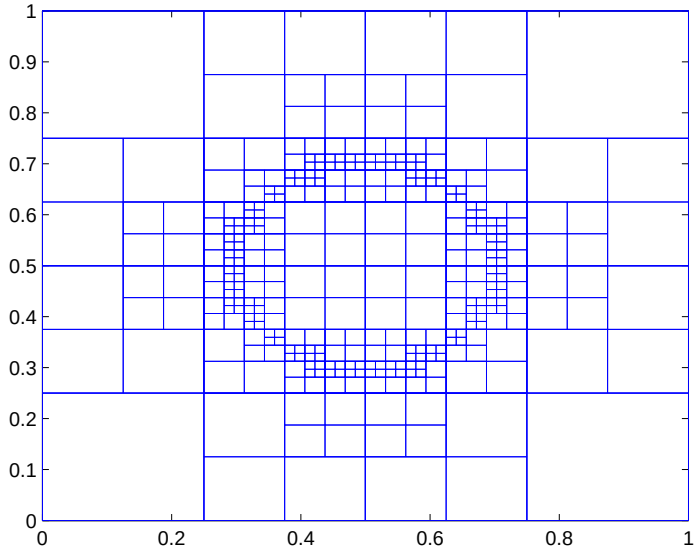

Figura 5.23: *Malha quadtree refinada ao longo de um círculo de raio* 0.2 *e centro* $(0.5, 0.5)$.

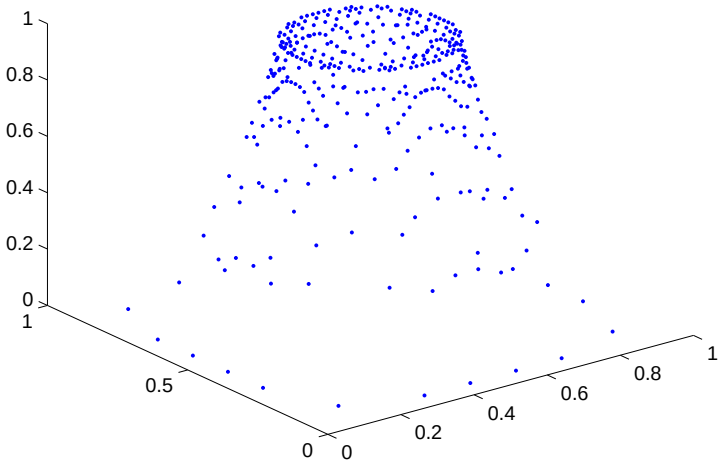

Figura 5.24: *Solução da equação do calor utilizando a malha da figura 5.23.*

Analisando os resultados apresentados nas figuras 5.22 e 5.23 é possível perceber que com menos quadrados na malha, obtemos a mesma solução. Isto indica uma grande vantagem das malhas *quadtree*, pois com menos quadrados, menos memória e menos tempo são necessários.

Utilizando uma malha como da figura 5.25 de nível 4 e refinada 4 vezes ao longo do círculo de raio 0.2 e centro $(0.5, 0.5)$, obtemos a solução conforme a figura 5.26 para o mesmo problema anterior.

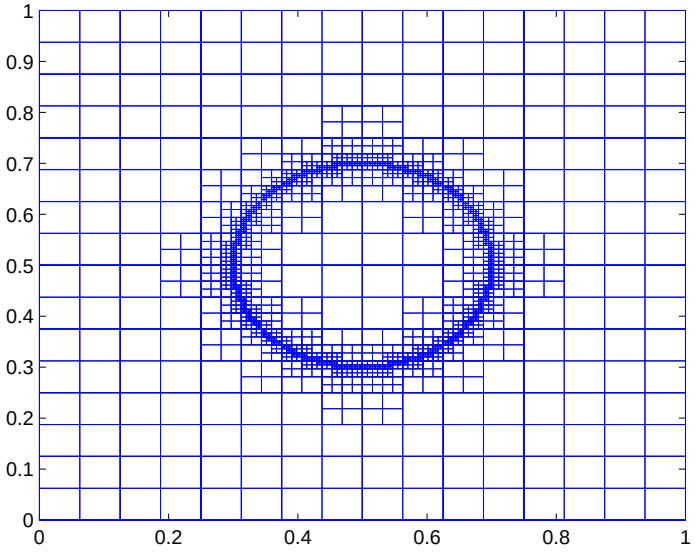

Figura 5.25: *Malha quadtree refinada ao longo de um círculo de raio* 0.2 *e centro* $(0.5, 0.5)$.

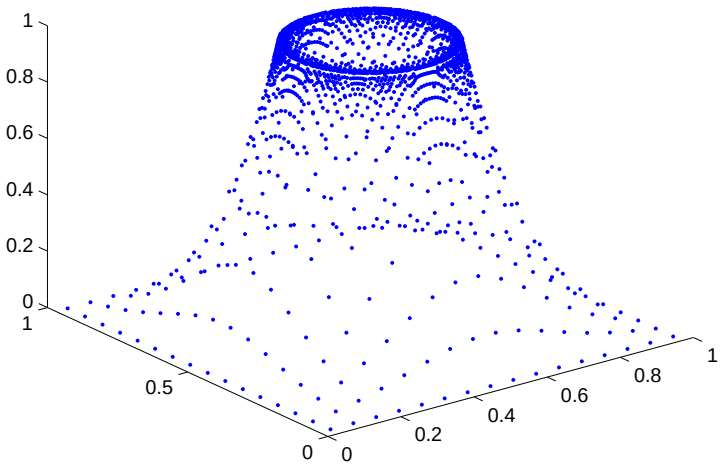

Figura 5.26: *Solução da equação do calor utilizando a malha da figura 5.25.*

Os próximos exemplos são casos de simulação de fluidos através das equações de Navier-Stokes (5.19) e (5.20). Primeiro, mostraremos os resultados obtidos na simulação de escoamento de um fluido em uma cavidade quadrada, cujas condições de contorno são definidas por 5.22, 5.23, 5.24 e 5.25 para a velocidade e para a pressão por 5.30 e 5.31, com condições iniciais conforme 5.32, 5.33 e 5.34.

A figura 5.27 mostra a solução obtida para a simulação do escoamento de um fluido em uma cavidade quadrada com $Re = 100$, onde é possível identificarmos os vórtices que são padrão deste tipo de simulação, ou seja, um vórtice maior na região mais central da cavidade e dois vórtices menores nos cantos inferiores.

Na figura 5.28 temos o gráfico de vetores da velocidade obtido na simulação de escoamento de um fluido no interior de uma cavidade quadrada com $Re = 400$ utilizando uma malha uniforme com espaçamento $\frac{1}{128}$, onde é possível identificarmos o vórtice principal e os dois vórtices secundários no inferior da cavidade.

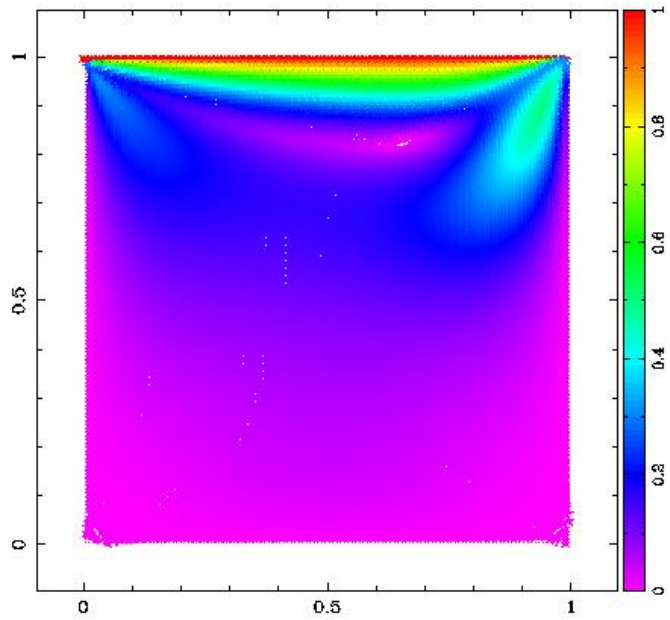

Figura 5.27: *Solução da equação do escoamento em uma cavidade quadrada com* $Re = 100$ *utilizando uma malha uniforme* 128×128. *Gráfico dos vetores velocidade.*

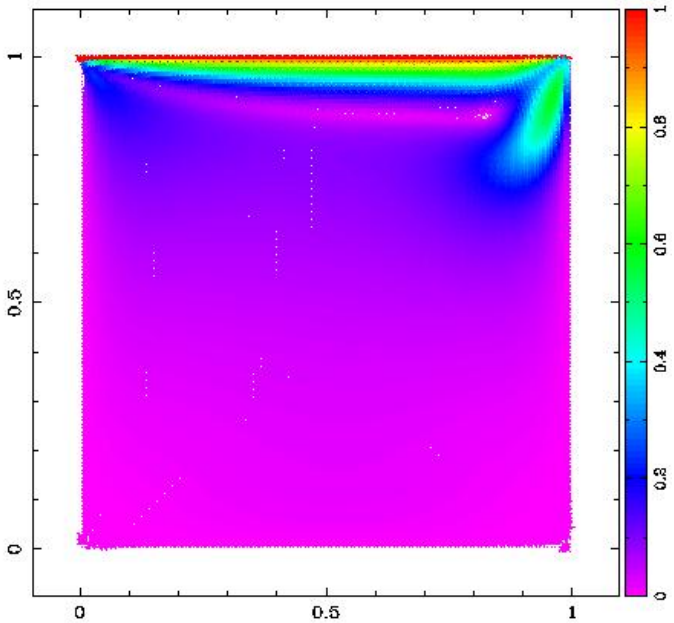

Figura 5.28: *Solução da equação do escoamento em uma cavidade quadrada com* $Re = 400$ *utilizando uma malha uniforme* 128×128. *Gráfico dos vetores velocidade.*

Também realizamos testes de simulação do problema da cavidade com malhas não uniformes. A primeira malha utilizada está representada na figura 5.29, onde há três níveis de refinamento sendo o menor deles com espaçamento $\frac{1}{64}$ nos cantos do domínio. E, os resultados obtidos para os números de Reynolds $Re = 100$ e $Re = 400$ estão representados através do gráfico de vetores da velocidade nas figuras 5.30 e 5.31, respectivamente.

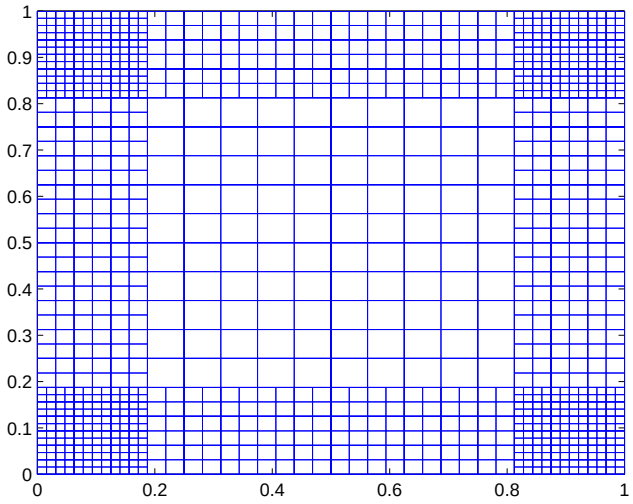

Figura 5.29: *Malha quadtree não uniforme para a simulação do problema da cavidade.*

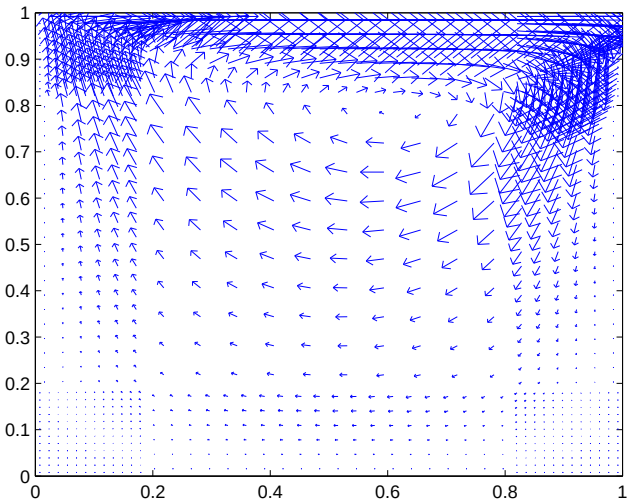

Figura 5.30: *Solução da equação do escoamento em uma cavidade quadrada com* $Re = 100$ *utilizando uma malha conforme a figura* (5.29). *Gráfico dos vetores velocidade.*

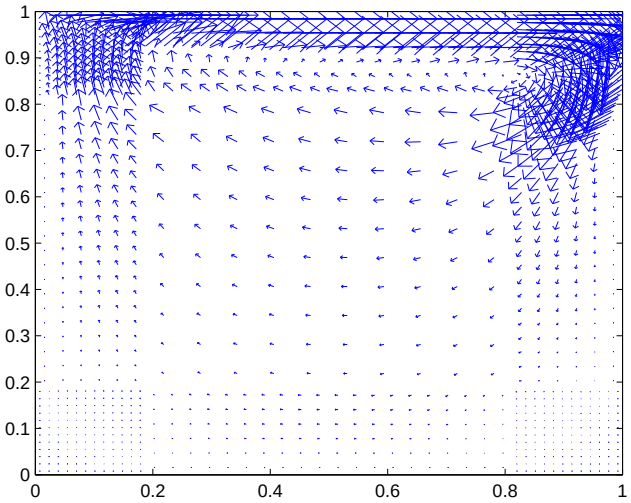

Figura 5.31: *Solução da equação do escoamento em uma cavidade quadrada com* $Re = 400$ *utilizando uma malha conforme a figura* (5.29). *Gráfico dos vetores velocidade.*

Na figura 5.32 temos a próxima malha utilizada nos testes de simulação de escoamento no interior de uma cavidade, cuja solução pode ser verificada na figura 5.33.

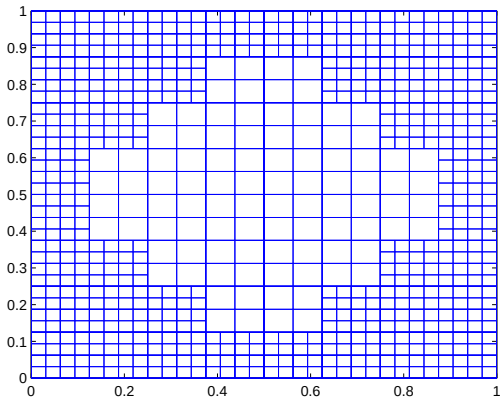

Figura 5.32: *Malha quadtree não uniforme com refinamento nas extremidades para a simulação do problema da cavidade.*

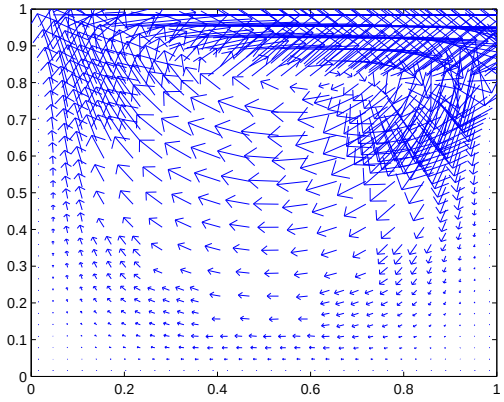

Figura 5.33: *Solução da equação do escoamento em uma cavidade quadrada com* $Re = 100$ *utilizando a malha conforme a figura* (5.32).

Continuando as simulações de fluidos através da equação de Navier-Stokes, mas agora utilizando as condições de contorno e iniciais apresentadas na subseção 5.2.1 para o duto, obtemos o gráfico de vetores da velocidade da figura 5.34 como solução do escoamento de um fluido no interior de um duto reto quadrado para $Re = 100$, onde é possível identificarmos a camada limite e também o perfil parabólico esperado neste tipo de escoamento.

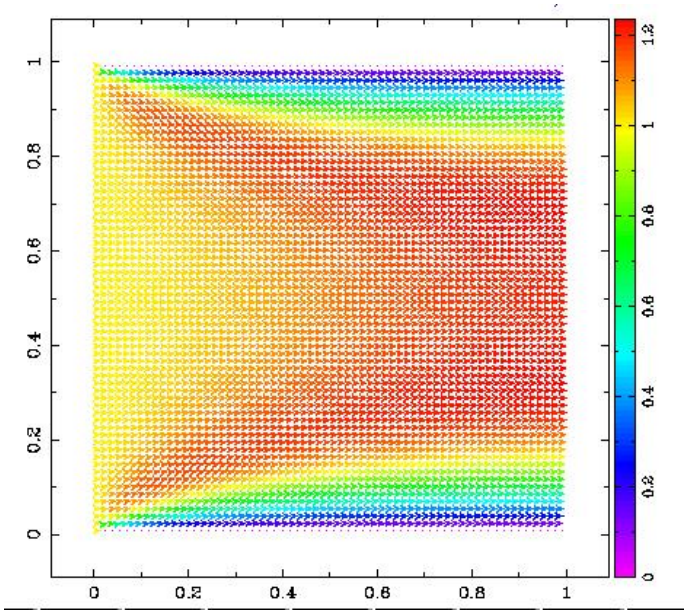

Figura 5.34: *Solução da equação do escoamento em um duto quadrado com* $Re = 100$ *utilizando uma malha uniforme* 64×64. *Gráfico dos vetores velocidade.*

O último teste realizado é de um duto reto 3×3 com um obstáculo circular e $Re = 500$ utilizando uma malha quadtree uniforme 256×256 pontos, o resultado obtido é mostrado na figura 5.35.

As soluções dos gráficos 5.27, 5.28 e 5.34 tem por objetivo mostrar que a malha *quadtree* para o caso uniforme implementada neste trabalho é eficiente na simulação de escoamento de fluidos, cuja implementação numa malha cartesiana já é, por si só, um caso complexo. Este tipo de simulação numa malha *quadtree* torna-se um pouco mais complexa, pois para o caso de uma malha com espaçamen-

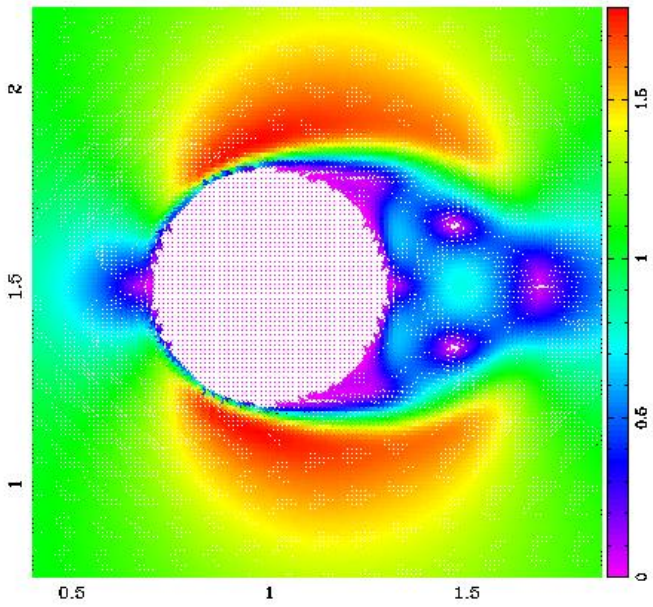

Figura 5.35: *Solução da equação do escoamento em um duto quadrado com* $Re = 500$ *utilizando uma malha uniforme* 256×256 *com obstáculo circular. Gráfico dos vetores velocidade.*

tos diferentes, precisamos usar interpolações para todas as variáveis envolvidas no problema, mas também sabemos que na simulação de fluidos é essencial termos uma malha refinada onde há maior variação de gradiente e, que ao mesmo tempo, seja eficiente.

6 CONCLUSÃO

O estudo de geração de malhas é um tema muito importante no que se refere à simulação de escoamento de fluidos, pois uma boa solução é obtida quando utiliza-se uma malha de qualidade. Nosso objetivo foi estudar as malhas *quadtree*, por acreditarmos que estas sejam uma alternativa na geração de malhas eficientes na simulação de fluxos. Em pesquisa realizada durante o desenvolvimento deste trabalho, percebemos que há um crescente interesse neste tipo de malha, principalmente em malhas que se adaptam automaticamente ao longo de uma simulação conforme critérios estabelecidos.

O algoritmo implementado e apresentado ao longo desta dissertação supriu as nossas expectativas, pois conseguimos gerar malhas com diversos tipos de refinamento, o que já é um bom começo. Nosso interesse não foi criar o algoritmo mais eficiente, mas sim criar uma alternativa inicial de geração de malhas *quadtree*, para em trabalhos futuros desenvolvermos malhas adaptativas e até quem sabe, malhas tridimensionais *octree*.

A ordem do método para malhas *quadtree* não uniformes verificada no capítulo 5 pode ser melhorada se utilizarmos interpolações de maior ordem. Um exemplo disto pode ser encontrado em [27], onde interpolações de segunda ordem são utilizadas na discretização do operador gradiente.

Enfim, podemos concluir que o método apresentado neste trabalho é uma boa ferramenta, pois possui uma grande diversidade de malhas com refinamentos que a princípio podem ser definidos pelo usuário, mas que podem, como já dissemos serem geradas automaticamente. Mesmo da forma apresentada já podemos identificar sua relevante importância na geração de malhas com contornos circulares, por exemplo, pois como dissemos na introdução, este tipo de malha requer um tratamento especial.

Referências Bibliográficas

[1] G. Agresar, JJ Linderman, G. Tryggvason, and KG Powell. An adaptive, Cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells. *Journal of Computational Physics*, 143(2):346–380, 1998.

[2] M. Bern and D. Eppstein. *Mesh generation and optimal triangulation*. XEROX Corp., Palo Alto Research Center, 1992.

[3] AGL Borthwick, RD Marchant, and GJM Copeland. Adaptive hierarchical grid model of water-borne pollutant dispersion. *Advances in Water Resources*, 23(8):849–865, 2000.

[4] W.E. Boyce and R.C. DiPrima. *Elementary differential equations and boundary value problems*. Wiley New York, 1977.

[5] G.C. Cohen. *Higher-order numerical methods for transient wave equations*. Springer Verlag, 2002.

[6] A. de Oliveira Fortuna. *Técnicas computacionais para dinâmica dos fluidos: conceitos básicos e aplicações*. Edusp, 2000.

[7] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-order methods for incompressible fluid flow*, volume 9. Cambridge Univ Pr, 2002.

[8] Nina Edelweiss and Renata Galante. *Estrutura de Dados*. Bookman, 2009.

[9] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 1997.

[10] J.H. Ferziger, M. Peric, and K.W. Morton. *Computational methods for fluid dynamics*. Springer Berlin, 1999.

[11] B. Fornberg. *A practical guide to pseudospectral methods.* Cambridge university press, 1996.

[12] PL George and VN Kaliakin. Automatic Mesh Generation; Application to Finite Element Methods. *Journal of Engineering Mechanics*, 119:643, 1993.

[13] F. Gibou, C. Min, and H.D. Ceniceros. Non-Graded Adaptive Grid Approaches to the Incompressible Navier-Stokes Equations. *FDMP: Fluid Dynamics & Materials Processing*, 3(1):37–48, 2007.

[14] D.M. Greaves and A.G.L. Borthwick. On the use of adaptive hierarchical meshes for numerical simulation of separated flows. *International Journal for Numerical Methods in Fluids*, 26(3):303–322, 1998.

[15] DM Greaves and AGL Borthwick. Hierarchical tree-based finite element mesh generation. *International Journal for Numerical Methods in Engineering*, 45:447–471, 1999.

[16] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction.* Society for Industrial Mathematics, 1997.

[17] Michael Griebel, Stephan Knapek, and Gerhard Zumbusch. *Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications.* Springer, 2007.

[18] Dagoberto Adriano Rizzotto Justo. *Visual.* LICC, 1998.

[19] Dagoberto Adriano Rizzotto Justo. Geração de malhas, condições de contorno e discretização de operadores para dinâmica de fluidos computacional. Master's thesis, UFRGS, 2001.

[20] G. Karypis, K. Schloegel, and V. Kumar. Parmetis—parallel graph partitioning and sparse matrix ordering library, version 2.0. *Univ. of Minnesota, Minneapolis, MN*, 1998.

[21] P.M. Knupp and S. Steinberg. *The fundamentals of grid generation*. CRC press, 1993.

[22] M.C. Lai and C.S. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of Computational Physics*, 160(2):705–719, 2000.

[23] S. Li and W.K. Liu. *Meshfree particle methods*. Springer, 2004.

[24] G.R. Liu. *Mesh free methods: moving beyond the finite element method*. CRC press, 2003.

[25] GR Liu and YT Gu. An Introduction to Meshfree Methods and Their Programming. *Springer Dordrecht, 2005. 479*, 2005.

[26] C.A.S. Moser. Simulação numérica de esteiras em transição utilizando o método dos contornos virtuais. 2002.

[27] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572–600, 2003.

[28] Maliska C. R. *Tranferência de Calor e Dinâmica dos Fluidos Computacional*. Livro Técnico e Científico, 1995.

[29] B. Rogers, M. Fujihara, and A.G.L. Borthwick. Adaptive Q-tree Godunov-type scheme for shallow water equations. *International Journal for Numerical Methods in Fluids*, 35(3):247–280, 2001.

[30] EM Saiki and S. Biringen. Numerical simulation of a cylinder in uniform flow: application of a virtual boundary method. *Journal of Computational Physics*, 123(2):450–465, 1996.

[31] H. Samet. Neighbor Finding in Quadtrees. In *Proceedings-Institute of Electrical and Electronics Engineers Computer Society Conference on Pattern Recognition and Image Processing*, pages 68–74. IEEE Computer Society, 1981.

[32] Hanan Samet. *Apllications of Spatial Data Structures*. Addison-Wesley Publishing Company, 1989.

[33] J.F. Thompson, Z.U.A Warsi, and C.W. Mastin. *Numerical grid generation: foundations and applications*. North-holland Amsterdam, 1985.

[34] JP Wang, AGL Borthwick, and R.E. Taylor. Finite-volume-type VOF method on dynamically adaptive quadtree grids. *International Journal for Numerical Methods in Fluids*, 45(5):485–508, 2004.