

UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**DESENVOLVIMENTO DE INTERFACE GRÁFICA COMO SUPORTE  
PARA SOLUÇÕES NUMÉRICAS DAS EQUAÇÕES DE MAXWELL EM  
COORDENADAS GERAIS-3D**

**ADOLFO FRANCESCO DE OLIVEIRA COLARES**

DM-20/2011

UFPA / ITEC / PPGEE  
Campus Universitário do Guamá  
Belém – Pará – Brasil  
2011

UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**DESENVOLVIMENTO DE INTERFACE GRÁFICA COMO SUPORTE  
PARA SOLUÇÕES NUMÉRICAS DAS EQUAÇÕES DE MAXWELL EM  
COORDENADAS GERAIS-3D**

**ADOLFO FRANCESCO DE OLIVEIRA COLARES**

Dissertação de Mestrado  
Submetida à Banca Examinadora do  
Programa de Pós Graduação em  
Engenharia Elétrica da UFPA para  
obtenção do Grau de Mestre em  
Engenharia Elétrica. Área de  
Concentração: Computação Aplicada.

**Orientador: Prof. Dr. Rodrigo Melo  
e Silva de Oliveira**

UFPA / ITEC / PPGEE  
Campus Universitário do Guamá  
Belém – Pará – Brasil  
2011

---

C683d          Colares, Adolfo Francesco de Oliveira

Desenvolvimento de Interface gráfica como suporte para soluções numéricas das equações de Maxwell em coordenadas gerais – 3D / Adolfo Francesco de Oliveira Colares; orientador, Rodrigo Melo e Silva de Oliveira.-2011.

Dissertação (Mestrado) – Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-graduação em Engenharia Elétrica, Belém, 2011.

1. Interfaces gráficas de usuário (sistema de computador). 2. Maxwell, equações de. 3. Diferenças finitas. I. orientador. II. título.

CDD 22. ed. 005.71

---

UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**DESENVOLVIMENTO DE INTERFACE GRÁFICA COMO SUPORTE  
PARA SOLUÇÕES NUMÉRICAS DAS EQUAÇÕES DE MAXWELL EM  
COORDENADAS GERAIS-3D**

AUTOR: ADOLFO FRANCESCO DE OLIVEIRA COLARES

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM 03/05/2011

BANCA EXAMINADORA:

---

Prof. Dr. Rodrigo Melo e Silva de Oliveira  
(ORIENTADOR – PPGEE/UFPA)

---

Prof. Dr. Carlos Leonidas da Silva Souza Sobrinho  
(CO-ORIENTADOR – PPGEE/UFPA)

---

Prof. Dr. Licinius Dimitri Sá de Alcantara  
(MEMBRO – UFRA)

---

Prof. Dr. Eduardo Tannus Tuma  
(MEMBRO – FEE/UFPA)

VISTO:

---

Prof. Dr. Marcus Vinícius Alves Nunes  
(COORDENADOR DO PPGEE/UFPA)



*"As pessoas, de início, não seguem causas dignas. Seguem líderes dignos que promovem causas dignas".*

*James Clerk Maxwell.*

*(1831-1879)*

*A minha esposa Tiza, aos meus pais Manoel e Gilvanete e aos meus irmãos, com muito carinho.*

## Agradecimentos

A *Deus* por mais essa conquista, a minha esposa *Tiza Tamiozzo Quintas Colares* pelo carinho, incentivo, por estar presente em minha vida e me apoiar nas horas mais difíceis.

Aos meus pais *Manoel de Oliveira Colares* e *Gilvanete de Oliveira Colares* pela criação, conhecimento, educação e por proporcionarem todo o apoio para tornar este trabalho possível.

Aos meus irmãos *Luís André de Oliveira Colares*, *Manoel de Oliveira Colares Filho* e *Priscilla Lorena de Oliveira Colares Silva* por estarem presentes em minha vida e serem meus melhores amigos.

Um agradecimento especial ao meu orientador *Rodrigo M. S. de Oliveira*, pela ajuda nos momentos difíceis e pela grandeza ao contornar qualquer situação. Ao seu apoio e confiança, e sua imensa contribuição na conclusão deste trabalho.

Ao querido professor e coorientador *Carlos Leônidas Sobrinho*, obrigado pela recepção quando cheguei a UFPA e por ter acreditado em meu potencial. Lamento que o senhor esteja se aposentando, pois, outros discentes não terão o privilégio que tive ao trabalhar com o senhor em conjunto com o professor Rodrigo.

A todos os amigos do LANE, *Rodrigo Lisboa*, *Rodrigo Maia*, *Cláudio Gonçalves*, *Péricles Machado*, *Jonathas Felipe*, *Ramon Araújo* e um agradecimento em especial ao *Rodrigo Maia* que me ajudou bastante neste trabalho, muito obrigado.

A todos os professores no PPGEE/UFPA pelo conhecimento repassado nas aulas para a obtenção da formação em Mestre em Engenharia Elétrica.

À *Universidade Federal do Pará* pela infraestrutura disponibilizada para a realização deste trabalho.

A *CAPES* (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo incentivo financeiro com a bolsa de mestrado.

À comunidade do *Software Livre*, que desenvolveram ferramentas essenciais para a realização deste trabalho.

Aos *membros da banca examinadora e revisores*, pelas suas considerações essenciais para versão final deste trabalho.

## Lista de Símbolos

$\vec{E}$	Vetor Intensidade de Campo Elétrico
$\vec{H}$	Vetor Intensidade de Campo Magnético
$\vec{D}$	Vetor Densidade de Fluxo Elétrico
$\vec{B}$	Vetor Densidade de Fluxo Magnético
$\mu_0$	Permeabilidade Magnética do Vácuo
$\epsilon_0$	Permissividade Elétrica do Vácuo
$\epsilon$	Permissividade Elétrica
$\mu$	Permeabilidade Magnética
$\sigma$	Condutividade Elétrica
$\epsilon_r$	Permissividade Elétrica Relativa
$\mu_r$	Permeabilidade Magnética Relativa
$\sigma_\alpha$	Condutividade para UPML
$t$	Tempo
$x, y$ e $z$	Coordenadas do Sistema Cartesiano
$E_x, E_y$ e $E_z$	Componentes Cartesianos de $\vec{E}$
$H_x, H_y$ e $H_z$	Componentes Cartesianos de $\vec{H}$
$D_x, D_y$ e $D_z$	Componentes Cartesianos de $\vec{D}$
$B_x, B_y$ e $B_z$	Componente Cartesianos de $\vec{B}$
$\frac{df}{d\alpha}$	Derivada de $f$ em relação a $\alpha$
$\frac{\partial f}{\partial \alpha}$	Derivada Parcial de $f$ em relação a $\alpha$
$(i, j, k)$	Endereçamento de um ponto no Espaço Discretizado
$n$	Índice Temporal
$\Delta_t$	Incremento Temporal

$A_{\alpha}^n$	Componente $\alpha$ de $\vec{A}$ discretizada no instante $n$ , no ponto $(i, j, k)$
$\Delta_x, \Delta_y$ e $\Delta_z$	Incrementos espaciais relativos a $x, y$ e $z$
$\nabla \times \vec{A}$	Rotacional de $\vec{A}$
$\tau$	Coeficiente de Transmissão
$\Gamma$	Coeficiente de Reflexão
$A^i$	$i$ -ésima componente contravariante de $\vec{A}$
$A_i$	$i$ -ésima componente covariante de $\vec{A}$
$z_s$	Altura medida a partir da superfície do solo

# Sumário

<b>Capítulo 1 – Introdução .....</b>	<b>1</b>
1.1. Objetivos .....	2
1.2. Organização do Trabalho .....	3
<b>Capítulo 2 - Fundamentação Teórica .....</b>	<b>4</b>
2.1. O Sistema de Coordenadas Gerais .....	4
2.2. O método LN-FDTD aplicado para solucionar as Equações de Maxwell.....	7
2.3. A Estabilidade e Precisão do método LN-FDTD .....	9
2.4. UPML para meios condutivos – Truncamento do Método LN-FDTD.....	10
2.5. Processamento Paralelo para o método LN-FDTD e threads .....	18
2.5.1. <i>Threads</i> .....	19
2.5.2. Memória Compartilhada.....	20
2.6. Malhas Estruturadas e o método LN-FDTD .....	22
2.7. Modelos de processos para construção de <i>Softwares</i> .....	23
2.7.1. Modelo Sequencial Linear .....	23
2.7.2. Modelo Cascata .....	25
2.7.3. Modelo de Processo em Espiral.....	27
2.7.4. Modelo de Prototipagem.....	29
2.8. Cenário Gráfico Tridimensional .....	30
2.8.1. Visualização e criação tridimensional. ....	30
2.8.2. Sólidos .....	31
2.8.3. Câmera .....	32
2.8.4. Projeção .....	33
2.8.5. Modelagem de superfícies (visualização da malha).....	34
<b>Capítulo 3 - Desenvolvimento do Software .....</b>	<b>36</b>
3.1. Etapas de Desenvolvimento .....	36

3.1.1. Análise de Requisitos .....	38
3.1.2. Estudo da Tecnologia .....	39
3.1.3. Desenvolvimento do Protótipo .....	40
3.1.4. Testes e Validação .....	61
<b>Capítulo 4 - Resultados .....</b>	<b>66</b>
4.1. Validação do <i>Software</i> e Simulações .....	66
4.1.1 - Caso A .....	66
4.1.2 - Caso B .....	75
4.1.3 – Caso C.....	78
4.1.4 – Casos D e E.....	82
4.1.5 – Caso F .....	87
<b>Capítulo 5 - Considerações Finais.....</b>	<b>93</b>
<b>APÊNDICE A .....</b>	<b>95</b>
<b>APÊNDICE B.....</b>	<b>103</b>
<b>APÊNDICE C .....</b>	<b>107</b>
<b>APÊNDICE D .....</b>	<b>113</b>
<b>APÊNDICE E.....</b>	<b>119</b>
Referências Bibliográficas.....	141

## Lista de Figuras

Figura 2.1: Sistema de coordenadas curvilíneas no ponto $P$ e os vetores unitários em uma célula não ortogonal.....	5
Figura 2.2: Distribuição das componentes covariantes nas células não ortogonais Yee célula primária para as componentes covariantes do campo elétrico $(e_1, e_2, e_3)$ e célula secundária para as componentes covariantes do campo magnético $(h_1, h_2, h_3)$ ....	7
Figura 2.3: Decomposição do domínio em dois subdomínios e seção transversal (superfície 1-2) de uma malha 3D não-ortogonal. ....	19
Figura 2.4: Processo e estrutura de memória compartilhada e privada em uma máquina com seis <i>cores</i> .....	20
Figura 2.5: Código fonte didático: uso da <i>pthread</i> . ....	22
Figura 2.6: Exemplo de uma malha estruturada na visão do domínio físico e na visão do domínio computacional (como o algoritmo visualiza as malhas). ....	23
Figura 2.7: O modelo sequencial linear. ....	24
Figura 2.8: O modelo cascata. ....	26
Figura 2.9: Modelo de processo em espiral.....	27
Figura 2.10: Modelo de prototipagem. ....	29
Figura 2.11: O SRU em três dimensões (OpenGL).....	31
Figura 2.12: Modelo de câmera utilizada em OpenGL. ....	32
Figura 2.13: Projeção paralela ortográfica. ....	33
Figura 2.14: Projeção Perspectiva. ....	34
Figura 2.15: Cenário gráfico tridimensional, criado a partir da leitura de um arquivo de entrada.....	35
Figura 3.1: Etapas do Processo de Desenvolvimento.....	37



Figura 3.2: Conexão entre os módulos do software. ....	38
Figura 3.3: Qt <i>Designer</i> .....	41
Figura 3.4: Tela de abertura do <i>software</i> . ....	41
Figura 3.5: Interface Gráfica do <i>software</i> e suas áreas. ....	42
Figura 3.6: (a) Menu Principal; (b) Sobre. ....	43
Figura 3.7: Barra de Atalhos. ....	43
Figura 3.8: Guia do Método Numérico (LN-FDTD em Coordenadas Gerais). ....	44
Figura 3.9: (a) Área Parâmetros da Simulação e Estruturas; (b) Área Parâmetros da Simulação e Estruturas. ....	45
Figura 3.10: Aba Parâmetros da Simulação. ....	45
Figura 3.11: Aba Malha. ....	46
Figura 3.12: Aba Função Matemática (Fonte). ....	48
Figura 3.13: Aba Blocos Dielétricos. ....	49
Figura 3.14: Aba Blocos Metálicos. ....	50
Figura 3.15: Aba Hastes Metálicas. ....	50
Figura 3.16: Aba Planos Metálicos. ....	51
Figura 3.17: Aba Tensões. ....	51
Figura 3.18: Aba Correntes. ....	52
Figura 3.19: Escolha da Direção do Cálculo de Corrente. ....	52
Figura 3.20: (a) Aba Campo Elétrico; (b) Escolha do plano para imagem do campo. ....	53
Figura 3.21: Área Ações. ....	53
Figura 3.22: Área Resultados. ....	54
Figura 3.23: Visão geral do sistema desenvolvido. ....	54
Figura 3.24: Algoritmo do simulador LN-FDTD. ....	55
Figura 3.25: Glmesh visão da casca da malha. ....	57

Figura 3.26: (a) Demonstração da funcionalidade da tecla <i>m</i> ativada e (b) funcionalidades das teclas <i>i</i> e <i>Shift + i</i> .	58
Figura 3.27: Funcionalidades das tecla <i>s</i> associada à tecla <i>x</i> .	58
Figura 3.28: Funcionalidade da tecla <i>r</i> : (a) todas as linhas da malha foram escondidas; (b) visualização restaurada da malha.	59
Figura 3.29: Conjunto de Hastes Metálicas.	60
Figura 3.30: Bloco Metálico e Blocos Dielétricos.	61
Figura 3.31: Planos Metálicos interligados por uma Haste Metálica.	61
Figura 3.32: Comparações da resposta transiente obtida através do método LN-FDTD, para um eletrodo vertical medindo 0,5×0,5×3,0 m, com os dados obtidos em [22].	62
Figura 3.33: Problema proposto para validação do <i>software</i> desenvolvido.	64
Figura 3.34: Secção transversal (superfície 1-2) da malha não-ortogonal gerada para modelar as hastes.	64
Figura 3.35: Tensão $V(t)$ : métodos FDTD (fio fino) e LN-FDTD.	65
Figura 4.1: Estrutura da residência modelada na GUI LN-FDTD em Coordenadas Gerais 3D (visão sem o teto e o solo).	68
Figura 4.2: Visão superior da residência para o Caso A.	68
Figura 4.3: Visão lateral da residência para o Caso A.	69
Figura 4.4: Visão frontal da residência para o Caso A.	69
Figura 4.5: (a) Tensão $V_1$ calculada na tomada; (b) Caminho de integração numérica utilizado no cálculo da tensão induzida na linha de distribuição (linha pontilhada).	71
Figura 4.6: (a) Tensão induzida entre o solo e a fase; (b) Tensão induzida entre o solo e o neutro (Caso A).	72
Figura 4.7: (a) Tensão induzida na tomada 1; (b) Tensão induzida na tomada 2 (Caso A).	72
Figura 4.8: (a) Tensão induzida na tomada 3; (b) Tensão induzida na tomada 4 (Caso A).	73

Figura 4.9: (a) Tensão induzida na tomada 5; (b) Tensão induzida na tomada 6 (Caso A).....	73
Figura 4.10: (a) Tensão induzida na tomada 7; (b) Tensão induzida na tomada 8 (Caso A)....	74
Figura 4.11: Distribuição espacial de $ E $ para $z = 2,88\text{m}$ ( $k=72$ ) e (a) $t = 1\mu\text{s}$ ; (b) $t = 2\mu\text{s}$ ; (c) $t = 3\mu\text{s}$ . .....	74
Figura 4.12: Distribuição espacial de $ E $ para $y = 3,60\text{m}$ ( $j=90$ ) e (a) $t = 1\mu\text{s}$ ; (b) $t = 2\mu\text{s}$ ; (c) $t = 3\mu\text{s}$ . .....	75
Figura 4.13: Visualização tridimensional e definição do Caso B.....	76
Figura 4.14: Tensão induzida entre solo e fase: Caso B.....	76
Figura 4.15: Tensão obtida entre solo e neutro (não aterrado): Caso B. ....	77
Figura 4.16: Tensão induzida entre os terminais da tomada 1: Caso B.....	77
Figura 4.17: Nova estrutura de pára-raios para o Caso C.....	79
Figura 4.18: Distribuição espacial de $ E $ para $x = 7,96\text{m}$ ( $i=199$ ) após (a) $0.333\mu\text{s}$ ; (b) $0.667\mu\text{s}$ ; (c) $1\mu\text{s}$ para o Caso C.....	79
Figura 4.19: Tensões induzidas para os Casos A, B e C para (a) Tomada 1; (b) Fase; (c) Neutro. ....	81
Figura 4.20: Visão frontal com a representação da catenária na linha de distribuição.....	83
Figura 4.21: Corte transversal da malha desenvolvida para a simulação. Destacam-se as linhas curvadas. ....	83
Figura 4.22: Estrutura metálica da residência e catenárias relativas às linhas de distribuição: visão em 3D sem os blocos dielétricos.....	84
Figura 4.23: Distribuição da componente $e_2$ do campo elétrico para $y=1,32\text{ m}$ , $t = 0,02474526\mu\text{s}$ para (a) $\Delta f = 0,20\text{m}$ e (b) $\Delta f = 0,30\text{m}$ . ....	84
Figura 4.24: Tensões induzidas para os casos D e E, e dos casos anteriores para (a) Fase; (b) Neutro; (c) Tomada 1.....	87
Figura 4.25: Definição do posicionamento das linhas para o Caso F.....	88

Figura 4.26: Tensões induzidas do Caso F para (a) Tomada 1; (b) Fase; (c) Neutro. ....	90
Figura 4.27: Tensões induzidas do Caso F e dos casos anteriores para (a) Tomada 1; (b) Fase; (c) Neutro.....	91

## RESUMO

Neste trabalho, é implementada uma interface gráfica de usuários (GUI) usando a ferramenta Qt da Nokia (versão 3.0). A interface visa simplificar a criação de cenários para a realização de simulações paralelas usando a técnica numérica Local Nonorthogonal Finite Difference Time-Domain (LN-FDTD), aplicada para solucionar as equações de Maxwell. O simulador foi desenvolvido usando a linguagem de programação C e paralelizado utilizando *threads*. Para isto, a biblioteca *pthread* foi empregada. A visualização 3D do cenário a ser simulado (e da malha) é realizada por um programa especialmente desenvolvido que utiliza a biblioteca *OpenGL*. Para melhorar o desenvolvimento e alcançar os objetivos do projeto computacional, foram utilizados conceitos da Engenharia de *Software*, tais como o modelo de processo de *software* por prototipagem. Ao privar o usuário de interagir diretamente com o código-fonte da simulação, a probabilidade de ocorrência de erros humanos durante o processo de construção de cenários é minimizada. Para demonstrar o funcionamento da ferramenta desenvolvida, foi realizado um estudo relativo ao efeito de flechas em linhas de baixa tensão nas tensões transitórias induzidas nas mesmas por descargas atmosféricas. As tensões induzidas nas tomadas da edificação também são estudadas.

**Palavras-chave:** engenharia de *software*, interface gráfica de usuário, Método FDTD, Método LN-FDTD, Método LN-UPML, processamento paralelo, sistemas distribuídos, *thread*, visualização 3D, *OpenGL*, flechas, linhas de distribuição, descargas atmosféricas, tensões induzidas.

## ABSTRACT

In this work, we have implemented a graphical user interface (GUI) by using the Nokia Qt library (version 3.0). The interface is designed to simplify the creation of scenarios for executing parallel E.M. Simulations by using the numerical technique Local Non-Orthogonal Finite Difference Time-Domain (LN-FDTD) method, applied to solve Maxwell's equations. The simulator was developed by using the C programming language and parallelized by using threads. This way, the pthread library was employed. The 3D visualization of the scenario and of the corresponding mesh to be simulated is performed by a specially developed program based on the OpenGL specification. In order to improve the development and to achieve the goals of computational design, we have used concepts of software engineering, such as the process model for software prototyping. Depriving the user to interact directly with the source code of the simulation program, the probability of human errors while performing the constructing process of scenarios is minimized. In order to demonstrate the operation of the developed tool, a study regarding lightning-induced voltages on low voltage lines with catenaries is performed. Induced voltages inside a small building (a residence) are also studied.

**Keywords:** software engineering, graphical user interface, FDTD Method, LN-FDTD Method, LN-UPML Method, parallel processing, distributed systems, thread, 3D visualization, OpenGL, catenaries, distribution lines, lightning strikes, induced voltages.

# Capítulo 1 – Introdução

Problemas relacionados ao eletromagnetismo podem ser solucionados utilizando diversos métodos, classificados como: analíticos, experimentais e numéricos [1]. Dentre estes grupos, destacam-se os métodos numéricos por, em geral, apresentarem soluções mais flexíveis em termos de geometria e de parâmetros para solucionar as equações de Maxwell [2,3].

Dentre os métodos numéricos existentes para a solução de problemas em eletromagnetismo, podem ser mencionados o FDTD (*Finite-Difference Time-Domain*) [3], MoM (*Method of Moments*) [2], FEM (*Finite Element Method*) [4] e finalmente o método LN-FDTD (*Local Nonorthogonal Finite-Difference Time-Domain*) [3][5], no qual este trabalho se baseia.

O método FDTD foi publicado por Yee em 1966 e é fundamentado na aproximação das equações diferenciais escalares das componentes dos campos elétrico e magnético, no sistema Cartesiano, por um grupo de equações de diferenças finitas (aproximando as derivadas por diferenças centradas) [5]. O método LN-FDTD é uma adaptação do método FDTD, no qual malhas estruturadas com células de arestas não ortogonais entre si são utilizadas de forma a modelar geometrias não compatíveis com o sistema Cartesiano de coordenadas, evitando aproximações do tipo *staircase* [3]. Assim como o FDTD clássico, o método LN-FDTD gera soluções de onda completa, sendo que a sua robustez e estabilidade estão diretamente ligadas à malha utilizada para representar o problema geometricamente [5].

Vale ressaltar que existem técnicas de interface gráfica de usuário (GUI) e de visualização tridimensional para a maioria dos métodos numéricos mencionados anteriormente, tais como a ferramenta FEMLAB [6], que utiliza um simulador baseado na técnica FEM. Em [7], é apresentada a ferramenta *Synthesis and Analysis of Grounding Systems* (SAGS) baseada na técnica FDTD, em coordenadas retangulares. Em [8], encontra-se descrita a ferramenta HARP baseada na técnica MoM e, até então, não foi encontrada qualquer técnica de GUI ou visualização tridimensional desenvolvida para um simulador baseado na técnica LN-FDTD, o que constitui o objetivo deste trabalho.

Para a implementação do *software* proposto neste trabalho, foram utilizados modelos e

padrões para a criação da GUI descritos em [9], assim como, teste para GUI baseados nos testes desenvolvidos por [10] e um modelo de processos de desenvolvimento de *software* por prototipagem descrito em [11].

Neste trabalho, foi desenvolvida uma interface gráfica para usuários utilizando a ferramenta Qt 3 [12]. Esta interface tem como objetivo dar suporte (entrada e saída de informação) para simulações paralelizadas ou sequenciais de problemas de eletromagnetismo aplicado. Na formulação do problema, as equações de Maxwell são solucionadas numericamente através do método LN-FDTD, em Coordenadas Gerais (3D).

Também foi implementado um visualizador de cenários e de malhas não ortogonais tridimensionais com os recursos da biblioteca OpenGL [13], o que certamente fornece aos usuários uma visualização adequada do problema em questão. A ferramenta desenvolvida e suas funções seguem o padrão *Unix* e foram implementadas e testadas em ambiente Linux com KDE 3.5 e Qt 3.3. O objetivo principal da ferramenta é proporcionar ao usuário transparência no que está sendo modelado no simulador e, conseqüentemente, afastar o engenheiro do desenvolvimento de códigos computacionais complexos, reduzindo as possibilidades de erro humano durante a fase de concepção das estruturas. Vale ressaltar que o ambiente computacional desenvolvido pode ser aplicado na solução de problemas 3D independentemente de haver ou não simetria, oferecendo saídas numéricas em forma de texto (.*dat*s), gráficas ou em forma de filmes (evolução temporal do processo eletromagnético).

## 1.1. Objetivos

O objetivo deste trabalho é a criação de uma interface gráfica de usuário (GUI), um simulador baseado no método numérico LN-FDTD (Local Nonorthogonal Finite-Difference Time-Domain), e um visualizador de cenários e de malhas discretizadas 3D. A ideia é simplificar a criação de cenários para a realização de simulações paralelas ou sequenciais e, ao mesmo tempo, privar o usuário de interagir diretamente com o código-fonte da simulação.

É também, objetivo do trabalho realizar a análise de tensões induzidas em linhas de distribuição, provenientes de descargas atmosféricas, considerando-se estruturas retangulares e não retangulares, além de paralelizar o método LN-FDTD utilizando *threads*.

O *software* desenvolvido é utilizado para realizar um estudo relativo a tensões



induzidas em linhas de distribuição, provocadas por descargas atmosféricas, quando as mesmas estão conectadas eletricamente a uma residência. As tensões induzidas nas tomadas da edificação também são calculadas.

## **1.2. Organização do Trabalho**

No Capítulo 2, é feita uma abordagem teórica acerca do trabalho realizado em que, inicialmente, é apresentado o sistema de coordenadas gerais, o método LN-FDTD e sua formulação matemática aplicada para solucionar numericamente as equações de Maxwell. Logo em seguida, é abordada a truncagem do método LN-FDTD e a utilização do processamento paralelo. Em seguida, são mostradas então as técnicas e modelos de processos que a Engenharia de *Software* oferece para a realização de um projeto de construção de *software*. Por fim, são descritos o conceito, métodos e funções de visualização tridimensional utilizando a biblioteca OpenGL.

No Capítulo 3, aborda-se o processo de desenvolvimento da GUI, do simulador numérico LN-FDTD e do visualizador de malhas e estruturas tridimensionais. Em seguida, são explicadas as funcionalidades da interface e são feitas demonstrações do visualizador tridimensional e de suas funcionalidades para a manipulação do cenário 3D a ser simulado.

No Capítulo 4, são mostrados e discutidos os resultados obtidos. Por fim, as considerações finais são feitas no Capítulo 5.

## Capítulo 2 - Fundamentação Teórica

### 2.1. O Sistema de Coordenadas Gerais

Considerando um sistema de coordenadas gerais ou curvilíneas, o vetor posição  $\vec{r}$ , relativo a um ponto  $P$  (Figura 2.1), pode ser obtido em um sistema de coordenadas gerais, tal que um vetor de comprimento diferencial  $d\vec{r}$  é dado por (2.1) [3],

$$d\vec{r} = \sum_{l=1}^3 \frac{\partial \vec{r}}{\partial u^l} du^l = \sum_{l=1}^3 \vec{a}_l du^l, \quad (2.1)$$

em que os vetores  $\vec{a}_l$  são chamados de vetores unitários e formam uma base unitária, que define os eixos curvilíneos gerais que passam por um ponto  $P$  no espaço (cada ponto tem seu próprio sistema de coordenadas). Na Figura 2.1, estão representados os vetores  $\vec{a}_l$  ( $l = 1, 2, 3$ ) que são tangentes aos eixos  $u^l$  em  $P$  e podem ser escritos como funções das coordenadas cartesianas  $x$ ,  $y$  e  $z$ . Um conjunto alternativo de três vetores complementares  $\vec{a}^l$ , denominados de vetores recíprocos, pode ser definido de modo que cada um é normal a dois vetores unitários com diferentes índices [5], formando assim uma base recíproca. Esse conjunto pode ser matematicamente calculado por

$$\vec{a}^1 = \frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}}, \quad (2.2a)$$

$$\vec{a}^2 = \frac{\vec{a}_1 \times \vec{a}_3}{\sqrt{g}}, \quad (2.2b)$$

$$\vec{a}^3 = \frac{\vec{a}_1 \times \vec{a}_2}{\sqrt{g}}, \quad (2.2c)$$

em que  $\sqrt{g}$  é o volume do hexaedro formado pelos vetores  $\vec{a}_1, \vec{a}_2, \vec{a}_3$  (Figura 2.1). Em (2.2),  $g$  é o determinante da matriz métrica ou o tensor covariante [5].

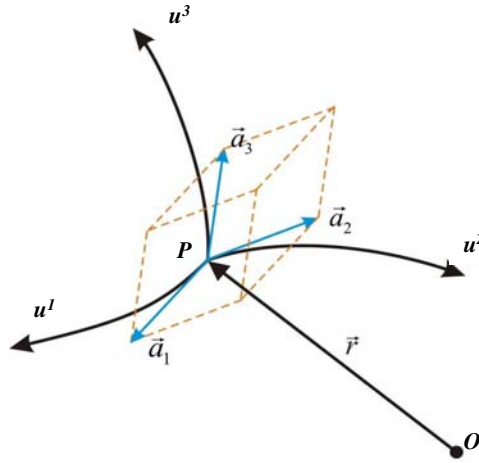


Figura 2.1: Sistema de coordenadas curvilíneas no ponto  $P$  e os vetores unitários em uma célula não ortogonal.

Com base na ideia apresentada anteriormente, um vetor  $\vec{F}$  pode ser representado por

$$\vec{F} = \sum_{l=1}^3 f^l \vec{a}_l = \sum_{l=1}^3 f_l \vec{a}^l, \quad (2.3)$$

na qual,  $f_l$  e  $f^l$  são chamadas de componentes covariantes e contravariantes do vetor  $\vec{F}$ , respectivamente. As componentes  $f^l$  e  $f_l$  podem ser calculadas por meio do produto escalar de  $\vec{F}$ , dado por (2.3), por  $\vec{a}^l$  e  $\vec{a}_l$  respectivamente. Assim, tem-se

$$f^l = \vec{F} \cdot \vec{a}^l, \quad (2.4a)$$

$$f_l = \vec{F} \cdot \vec{a}_l. \quad (2.4b)$$

Estas componentes (covariante e contravariante) podem ser relacionadas por meio das expressões

$$f_m = \sum_{l=1}^3 g_{lm} f^l \quad (2.5a)$$

e

$$f^m = \sum_{l=1}^3 g^{lm} f_l, \quad (2.5b)$$

nas quais  $g_{lm} = \vec{a}_l \cdot \vec{a}_m$  e  $g^{lm} = \vec{a}^l \cdot \vec{a}^m$ . É importante ressaltar que os vetores unitários e os recíprocos não têm necessariamente amplitudes unitárias porque dependem da natureza do sistema de coordenadas curvilínea (comprimento das arestas das células). Neste caso, hexaedros similares aos da Figura 2.1 são usados como células de Yee. Dessa forma, um conjunto apropriado de vetores unitários e seus respectivos comprimentos unitários são definidos pelas equações elementares

$$\vec{i}_1 = \frac{\vec{a}_1}{\sqrt{\vec{a}_1 \cdot \vec{a}_1}} = \frac{\vec{a}_1}{\sqrt{g_{11}}}, \quad (2.6a)$$

$$\vec{i}_2 = \frac{\vec{a}_2}{\sqrt{g_{22}}}, \quad (2.6b)$$

e

$$\vec{i}_3 = \frac{\vec{a}_3}{\sqrt{g_{33}}}. \quad (2.6c)$$

A partir de (2.3) e (2.6), pode-se escrever

$$\vec{F} = F^1 \vec{i}_1 + F^2 \vec{i}_2 + F^3 \vec{i}_3, \quad (2.7)$$

em que  $F^l$  representa o valor físico da  $l$ -ésima componente contravariante de  $\vec{F}$  no sistema de base, e é dada por

$$F^l = f^l \sqrt{g_{ll}}. \quad (2.8a)$$

De forma análoga, a  $l$ -ésima componente covariante de  $\vec{F}$  é dada por

$$F_l = f_l \sqrt{g^{ll}}. \quad (2.8b)$$

Essa representação pode ser utilizada para descrever as componentes dos campos elétrico e magnético, como será visto a seguir.

## 2.2. O método LN-FDTD aplicado para solucionar as Equações de Maxwell

As equações de Maxwell na forma diferencial no domínio do tempo para meios com perdas, isotrópicos e não dispersivos, são dadas por

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t}, \quad (2.9a)$$

e

$$\nabla \times \vec{H} = \sigma \vec{E} + \varepsilon \frac{\partial \vec{E}}{\partial t}, \quad (2.9b)$$

em que  $\vec{E}$  é o vetor intensidade de campo elétrico,  $\vec{H}$  é o vetor intensidade de campo magnético,  $\varepsilon$  é a permissividade elétrica,  $\mu$  é a permeabilidade magnética e  $\sigma$  é a condutividade elétrica. Onde os parâmetros ( $\varepsilon$ ,  $\sigma$  e  $\mu$ ) caracterizam o meio. Calculando as componentes contravariantes dos campos  $\vec{E}$  e  $\vec{H}$ , utilizando diferenças centradas [3] para os termos derivativos e observando as células primárias e secundárias na Figura 2.1, podem ser obtidas as equações para atualização das componentes de  $\vec{E}$  e  $\vec{H}$ , de forma similar ao algoritmo de Yee original.

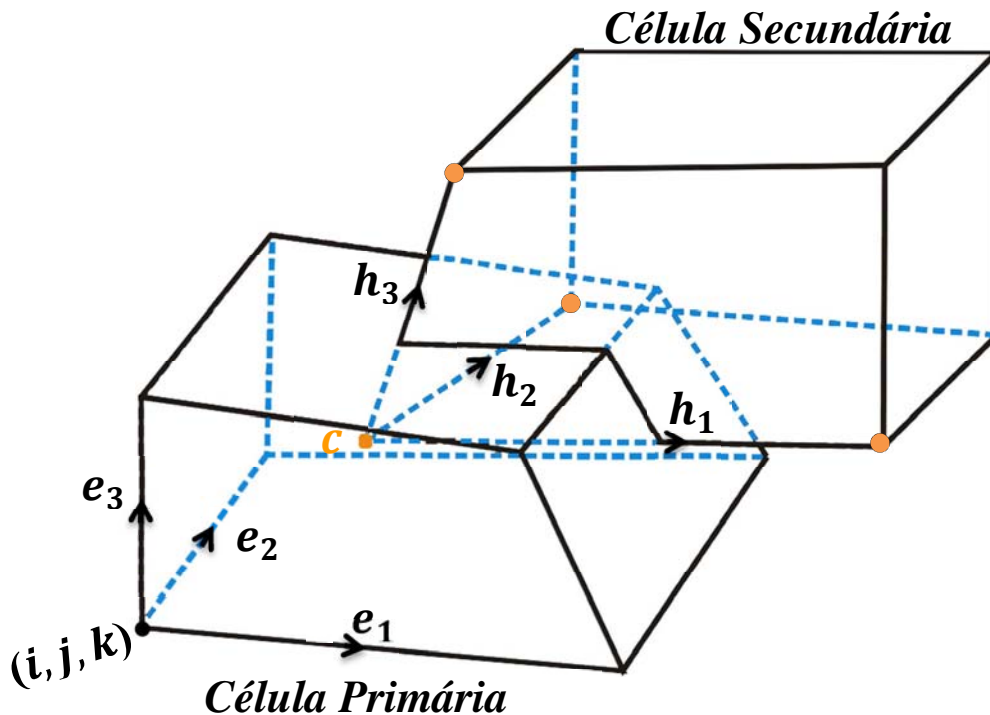


Figura 2.2: Distribuição das componentes covariantes nas células não ortogonais Yee célula primária para as componentes covariantes do campo elétrico ( $e_1, e_2, e_3$ ) e célula secundária para as componentes covariantes do campo magnético ( $h_1, h_2, h_3$ ).

Para obtenção de uma componente de campo, por exemplo, a primeira componente contravariante de  $\vec{H}$ , (2.4) e (2.2) podem ser usadas com  $l = 1$ . Nesse caso, a operação produto escalar é realizada em (2.9a) (lei de Faraday). Assim,

$$(\nabla \times \vec{E}) \cdot \frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}} = -\mu \left( \frac{\partial \vec{H}}{\partial t} \right) \cdot \vec{a}^1, \quad (2.10)$$

a partir da qual é fácil ver que

$$\frac{\partial H^1}{\partial t} = -\frac{1}{\mu\sqrt{g}} \left( \frac{\partial e_3}{\partial u^2} - \frac{\partial e_2}{\partial u^3} \right). \quad (2.11a)$$

Realizando o mesmo procedimento para  $e^1$ , chega-se a (2.11b) [14]

$$\epsilon \frac{\partial e^1}{\partial t} + \sigma e^1 = \left( \frac{\partial h_3}{\partial u^2} - \frac{\partial h_2}{\partial u^3} \right) / \sqrt{g}. \quad (2.11b)$$

Vale ressaltar que a partir de (2.2), é possível dizer que  $\vec{a}^m \cdot \vec{a}_n = \delta_{m,n}$ , em que  $\delta_{m,n}$  é o delta de Kronecker. As expressões (2.11a) e (2.11b) e as equações para as outras cinco componentes de campo podem ser representadas por diferenças finitas utilizando o algoritmo de Yee [5] (com derivadas centradas).

As expressões (2.11a) e (2.11b) e as equações para as outras cinco componentes de campo representadas por diferenças finitas podem ser aproximadas de forma usual por diferenças centradas. Dessa forma, as seguintes as seguintes equações de atualização são obtidas

$$h_{(i,j,k)}^{1(n+\frac{1}{2})} = h_{(i,j,k)}^{1(n-\frac{1}{2})} - \frac{\Delta t}{\mu\sqrt{g}} \left[ \frac{e_{3(i,j+1,k)}^{(n)} - e_{3(i,j,k)}^{(n)}}{\Delta u^2} - \frac{e_{2(i,j,k+1)}^{(n)} - e_{2(i,j,k)}^{(n)}}{\Delta u^3} \right], \quad (2.11c)$$

$$h_{(i,j,k)}^{2(n+\frac{1}{2})} = h_{(i,j,k)}^{2(n-\frac{1}{2})} - \frac{\Delta_t}{\mu\sqrt{g}} \left[ \frac{e_{1(i,j,k+1)}^{(n)} - e_{1(i,j,k)}^{(n)}}{\Delta u^3} - \frac{e_{3(i+1,j,k)}^{(n)} - e_{3(i,j,k)}^{(n)}}{\Delta u^1} \right], \quad (2.11d)$$

$$h_{(i,j,k)}^{3(n+\frac{1}{2})} = h_{(i,j,k)}^{3(n-\frac{1}{2})} - \frac{\Delta_t}{\mu\sqrt{g}} \left[ \frac{e_{2(i+1,j,k)}^{(n)} - e_{2(i,j,k)}^{(n)}}{\Delta u^1} - \frac{e_{1(i,j+1,k)}^{(n)} - e_{1(i,j,k)}^{(n)}}{\Delta u^2} \right], \quad (2.11e)$$

$$e_{(i,j,k)}^{1(n+1)} = e_{(i,j,k)}^{1(n)} \left( \frac{1 - \sigma \frac{\Delta_t}{2\epsilon}}{1 + \sigma \frac{\Delta_t}{2\epsilon}} \right) + \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{3(i,j,k)}^{1(n+\frac{1}{2})} - h_{3(i,j-1,k)}^{1(n+\frac{1}{2})}}{\Delta u^2} \right] - \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{2(i,j,k)}^{1(n+\frac{1}{2})} - h_{2(i,j,k-1)}^{1(n+\frac{1}{2})}}{\Delta u^3} \right], \quad (2.11f)$$

$$e_{(i,j,k)}^{2(n+1)} = e_{(i,j,k)}^{2(n)} \left( \frac{1 - \sigma \frac{\Delta_t}{2\epsilon}}{1 + \sigma \frac{\Delta_t}{2\epsilon}} \right) + \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{1(i,j,k)}^{1(n+\frac{1}{2})} - h_{1(i,j,k-1)}^{1(n+\frac{1}{2})}}{\Delta u^3} \right] - \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{3(i,j,k)}^{1(n+\frac{1}{2})} - h_{3(i-1,j,k)}^{1(n+\frac{1}{2})}}{\Delta u^1} \right], \quad (2.11g)$$

e

$$e_{(i,j,k)}^{3(n+1)} = e_{(i,j,k)}^{3(n)} \left( \frac{1 - \sigma \frac{\Delta_t}{2\epsilon}}{1 + \sigma \frac{\Delta_t}{2\epsilon}} \right) + \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{2(i,j,k)}^{1(n+\frac{1}{2})} - h_{2(i-1,j,k)}^{1(n+\frac{1}{2})}}{\Delta u^1} \right] - \frac{\Delta_t}{(\epsilon + \frac{1}{2}\Delta_t\sigma)\sqrt{g}} \left[ \frac{h_{1(i,j,k)}^{1(n+\frac{1}{2})} - h_{1(i,j-1,k)}^{1(n+\frac{1}{2})}}{\Delta u^2} \right] \quad (2.11h)$$

### 2.3. A Estabilidade e Precisão do método LN-FDTD

A estabilidade numérica do método está relacionada ao incremento  $\Delta t$ , que segue as condições dadas pela expressão (2.12) [3], em que  $V_m$  é a máxima velocidade de propagação da onda no domínio de análise.

$$\Delta t \leq \frac{1}{V_m \left( \sqrt{\sum_{l=1}^3 \sum_{m=1}^3 g^{lm}} \right)}. \quad (2.12)$$

A expressão (2.12) é conhecida como condição de Courant para o método LN-FDTD [3].

Para reduzir os efeitos da dispersão numérica para níveis adequados e para garantir a precisão dos cálculos, todas as arestas de cada célula não ortogonal devem ser menores que  $\lambda/10$  [3], onde  $\lambda$  é o menor comprimento de onda envolvido no problema.

## 2.4. UPML para meios condutivos – Truncamento do Método LN-FDTD

Para solucionar problemas abertos, há a necessidade de se limitar a região de análise. Nesta situação, a técnica de camadas perfeitamente casadas uniaxiais (UPML) [5] é utilizada com a função de trincar o domínio computacional. A truncagem consiste em absorver as ondas que chegam aos limites da região de análise de maneira a simular sua propagação ao infinito [14].

A formulação matemática da UPML usada no método LN-FDTD deste trabalho, é fundamentada por [14] e é descrita abaixo.

As equações de Maxwell no domínio da frequência, para um meio uniaxialmente anisotrópico são dadas por (2.13)

$$\nabla \times \vec{E} = -j\omega\mu\bar{S}\vec{H}, \quad (2.13a)$$

$$\nabla \times \vec{H} = (j\omega\varepsilon + \sigma)\bar{S}\vec{E}, \quad (2.13b)$$

em que  $\omega$  define a frequência angular da onda eletromagnética,  $\vec{E}$  e  $\vec{H}$  são as transformadas de Fourier do vetor intensidade de campo elétrico  $\vec{E}$  e do campo magnético  $\vec{H}$ , respectivamente,  $\mu = \mu_0\mu_r$  é a permeabilidade magnética e  $\varepsilon = \varepsilon_0\varepsilon_r$  é a permissividade elétrica,  $\bar{S}$  é o tensor que promove a atenuação na região absorvente e a anisotropia do meio e  $\sigma$  é a condutividade



elétrica do meio. O tensor  $\bar{\bar{S}}$  tem a forma [3] [5]

$$\bar{\bar{S}} = \begin{bmatrix} \frac{S_2 \cdot S_3}{S_1} & 0 & 0 \\ 0 & \frac{S_1 \cdot S_3}{S_2} & 0 \\ 0 & 0 & \frac{S_1 \cdot S_2}{S_3} \end{bmatrix}, \quad (2.14)$$

na qual  $S_\alpha$  ( $\alpha = 1, 2, 3$ ) são os parâmetros que caracterizam a atenuação na UPML, e são dados por [5]

$$S_\alpha = K_\alpha + \frac{\sigma_\alpha}{j\omega\epsilon_0}. \quad (2.15)$$

Desenvolvendo a expressão (2.13b) em coordenadas gerais e utilizando a expressão (2.14), tem-se

$$\frac{1}{\sqrt{g}} \begin{bmatrix} \partial \mathbf{h}_3 / \partial u^2 - \partial \mathbf{h}_2 / \partial u^3 \\ \partial \mathbf{h}_1 / \partial u^3 - \partial \mathbf{h}_3 / \partial u^1 \\ \partial \mathbf{h}_2 / \partial u^1 - \partial \mathbf{h}_1 / \partial u^2 \end{bmatrix} = j\omega\epsilon_0 \left( \epsilon_r + \frac{\sigma}{j\omega\epsilon_0} \right) \begin{bmatrix} \frac{S_2 S_3}{S_1} & 0 & 0 \\ 0 & \frac{S_1 S_3}{S_2} & 0 \\ 0 & 0 & \frac{S_1 S_2}{S_3} \end{bmatrix} \begin{bmatrix} \mathbf{e}^1 \\ \mathbf{e}^2 \\ \mathbf{e}^3 \end{bmatrix}. \quad (2.16)$$

Então, tem-se para  $\mathbf{e}^1$

$$\frac{1}{\sqrt{g}} \left( \frac{\partial \mathbf{h}_3}{\partial u^2} - \frac{\partial \mathbf{h}_2}{\partial u^3} \right) = j\omega\epsilon_0 \left( \epsilon_r + \frac{\sigma}{j\omega\epsilon_0} \right) \left( \frac{S_2 S_3}{S_1} \right) \mathbf{e}^1.$$

A continuidade das componentes é garantida utilizando a normalização

$$\vec{\mathbf{v}} = \vec{\mathbf{v}} \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}^{-1}. \quad (2.17)$$

Com isso, chega-se a

$$\frac{1}{\sqrt{g}} (s_3 (\mathbf{h}_{3(i,j,k)} - \mathbf{h}_{3(i,j-1,k)}) - s_2 (\mathbf{h}_{2(i,j,k)} - \mathbf{h}_{2(i,j,k-1)})) = j\omega \varepsilon_0 \left( \varepsilon_r + \frac{\sigma}{j\omega \varepsilon_0} \right) (s_2 s_3) \mathbf{e}_{(i,j,k)}^1.$$

Dividindo a expressão acima por  $s_2$  e ajustando o lado direito, tem-se

$$\frac{1}{\sqrt{g}} \left( \frac{s_3}{s_2} (\mathbf{h}_{3(i,j,k)} - \mathbf{h}_{3(i,j-1,k)}) - (\mathbf{h}_{2(i,j,k)} - \mathbf{h}_{2(i,j,k-1)}) \right) = (j\omega \varepsilon + \sigma) s_3 \mathbf{e}_{(i,j,k)}^1. \quad (2.18)$$

Na expressão (2.15), que contém o termo  $j\omega$ , é fácil notar que os termos  $\frac{s_3}{s_2} (\mathbf{h}_{3(i,j,k)} - \mathbf{h}_{3(i,j-1,k)})$  e  $s_3 \mathbf{e}_{(i,j,k)}^1$  devem ser tratados de forma particular, para evitar convoluções computacionalmente intensivas, quando utilizado o domínio do tempo. Com isso, definem-se

$$\tilde{\mathbf{h}}_{3(i,j,k)} = \frac{s_3}{s_2} (\mathbf{h}_{3(i,j,k)} - \mathbf{h}_{3(i,j-1,k)}) = \left( \frac{k_3 + \frac{\sigma_3}{j\omega \varepsilon_0}}{k_2 + \frac{\sigma_2}{j\omega \varepsilon_0}} \right) \bar{\mathbf{h}}_{3(i,j,k)} \quad (2.19)$$

e

$$\mathbf{P}_{(i,j,k)}^1 = s_3 \mathbf{e}_{(i,j,k)}^1 = \left( k_3 + \frac{\sigma_3}{j\omega \varepsilon_0} \right) \mathbf{e}_{(i,j,k)}^1. \quad (2.20)$$

Portanto, a expressão (2.18) pode ser reescrita como:

$$\frac{1}{\sqrt{g}} (\tilde{\mathbf{h}}_{3(i,j,k)} - (\mathbf{h}_{2(i,j,k)} - \mathbf{h}_{2(i,j,k-1)})) = (j\omega \varepsilon + \sigma) \mathbf{P}_{(i,j,k)}^1 \quad (2.21)$$

Em (2.19), observa-se que é possível obter uma equação de atualização explícita para  $\tilde{h}_{3(i,j,k)}$ , no domínio do tempo, em função de  $\bar{h}_{3(i,j,k)} = h_{3(i,j,k)} - h_{3(i,j-1,k)}$ . Em (2.19), nota-se que

$$j\omega k_2 \tilde{\mathbf{h}}_3 + \frac{\sigma_2}{\varepsilon_0} \tilde{\mathbf{h}}_3 = j\omega k_3 \bar{\mathbf{h}}_3 + \frac{\sigma_3}{\varepsilon_0} \bar{\mathbf{h}}_3.$$

Passando a expressão acima para o domínio do tempo, empregando a transformação  $j\omega \rightarrow \partial / \partial t$ , tem-se

$$k_2 \frac{\partial \tilde{h}_3}{\partial t} + \frac{\sigma_2}{\varepsilon_0} \tilde{h}_3 = k_3 \frac{\partial \bar{h}_3}{\partial t} + \frac{\sigma_3}{\varepsilon_0} \bar{h}_3.$$

Usando diferenças centradas para aproximar as derivadas temporais e média temporal para as parcelas sem derivadas, tem-se a seguinte expressão de atualização para  $\tilde{h}_{3(i,j,k)}^{n+1}$ :

$$\tilde{h}_{3(i,j,k)}^{n+1} = \frac{\left(\frac{k_2}{\Delta_t} - \frac{\sigma_2}{2\varepsilon_0}\right) \tilde{h}_{3(i,j,k)}^n + \left(\frac{k_3}{\Delta_t} + \frac{\sigma_3}{2\varepsilon_0}\right) \bar{h}_{3(i,j,k)}^{n+1} - \left(\frac{k_3}{\Delta_t} - \frac{\sigma_3}{2\varepsilon_0}\right) \bar{h}_{3(i,j,k)}^n}{\left(\frac{k_2}{\Delta_t} + \frac{\sigma_2}{2\varepsilon_0}\right)}. \quad (2.22)$$

Em (2.21), aplicando procedimento idêntico ao usado para se obter (2.22), chega-se à equação de atualização para  $P_{(i,j,k)}^1$ , dada por

$$P_{(i,j,k)}^{1(n+1)} = \frac{\left(\frac{\varepsilon}{\Delta_t} - \frac{\sigma}{2}\right) P_{(i,j,k)}^{1(n)} + \frac{1}{\sqrt{g}} \left( \tilde{h}_{3(i,j,k)}^{n+1} - h_{2(i,j,k)}^{n+\frac{1}{2}} + h_{2(i,j,k-1)}^{n+\frac{1}{2}} \right)}{\left(\frac{\varepsilon}{\Delta_t} + \frac{\sigma}{2}\right)}. \quad (2.23)$$

Por fim, seguindo o procedimento de discretização para a Equação (2.20), chega-se à equação, em diferenças finitas, para a atualização de  $e_{(i,j,k)}^1$

$$e_{(i,j,k)}^{1(n+1)} = \frac{\left(\frac{k_3}{\Delta_t} - \frac{\sigma_3}{2\varepsilon_0}\right) e_{(i,j,k)}^{1(n)} + \frac{P_{(i,j,k)}^{1(n+1)} - P_{(i,j,k)}^{1(n)}}{\Delta_t}}{\left(\frac{k_3}{\Delta_t} + \frac{\sigma_3}{2\varepsilon_0}\right)}. \quad (2.24)$$

Resumindo: Para se obter  $e^1$ , devem ser calculadas (2.22),(2.23) e (2.24), nesta ordem.

O processo descrito acima pode ser aplicado para encontrar as demais componentes de  $\vec{E}$  e  $\vec{H}$  (utilizando (2.13a) e (2.13b)) para a região da UPML. Definindo as variáveis

$$\psi_\alpha^+ = \frac{k_\alpha}{\Delta_t} + \frac{\sigma_\alpha}{2\varepsilon_0}, \psi_\alpha^- = \frac{k_\alpha}{\Delta_t} - \frac{\sigma_\alpha}{2\varepsilon_0}, \quad (2.25)$$

e

$$\Psi^+ = \frac{\varepsilon}{\Delta_t} + \frac{\sigma}{2}, \Psi^- = \frac{\varepsilon}{\Delta_t} - \frac{\sigma}{2}, \quad (2.26)$$

é possível simplificar as equações de atualização. Dessa forma, o conjunto completo de equações obtidas para a UPML em coordenadas gerais para truncar meios condutivos é dado abaixo [5]:

Para  $e^1$ :

$$\bar{h}_{3(i,j,k)} = h_{3(i,j,k)} - h_{3(i,j-1,k)}, \quad (2.27)$$

$$\tilde{h}_{3(i,j,k)}^{(n+1)} = \frac{\psi_2^- \tilde{h}_{3(i,j,k)}^{(n)} + \psi_3^+ \bar{h}_{3(i,j,k)}^{(n+1)} - \psi_3^- \bar{h}_{3(i,j,k)}^{(n)}}{\psi_2^+}, \quad (2.28)$$

$$P_{(i,j,k)}^{1(n+1)} = \frac{P_{(i,j,k)}^{1(n)} \Psi^- + \frac{1}{\sqrt{g}} \left( \tilde{h}_{3(i,j,k)}^{(n+1)} - h_{2(i,j,k)}^{(n+\frac{1}{2})} + h_{2(i,j,k-1)}^{(n+\frac{1}{2})} \right)}{\Psi^+}, \quad (2.29)$$

$$e_{(i,j,k)}^{1(n+1)} = \frac{e_{(i,j,k)}^{1(n)} \psi_3^- + \frac{1}{\Delta_t} (P_{(i,j,k)}^{1(n+1)} - P_{(i,j,k)}^{1(n)})}{\psi_3^+}. \quad (2.30)$$

Para  $e^2$ :

$$\bar{h}_{1(i,j,k)} = h_{1(i,j,k)} - h_{1(i,j,k-1)}, \quad (2.31)$$

$$\tilde{h}_{1(i,j,k)}^{(n+1)} = \frac{\psi_3^- \tilde{h}_{1(i,j,k)}^{(n)} + \psi_1^+ \bar{h}_{1(i,j,k)}^{(n+1)} - \psi_1^- \bar{h}_{1(i,j,k)}^{(n)}}{\psi_3^+}, \quad (2.32)$$

$$P_{(i,j,k)}^{2(n+1)} = \frac{P_{(i,j,k)}^{2(n)} \Psi^- + \frac{1}{\sqrt{g}} \left( \tilde{h}_{1(i,j,k)}^{(n+1)} - h_{3(i,j,k)}^{(n+\frac{1}{2})} + h_{3(i-1,j,k)}^{(n+\frac{1}{2})} \right)}{\Psi^+}, \quad (2.33)$$

$$e_{(i,j,k)}^{2(n+1)} = \frac{e_{(i,j,k)}^{2(n)} \psi_1^- + \frac{1}{\Delta_t} (P_{(i,j,k)}^{2(n+1)} - P_{(i,j,k)}^{2(n)})}{\psi_1^+}. \quad (2.34)$$

Para  $e^3$ :

$$\bar{h}_{2(i,j,k)} = h_{2(i,j,k)} - h_{2(i-1,j,k)}, \quad (2.35)$$

$$\tilde{h}_{2(i,j,k)}^{(n+1)} = \frac{\psi_1^- \tilde{h}_{2(i,j,k)}^{(n)} + \psi_2^+ \bar{h}_{2(i,j,k)}^{(n+1)} - \psi_2^- \bar{h}_{2(i,j,k)}^{(n)}}{\psi_1^+}, \quad (2.36)$$

$$P_{(i,j,k)}^{3(n+1)} = \frac{P_{(i,j,k)}^{3(n)} \Psi^- + \frac{1}{\sqrt{g}} \left( \tilde{h}_{2(i,j,k)}^{(n+1)} - h_{1(i,j,k)}^{(n+\frac{1}{2})} + h_{1(i,j-1,k)}^{(n+\frac{1}{2})} \right)}{\Psi^+}, \quad (2.37)$$

$$e_{(i,j,k)}^{3(n+1)} = \frac{e_{(i,j,k)}^{3(n)} \psi_2^- + \frac{1}{\Delta_t} \left( P_{(i,j,k)}^{3(n+1)} - P_{(i,j,k)}^{3(n)} \right)}{\psi_2^+}. \quad (2.38)$$

Para  $h^1$ :

$$\bar{e}_{3(i,j,k)} = e_{3(i,j,k)} - e_{3(i,j+1,k)}, \quad (2.39)$$

$$\tilde{e}_{3(i,j,k)}^{(n+\frac{1}{2})} = \frac{\psi_2^- \tilde{e}_{3(i,j,k)}^{(n-\frac{1}{2})} + \psi_3^+ \bar{e}_{3(i,j,k)}^{(n+1)} - \psi_3^- \bar{e}_{3(i,j,k)}^{(n)}}{\psi_2^+}, \quad (2.40)$$

$$h_{(i,j,k)}^{1(n+\frac{1}{2})} = \frac{h_{(i,j,k)}^{1(n-\frac{1}{2})} \psi_3^- + \frac{1}{\mu\sqrt{g}} \left( \tilde{e}_{3(i,j,k)}^{(n+\frac{1}{2})} + e_{2(i,j,k+1)}^{(n+1)} - e_{2(i,j,k)}^{(n+1)} \right)}{\psi_3^+}. \quad (2.41)$$

Para  $h^2$ :

$$\bar{e}_{1(i,j,k)} = e_{1(i,j,k)} - e_{1(i,j,k+1)}, \quad (2.42)$$

$$\tilde{e}_{1(i,j,k)}^{(n+\frac{1}{2})} = \frac{\psi_3^- \tilde{e}_{1(i,j,k)}^{(n-\frac{1}{2})} + \psi_1^+ \bar{e}_{1(i,j,k)}^{(n+1)} - \psi_1^- \bar{e}_{1(i,j,k)}^{(n)}}{\psi_3^+}, \quad (2.43)$$

$$h_{(i,j,k)}^{2(n+\frac{1}{2})} = \frac{h_{(i,j,k)}^{2(n-\frac{1}{2})} \psi_1^- + \frac{1}{\mu\sqrt{g}} \left( \tilde{e}_{1(i,j,k)}^{(n+\frac{1}{2})} + e_{3(i+1,j,k)}^{(n+1)} - e_{3(i,j,k)}^{(n+1)} \right)}{\psi_1^+}. \quad (2.44)$$

Para  $h^3$ :

$$\bar{e}_{2(i,j,k)} = e_{2(i,j,k)} - e_{2(i+1,j,k)}, \quad (2.45)$$

$$\bar{e}_{2(i,j,k)}^{(n+\frac{1}{2})} = \frac{\psi_1^- \bar{e}_{2(i,j,k)}^{(n-\frac{1}{2})} + \psi_2^+ \bar{e}_{2(i,j,k)}^{(n+1)} - \psi_2^- \bar{e}_{2(i,j,k)}^{(n)}}{\psi_1^+}, \quad (2.46)$$

$$h_{(i,j,k)}^{3(n+\frac{1}{2})} = \frac{h_{(i,j,k)}^{3(n-\frac{1}{2})} \psi_2^- + \frac{1}{\mu\sqrt{g}} \left( \bar{e}_{2(i,j,k)}^{(n+\frac{1}{2})} + e_{1(i,j+1,k)}^{(n+1)} - e_{1(i,j,k)}^{(n+1)} \right)}{\psi_2^+}. \quad (2.47)$$

Por fim, as funções  $\sigma_\alpha$  e  $k_\alpha$  são definidas. As funções  $\sigma_\alpha$  podem ser generalizadas por

$$\sigma_\alpha(i_\alpha)|_{E^\beta} = \frac{\left| i_\alpha - I_\alpha + \frac{1}{2} \delta_{\alpha,\beta} \right|^m}{n_c^m} \sigma_{\alpha\max} \quad (2.48)$$

e

$$\sigma_\alpha(i_\alpha)|_{H^\beta} = \frac{\left| i_\alpha - I_\alpha + \frac{1}{2} \bar{\delta}_{\alpha,\beta} \right|^m}{n_c^m} \sigma_{\alpha\max} \quad (2.49)$$

nas quais

$$\delta_{\alpha,\beta} = \begin{cases} 1, & \text{se } \alpha = \beta \\ 0, & \text{se } \alpha \neq \beta \end{cases}, \quad (2.50)$$

$$\bar{\delta}_{\alpha,\beta} = \begin{cases} 0, & \text{se } \alpha = \beta \\ 1, & \text{se } \alpha \neq \beta \end{cases}, \quad (2.51)$$

$\alpha = 1,2$  ou  $3$ ,  $\beta = 1,2$  ou  $3$ ,  $i_\alpha$  é o índice da célula atual ( $i_\alpha = i, j$  ou  $k$  de acordo com  $\alpha$ ),  $I_\alpha$

indica a coordenada discreta da interface entre a região de análise e a UPML na direção  $\alpha$ ,  $m$  é a ordem do polinômio  $\sigma_\alpha$  (usualmente  $m = 4$ ),  $\sigma_{\alpha\max}$  é o máximo valor da condutividade  $\sigma_\alpha$ , que é atribuído à última célula da UPML na direção  $\alpha$  e  $n_c$  é o número de camadas (em células) usadas para a UPML ( $n_c = 10$ ). A notação  $\sigma_\alpha(i_\alpha)|_{A^\beta}$  representa a condutividade  $\sigma_\alpha$  na direção  $\alpha$  para a componente contravariante  $\beta$  do campo elétrico ( $A = E$ ) ou para a componente contravariante  $\beta$  do campo magnético ( $A = H$ ).

As funções  $k_\alpha$  são, de forma análoga, definidas por

$$k_\alpha(i_\alpha)|_{E^\beta} = 1 + \frac{(k_{\alpha\max} - 1) \left| i_\alpha - I_\alpha + \frac{1}{2} \delta_{\alpha,\beta} \right|^m}{n_c^m} \quad (2.52)$$

e

$$k_\alpha(i_\alpha)|_{H^\beta} = 1 + \frac{(k_{\alpha\max} - 1) \left| i_\alpha - I_\alpha + \frac{1}{2} \bar{\delta}_{\alpha,\beta} \right|^m}{n_c^m}. \quad (2.53)$$

Neste trabalho, foi utilizado  $k_{\alpha\max} = 7$  [88] e  $\sigma_{\alpha\max} = 11(m + 1)/(1500\pi\sqrt{g_{\alpha\alpha_{av}}})$ , em que  $\sqrt{g_{\alpha\alpha_{av}}}$  é a média de  $\sqrt{g_{\alpha\alpha}}$  dentro da UPML.

## 2.5. Processamento Paralelo para o método LN-FDTD e threads

Uma das maneiras de aplicar a computação paralela para resolver problemas em eletromagnetismo pelo LN-FDTD é a divisão espacial do domínio de análise em subdomínios [14]. A Figura 2.3 ilustra o caso em que são considerados dois subdomínios. O conceito de subdomínio é geralmente associado a arquiteturas com memória distribuída, como em *clusters* do tipo *Beowulf* [15]. Neste caso, cada subdomínio é visto computacionalmente como uma fração de todo o volume numérico (os *arrays* são fragmentados em *arrays* menores), que será tratada por um único núcleo (*core*) de processamento. Isto ocorre porque, neste tipo de sistema, a memória é distribuída fisicamente entre os computadores. Os núcleos executam essencialmente o mesmo código, mas com condições particulares de contorno e a



continuidade do algoritmo é garantida através de trocas de mensagens entre os computadores envolvidos através de uma rede local (*ethernet*, por exemplo). Para isto, são comumente utilizadas bibliotecas baseadas nas especificações MPI (*Message Passing Interface*) ou PVM (*Parallel Virtual Machine*) [15].

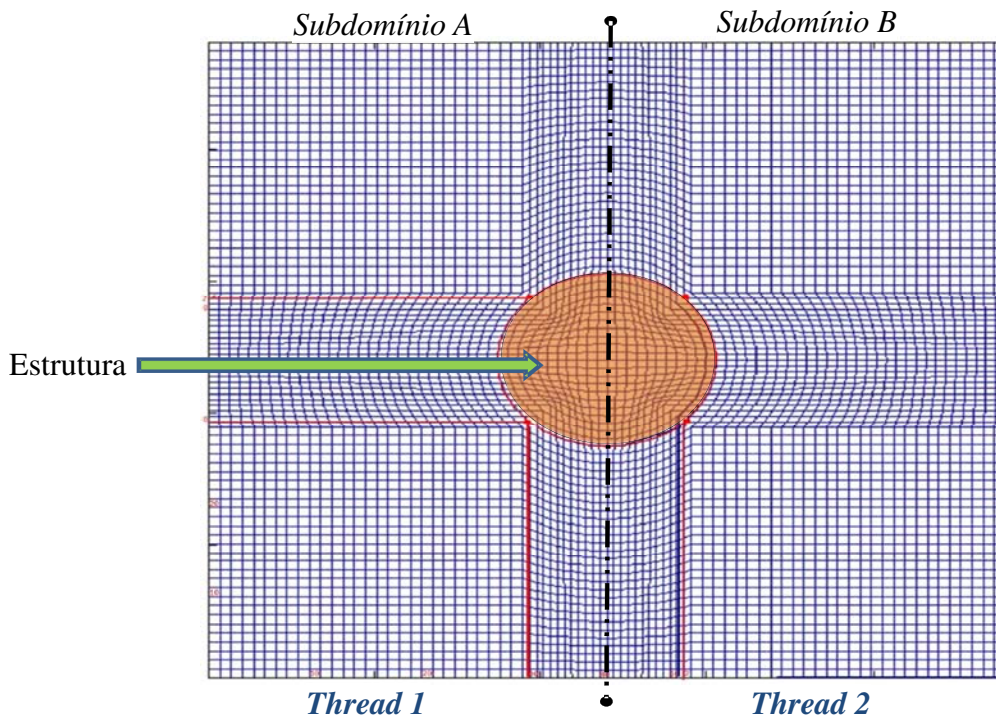


Figura 2.3: Decomposição do domínio em dois subdomínios e seção transversal (superfície 1-2) de uma malha 3D não-ortogonal.

Recentemente, limitações de ordem física impediram que as empresas produtoras de *chips* aumentassem a frequência de operação dos processadores baseados em silício [16]. Este entrave levou a indústria de processadores a aumentar o desempenho dos computadores digitais através de *chips* multiprocessados, ou seja, com múltiplos núcleos de processamento em um único *chip* processador. Dessa forma, paradigmas de programação tais como *threads* [17], utilizada no passado para emular multitarefa e em sistemas com múltiplos processadores, têm recebido grande atenção neste novo cenário [16]. Neste trabalho, o processamento paralelo foi implementado através de *threads* no padrão POSIX (*pthread*) [17], cujos conceitos básicos são abordados objetivamente a seguir.

### 2.5.1. *Threads*

*Thread* é a menor unidade de processamento que pode ser escalonada pelo sistema

operacional: é um recurso do qual os processos se utilizam para poderem executar tarefas de forma concorrente, ou seja, paralelamente. Este paralelismo só ocorre de forma real em máquinas com suporte para *multithreading* (o sistema operacional deve suportá-las), tal como ocorre nos sistemas com múltiplos núcleos. Dessa forma, é possível aumentar o desempenho do referido processo [17].

As *threads* criadas por um programa (um dado processo) compartilham o mesmo espaço de memória. Dessa forma, não é necessário efetuar troca de mensagens entre elas, como ocorre com sistemas distribuídos (baseados em MPI ou PVM, nos quais os processos trocam informações através da rede). Sendo assim, informações relativas a outras *threads* podem ser obtidas diretamente através do acesso a variáveis (e *arrays*) globais.

### 2.5.2. Memória Compartilhada

Em um programa baseado em *threads*, as variáveis podem ser declaradas de forma local (memória privada) e/ou de forma global (memória compartilhada), tal como ilustrado pela Figura 2.4.

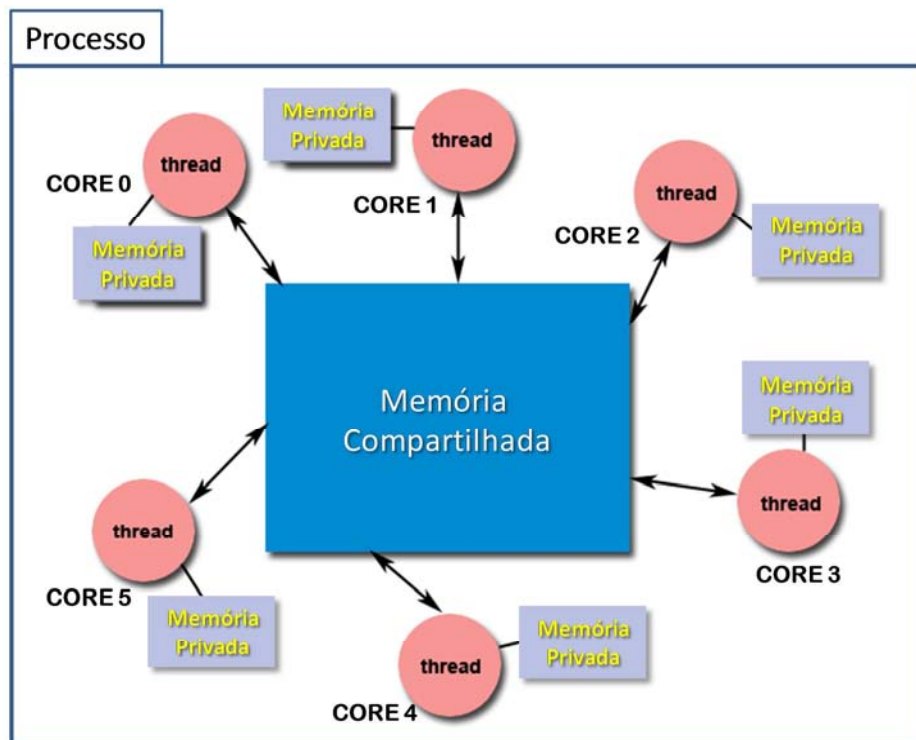


Figura 2.4: Processo e estrutura de memória compartilhada e privada em uma máquina com seis *cores*.

Para programas escritos em C, baseados na biblioteca pthread [17], as *threads* são implementadas como sub-rotinas, as quais podem ser executadas concorrentemente utilizando

a função `pthread_create`. Tal como ocorre em programas convencionais, as variáveis globais são declaradas fora das sub-rotinas e da função `main()` e as locais (memória privada) são declaradas em suas respectivas funções.

Variáveis pertencentes à memória compartilhada podem ser acessadas por múltiplas *threads* simultaneamente. Se os acessos forem para leitura, deve-se ter, por tanto, o cuidado de garantir que uma dada *thread* realize esta operação após as variáveis de interesse tenham sido atualizadas previamente na memória compartilhada (possivelmente por outras *threads*). No caso do acesso para alterar o valor de variáveis na memória compartilhada (escrita), não se pode permitir que múltiplas *threads* alterem uma dada variável simultaneamente; caso contrário, o último valor assumido por esta variável se torna imprevisível.

Dessa forma, variáveis pertencentes à memória compartilhada devem ser tratadas de forma sincronizada para evitar imprevisibilidades na resposta do programa. Isto pode ser alcançado através de mecanismos de sincronismo relativos aos acessos, tais como os semáforos [17]. No caso do método LN-FDTD, é possível garantir a sincronia dos processos suspendendo a execução da função (*thread*) `main()` até que as *threads* usadas para calcular as campos covariantes ou contravariantes de  $\vec{E}$  ou  $\vec{H}$  tenham sido finalizadas. Isto pode ser implementado através de chamadas a `pthread_join()`, tal como é ilustrado pelo programa-exemplo da Figura 2.5.

O exemplo mostrado na Figura 2.5 mostra didaticamente como inicializar em paralelo as quatro posições de um *array* **x** utilizando quatro *threads*. A função `MyThread()` é usada com quatro *threads* diferentes. A saída em tela do programa abaixo é “0 1 2 3”.

---

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 4

float *xp; //ponteiro de acesso a x (para compartilhamento)
void *MyThread( void *tid ){ // funcao usada como thread: *tid é o seu argumento
    int t = (int) tid;
    xp[t] = (float) (t) ;
    pthread_exit (NULL);
}

int main (int argc, char *argv[]){ //função principal do programa (thread mãe)
    float x[NUM_THREADS];
    xp = x; // compartilhamento do endereço de x
    pthread_t threads[NUM_THREADS]; //vetor usado para guardar ids das threads
```

```

int rc, t;
////////////////////////////////////
for(t=0; t<NUM_THREADS; t++){
    // cria as threads (inicializa todos os elementos de x em paralelo):
    // t é argumento de MyThread
rc = pthread_create(&threads[t], NULL, MyThread, (void *) t);
if (rc){ printf("Erro: %d\n", rc);
    exit(-1);} // termina programa em caso de erro na criação de uma thread
}

for(t=0; t<NUM_THREADS; t++){
    rc = pthread_join(threads[t], NULL); //aguarda o fim das 4 threads
}
////////////////////////////////////
printf("x = ");
for(t=0; t<NUM_THREADS; t++){
    printf( "%f ", x[t] ); // imprime x
}
printf("\n");
pthread_exit(NULL);
} // fim da main()

// compilação: gcc -pthread -lm -O programa.c

```

---

Figura 2.5: Código fonte didático: uso da *pthread*.

## 2.6. Malhas Estruturadas e o método LN-FDTD

Para a geração de muitas das figuras dos cenários tridimensionais apresentados neste trabalho, é utilizada uma malha estruturada e esta é a mesma usada na realização da simulação numérica. Como mencionado anteriormente esta malha deve ser estruturada (Figura 2.6) [3] devido à formulação apresentada nos tópicos 2.2-2.4, o que está relacionado à natureza do método LN-FDTD.

A malha deve ser desenvolvida de acordo com a natureza geométrica do problema a ser simulado e é construída pelo usuário e fornecida ao *software* desenvolvido neste trabalho. Os dados da malha construída são passados através de um arquivo de entrada para concepção da estrutura (que consiste na definição das condições de contorno e parâmetros eletromagnéticos) e para realizar os cálculos através do simulador. Por exemplo, a malha representada pela Figura 2.3 pode ser usada para representar um cilindro.

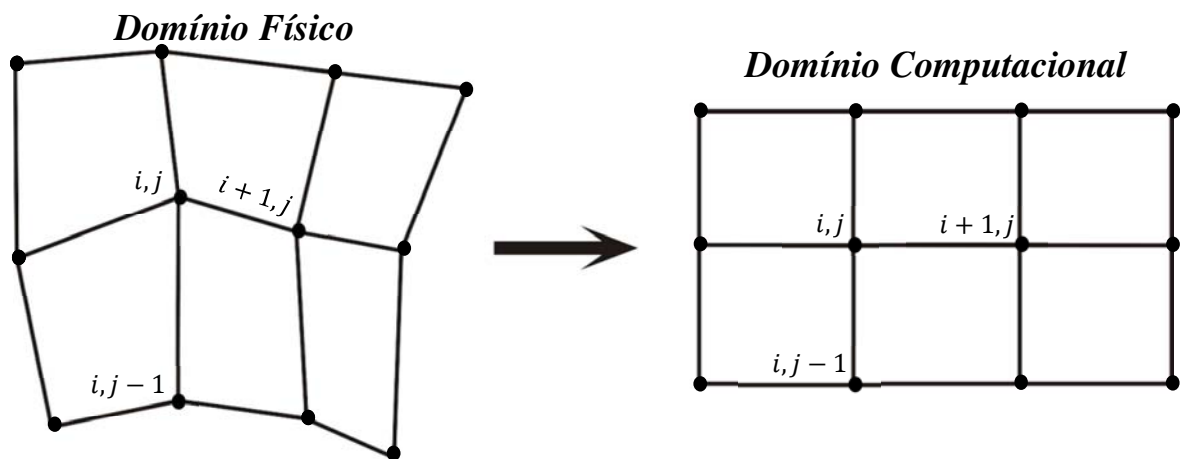


Figura 2.6: Exemplo de uma malha estruturada na visão do domínio físico e na visão do domínio computacional (como o algoritmo visualiza as malhas).

## 2.7. Modelos de processos para construção de *Softwares*

A maioria dos *softwares* de hoje fornece algum tipo de interface gráfica de usuário (GUI). Uma GUI deve ser de fácil uso e operar de forma intuitiva, para ser alcançado o resultado desejado. As ações devem ser acessadas por cliques do mouse sobre objetos gráficos, tornando o *software* ágil e deixando suas funcionalidades acessíveis, ao ponto do usuário poder operá-lo de forma adequada e sem grande esforço [10].

Para que o *software* atenda a essas características, a Engenharia de *Software* [11] [18] aborda vários modelos de processo de construção de *software*, permitindo ao desenvolvedor optar pelo modelo adequado para seu projeto.

O modelo adaptado para o desenvolvimento de *software* deve ser eficiente, de forma a satisfazer plenamente as necessidades do usuário [9]. Assim são descritos a seguir os modelos de processo [11] [18] mais comumente utilizados na Engenharia de *Software*. Vale lembrar que a escolha do modelo de processo depende do projeto de *software* a ser executado e das necessidades identificadas.

### 2.7.1. Modelo Sequencial Linear

O modelo sequencial linear sugere uma abordagem sequencial para o desenvolvimento de *software*, que inicia na Engenharia de Sistema, que ocorre simultaneamente com a Análise

e Projeto e segue para Codificação, Teste e por fim a Manutenção.

Na Figura 2.7 são apresentadas as atividades ou etapas do modelo sequencial linear.

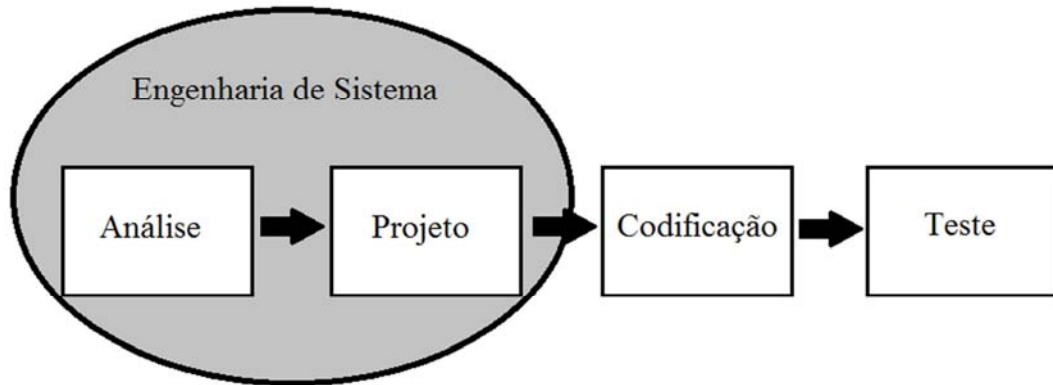


Figura 2.7: O modelo sequencial linear.

- **Etapa Engenharia de Sistema:** Esta etapa tem por objetivo, estabelecer os requisitos para todos os elementos do sistema e alocar um subconjunto desses requisitos para o *software*. Isto é essencial quando o *software* necessita interagir com outros elementos tais como *hardware* e usuários.
- **Etapa Análise de Requisitos de Software:** A análise de requisitos é intensificada e focalizada somente no *software*. Para entender a natureza do programa a ser desenvolvido, o engenheiro de *software* deve ter o domínio da informação do *software*, definir as funções necessárias e o comportamento desejado, fazer a análise de desempenho e estabelecer critérios para construir a interface. Os requisitos do *software* são documentados e analisados com o cliente.
- **Etapa Projeto:** Esta etapa é, na verdade, um processo de múltiplos passos que focaliza quatro propriedades distintas do programa: estrutura de dados, arquitetura do *software*, representações da interface e detalhes algorítmicos. Esta etapa traduz os requisitos para uma representação do *software*, que pode ser aferida quanto à qualidade, antes que a etapa de codificação se inicie.
- **Etapa Codificação:** Esta etapa se caracteriza pela tradução do projeto para a linguagem de programação. Se a etapa projeto é realizada de maneira detalhada, a codificação pode ser realizada de forma eficiente.
- **Etapa Teste:** Após a etapa de codificação, os testes do programa têm início. A

etapa teste enfoca os aspectos lógicos internos do *software*, garantindo que todas as funções sejam testadas, além dos aspectos externos funcionais, isto é, os testes são realizados para descobrir possíveis erros e garantir que entradas definidas produzirão os resultados esperados, que devem concordam com o que foi definido pelo cliente.

- **Etapa Manutenção:** Após o *software* ser liberado ao cliente, é inevitável que este sofra modificações posteriores. A modificação é realizada quando erros são encontrados, quando o *software* precisa ser adaptado para modificações em seu ambiente externo (novo sistema operacional ou dispositivo periférico), ou quando o cliente deseja atualizações (inclusão de funcionalidades ou mesmo melhor desempenho). Esta etapa re replica cada uma das fases precedentes a um programa existente, sem construir um novo programa.

Estas etapas são descritas em [11]. Este modelo é o mais antigo e é o mais usado na engenharia de *software*. Contudo, ele pode apresenta alguns problemas identificados por [11], que são:

- A maioria dos projetos raramente segue o fluxo sequencial que o modelo propõe. Com isso, modificações podem causar confusão à medida que a equipe de projeto avança.
- Este modelo exige que o cliente estabeleça todos os requisitos explicitamente, o que é difícil de acontecer, e com isso, o modelo tem a dificuldade de acomodar a incerteza natural que existe no início da maioria dos projetos.
- Este modelo gera um executável somente ao final do projeto. Com isso, erros não detectados podem ser desastrosos para o projeto.

### 2.7.2. Modelo Cascata

Este modelo se caracteriza pela sequência em cascata de uma etapa para outra, conforme a Figura 2.8.

Quando uma etapa é finalizada, são gerados documentos para serem assinados pelo engenheiro de *software*. Com isso, é garantido que a fase seguinte não inicie até que a etapa atual seja concluída [18].



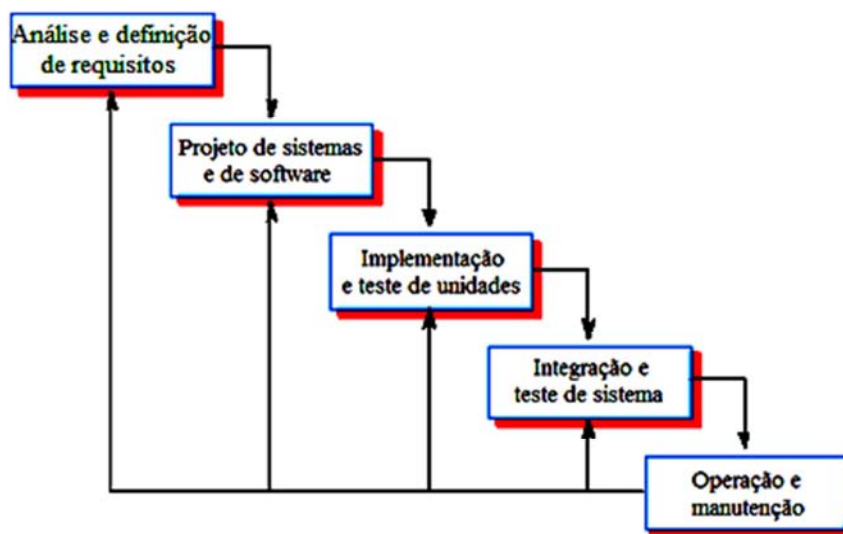


Figura 2.8: O modelo cascata.

As etapas deste modelo descritas por [18] são:

- **Análise e definição de requisitos:** Todas as funcionalidades do sistema, as restrições e os objetivos são os aspectos tratados nesta etapa em conjunto com os usuários do sistema. Com isso, esses requisitos são definidos em detalhes e servem como uma especificação do sistema.
- **Projeto de sistema e de software:** Esta etapa agrupa os requisitos sistêmicos de *hardware* ou de *software*. Em seguida, é estabelecida uma arquitetura geral do sistema. Por fim, são identificadas e descritas as abstrações do sistema de *software* e suas relações.
- **Implementação e teste de unidades:** Durante esta etapa, o projeto de *software* é compreendido como um conjunto de programas ou unidades de programa. O teste de unidades garante que cada unidade atenda a sua especificação.
- **Integração e teste de sistemas:** As unidades de programas (ou programas individuais) são integradas e testadas como um sistema completo, a fim de garantir que os requisitos de *software* sejam atendidos. Ao final dos testes, o sistema de *software* é entregue ao cliente.
- **Operação e manutenção:** Nesta última etapa, o sistema é instalado e colocado em uso. A manutenção serve para corrigir erros que não foram descobertos nas etapas anteriores, melhorando a implementação do sistema e incrementando suas funções, conforme são gerados novos requisitos.



De acordo com [18], este modelo de processo de *software* deve ser utilizado somente quando os requisitos forem bem compreendidos. Caso contrário, o sistema não fará o que foi definido pelo usuário, gerando assim um sistema mal estruturado que não será capaz de comportar futuros requisitos estabelecidos pelo cliente. Contudo, este modelo reflete a prática da engenharia e é mais utilizado quando um projeto de grande porte de engenharia de sistemas é desenvolvido.

### 2.7.3. Modelo de Processo em Espiral

O modelo de processo em espiral (Figura 2.9) desvia da representação dos modelos de processo mencionados anteriormente, pois, em vez de organizar as etapas do processo em uma sequência com algum retorno de uma etapa para outra, o processo é modelado por uma espiral. Cada *loop* na espiral significa uma etapa do processo de *software*. Assim, o *loop* mais interno está relacionado à viabilidade do sistema; o *loop* seguinte, à definição de requisitos do sistema; o próximo *loop*, ao projeto do sistema, e assim por diante [18].

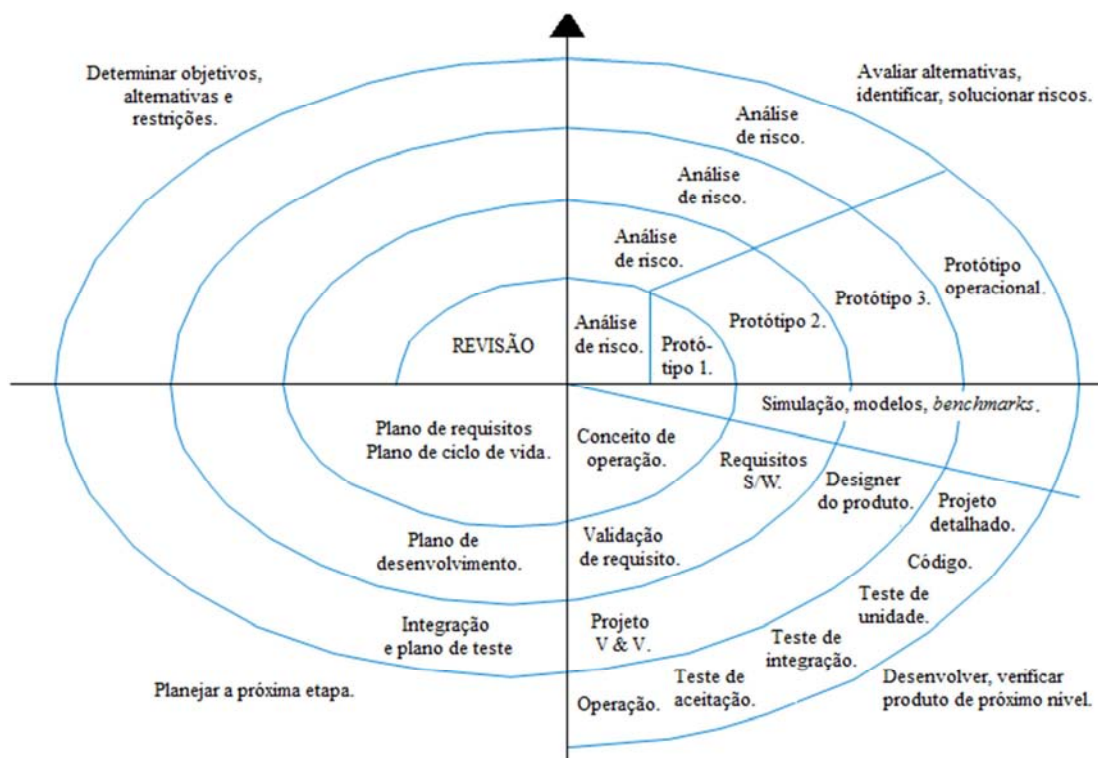


Figura 2.9: Modelo de processo em espiral.

Cada *loop* da espiral é dividido em quatro seções [18]:

- **Definição de objetivos:** Nesta seção são definidos os objetivos específicos para cada etapa do projeto. São identificadas restrições para o projeto e o produto (*software*) e é preparado um plano de gerenciamento detalhado. Os riscos do projeto são identificados e, dependendo dos riscos, poderão ser planejadas estratégias alternativas.
- **Avaliação e redução de riscos:** Para todos os riscos de projetos identificados, é realizada uma análise e são tomadas medidas para reduzir esses riscos. Por exemplo, se houver um risco de os requisitos serem inadequados, poderá ser desenvolvido um protótipo (Figura 2.9).
- **Desenvolvimento e validação:** Nesta seção é escolhido um modelo de desenvolvimento para o sistema. Se forem dominantes os riscos relacionados à interface de usuário, um modelo de prototipagem seria mais apropriado (próximo tópico). Se os riscos de segurança forem o principal foco, o modelo desenvolvimento formal de sistemas [18] poderá ser o mais adequado. Assim, se o risco principal for a integração de sistemas, o modelo em cascata poderá ser o mais apropriado.
- **Planejamento:** Nesta seção, o projeto é revisto e é tomada a decisão sobre continuar com o próximo *loop* da espiral. Se a decisão for continuar, os planos para a próxima etapa serão traçados.

Neste modelo de processo, é explícita a importância dos riscos. De maneira informal, o risco é simplesmente algo que pode acontecer de errado. Por exemplo, se há a intenção de se utilizar uma nova linguagem de programação, haverá o risco de que os compiladores disponíveis não sejam confiáveis ou que não produzam o código-objeto suficientemente eficiente. Com isso, poderão ser desencadeados problemas no projeto, como a possibilidade de exceder o prazo e o custo previstos no projeto. Portanto, minimizar os riscos é uma atividade muito importante neste modelo de processo de *software* [18].

Dessa forma, no modelo de processo em espiral não há etapas fixas, como especificação e projeto. Ele abrange os outros modelos de processos de acordo com as necessidades do projeto. O modelo de prototipagem poderá ser utilizado em um modelo espiral para resolver dúvidas em relação a requisitos e, assim, reduzir riscos. O modelo desenvolvimento formal de sistemas poderá ser utilizado para resolver partes de um sistema com muitos requisitos de segurança [18].

## 2.7.4. Modelo de Prototipagem

Em muitos projetos para desenvolvimento de *software*, o cliente define um conjunto de objetivos gerais para o *software*, mas não identifica detalhadamente requisitos de entrada, processamento ou saída. É possível que o desenvolvedor esteja inseguro quanto à eficiência de um algoritmo, quanto ao pleno atendimento das necessidades do cliente, quanto ao comportamento do *software* em relação ao sistema operacional ou quanto à forma que a *interação homem/máquina* deve assumir. Para muitas dessas questões de inseguranças, o modelo de prototipagem pode oferecer a melhor abordagem [11].

O modelo de prototipagem (Figura 2.10) inicia com a definição de requisitos [11]. O desenvolvedor e o cliente definem os objetivos gerais do *software*, identificam as necessidades conhecidas e descrevem as áreas que necessitam de mais definições. Com isso, um “projeto rápido” é realizado. Esse projeto irá expor a representação dos aspectos do *software* (por exemplo, requisitos de entrada e formatos de saída) que irão ficar visíveis ao cliente/usuário. O projeto rápido inicia um protótipo. O protótipo é avaliado pelo cliente/usuário e usado para refinar os requisitos do *software* que será desenvolvido. No decorrer do projeto, as interações ocorrem à medida que o protótipo é ajustado para satisfazer as necessidades do cliente e, com isso, o desenvolvedor pode entender com precisão o que precisa ser feito.

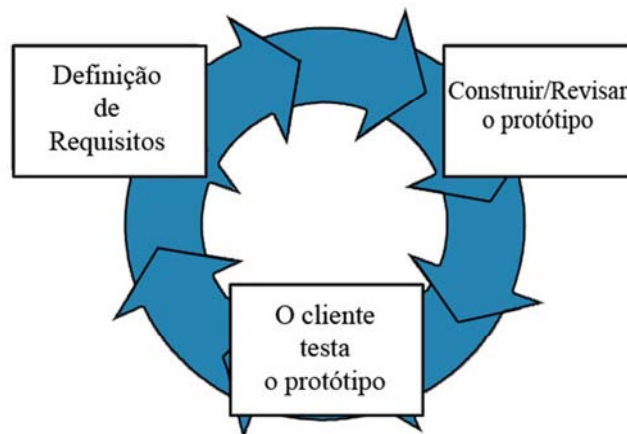


Figura 2.10: Modelo de prototipagem.

Tantos os clientes como os desenvolvedores gostam do processo de modelo de prototipagem. Os clientes conseguem ter uma ideia prévia de como ficaria o sistema final e os

desenvolvedores conseguem construir um protótipo em um curto espaço de tempo. Porém, este modelo pode apresentar os seguintes problemas [11]:

- O modelo de prototipagem pode dar ao cliente a impressão de que praticamente qualquer sugestão (acréscimo de funcionalidades) pode ser implementada, não levando em conta em que estágio do processo de desenvolvimento o projeto se encontra.
- O cliente não entende o porquê da demora em entregar a versão final do *software*, depois de um protótipo ser apresentado.
- A cada protótipo apresentado, o cliente irá pedir modificações e, com o tempo, se o desenvolvedor não definir as regras para o cliente de que o protótipo é construído para servir como mecanismo para a definição dos requisitos, isto passa a ser um problema quando o cliente passar a realizar muitas mudanças a cada protótipo.

Apesar de ocorrerem problemas, o modelo de prototipagem é eficiente e bastante utilizado na engenharia de *software*. O desenvolvedor e o cliente devem estar de acordo que o protótipo servirá como mecanismo para a definição dos requisitos. Este protótipo será descartado (em partes), e o *software* real é submetido à engenharia com o objetivo de buscar qualidade, eficiência e com isso atendendo plenamente às necessidades do cliente [11].

## **2.8. Cenário Gráfico Tridimensional**

Um cenário gráfico tridimensional é uma representação virtual de um cenário real. Para entender como é desenvolvido um cenário gráfico tridimensional, é necessário conhecer os principais conceitos envolvidos, que são: visualização e criação tridimensional, sólidos, câmera, projeção, e modelagem de superfícies (visualização da malha).

A biblioteca utilizada para a criação de cenários gráficos tridimensionais é a OpenGL [13].

### **2.8.1. Visualização e criação tridimensional.**

Quando se trabalha com três dimensões em OpenGL, o Sistema de Referência do

Universo (SRU) passa a possuir três eixos ortogonais entre si ( $x$ ,  $y$ ,  $z$ ) e a origem (0.0, 0.0, 0.0), conforme ilustra a Figura 2.11, tal como ocorre no sistema Cartesiano de Coordenadas.

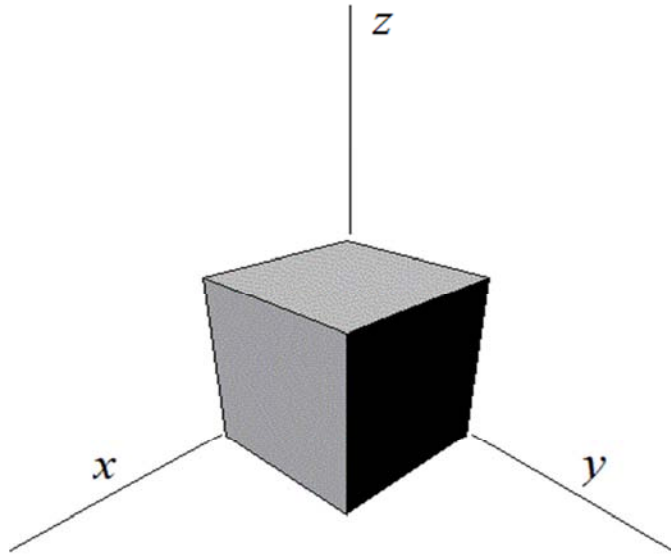


Figura 2.11: O SRU em três dimensões (OpenGL).

A partir daí, um ponto é definido pelos valores de  $x$ ,  $y$  e  $z$  que correspondem às suas coordenadas nos respectivos eixos [13]. Com isso, compreende-se facilmente como o sistema de coordenadas funciona em 3D (idêntico ao sistema Cartesiano), e, a partir dele, poderão ser formadas as superfícies do cenário, especificando e posicionando os pontos (em uma malha associada), usando as coordenadas de cada eixo.

O processo de visualização tridimensional (3D) é mais complexo do que o bidimensional (2D), pois envolve um número maior de etapas. A complexidade está ligada diretamente aos dispositivos de saída, tais como monitores e impressoras que são 2D, embora hoje já existam monitores e impressoras 3D no mercado, mas com um custo elevado em relação aos dispositivos 2D. Assim, é preciso definir como um cenário 3D será projetado em um dispositivo 2D (Tópicos 2.7.3 e 2.7.4).

### 2.8.2. Sólidos

Para o cenário gráfico tridimensional, um sólido é considerado tudo que tem forma

própria (Figura 2.11). Como o cenário não será representado com objetos compostos por materiais líquidos, gasosos ou flexíveis, este conceito se emprega perfeitamente.

### 2.8.3. Câmera

Para criar um cenário gráfico tridimensional, deve-se primeiramente, definir os objetos que o compõe. Tais elementos serão incluídos e posicionados no SRU através de suas coordenadas tal como anteriormente comentado [13].

O próximo passo consiste em definir as especificações do observador no cenário 3D, em que é estabelecido de que ponto no espaço deseja-se que a cena 3D seja exibida. Deste modo, a especificação do observador inclui a sua posição e orientação, ou seja, onde o observador está e para onde ele está olhando no cenário 3D (alvo) [13]. A imagem gerada da posição e orientação do observador é estática. Assim, faz-se a relação com uma foto (Figura 2.12). Porém, modificando-se a posição do observador, cria-se a ideia de movimento e de interatividade. Isto pode ser controlado e implementado utilizando os botões do *mouse* ou do teclado.

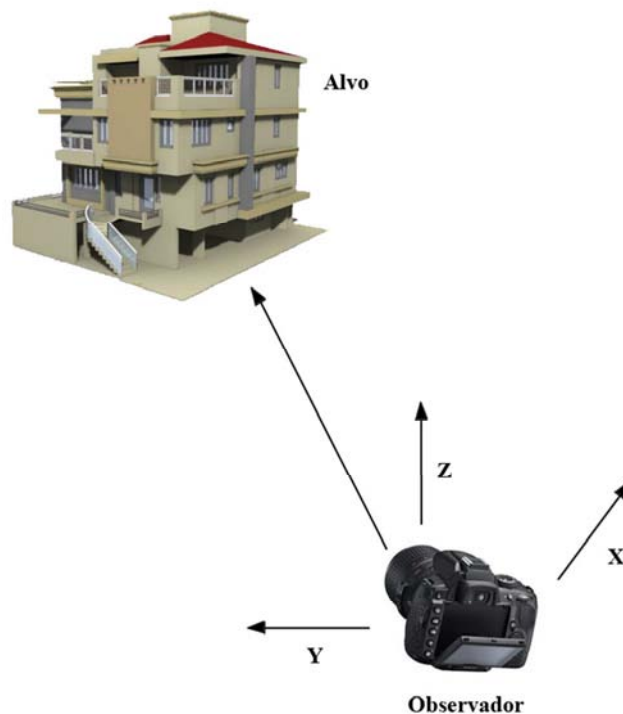


Figura 2.12: Modelo de câmera utilizada em OpenGL.

## 2.8.4. Projeção

Para representar os objetos no cenário 3D, existe uma etapa obrigatória: mapear suas representações 3D para imagens 2D que serão exibidas em um dispositivo como um monitor. Esta operação (obter representações bidimensionais a partir de objetos tridimensionais) é chamada de projeção [13].

Um objeto 3D é um conjunto de pontos (vértices). Sua projeção é definida por segmentos de retas chamados de projetantes, que passam através de cada vértice do objeto e interseccionam um plano de projeção. As projeções são divididas em dois tipos principais [13]:

- **Projeção paralela ortográfica:** os segmentos de reta são paralelos entre si, passam pelos pontos que definem os objetos e interseccionam o plano e a janela de projeção com um ângulo de  $90^\circ$  conforme a Figura 2.13.

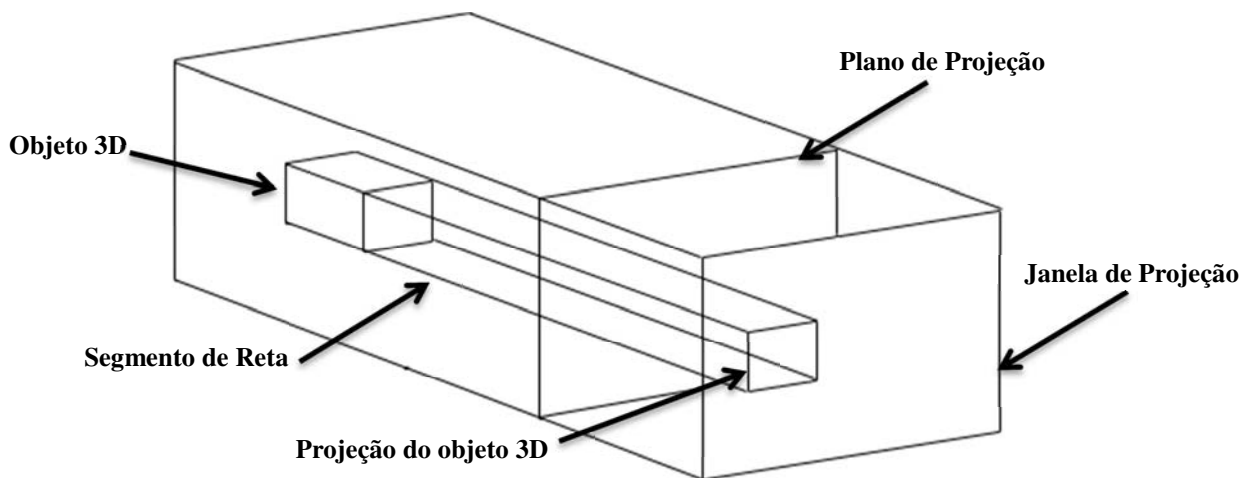


Figura 2.13: Projeção paralela ortográfica.

- **Projeção Perspectiva:** os segmentos de reta surgem de um único ponto (centro de projeção) que está a uma distância finita do plano e da janela de projeção e passam pelos pontos que definem os objetos conforme a Figura 2.14.

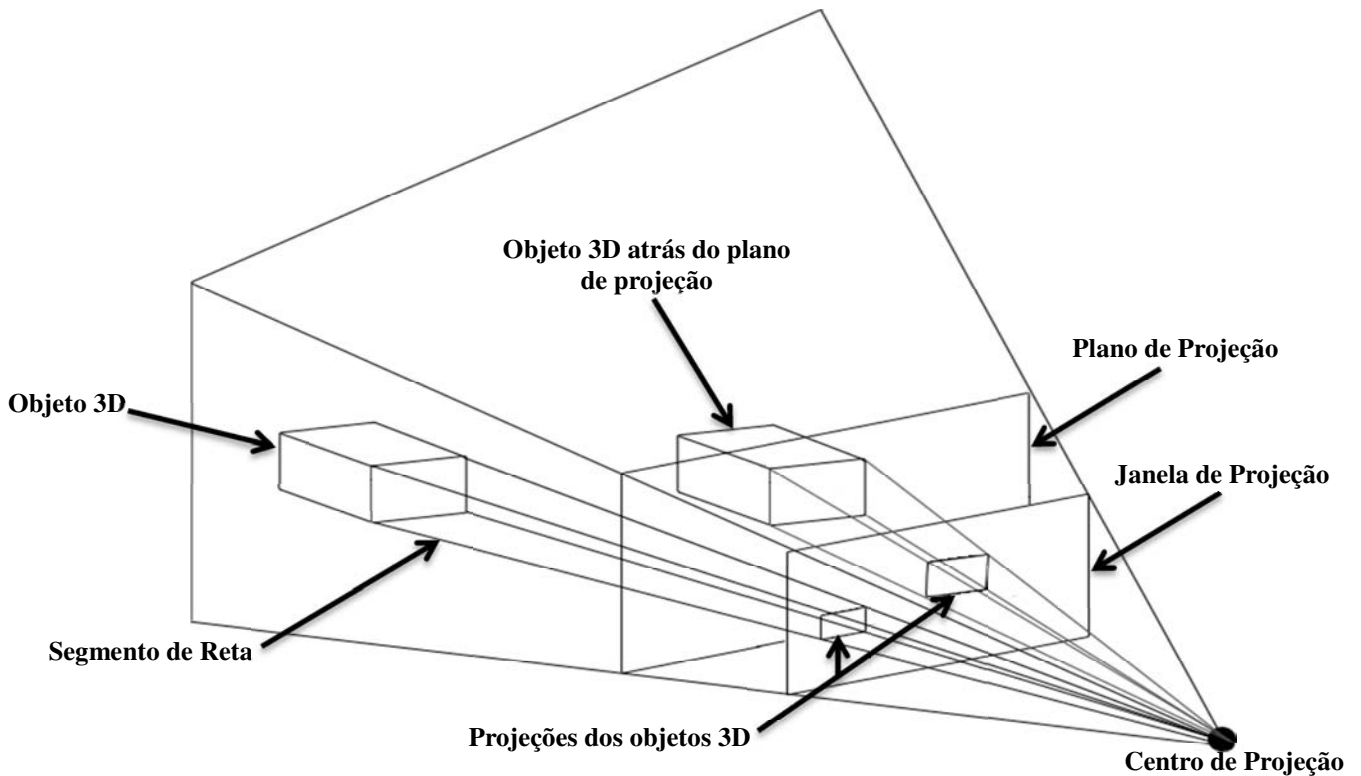


Figura 2.14: Projeção Perspectiva.

A projeção paralela ortográfica não altera as medidas do objeto na janela de projeção. Entretanto, a projeção perspectiva é mais utilizada, pois ela representa melhor o que acontece na realidade. No caso de dois objetos de mesmo tamanho, porém posicionados em distâncias diferentes da janela de projeção, o que estiver mais longe aparecerá menor do que o objeto que estiver mais próximo (Figura 2.14) [13].

### 2.8.5. Modelagem de superfícies (visualização da malha)

A modelagem de superfícies tem por finalidade criar objetos a partir das coordenadas  $x$ ,  $y$  e  $z$  no ambiente tridimensional, o que pode ser feito passando como parâmetro de entrada um arquivo contendo as coordenadas e parâmetros para a criação do cenário gráfico tridimensional. A partir deste ponto, usando as coordenadas especificadas em outro arquivo de entrada, os objetos poderão ser criados no cenário 3D, desde que seja respeitado o limite imposto pelas células utilizadas para discretizar o espaço tridimensional, conforme ilustra a Figura 2.15.



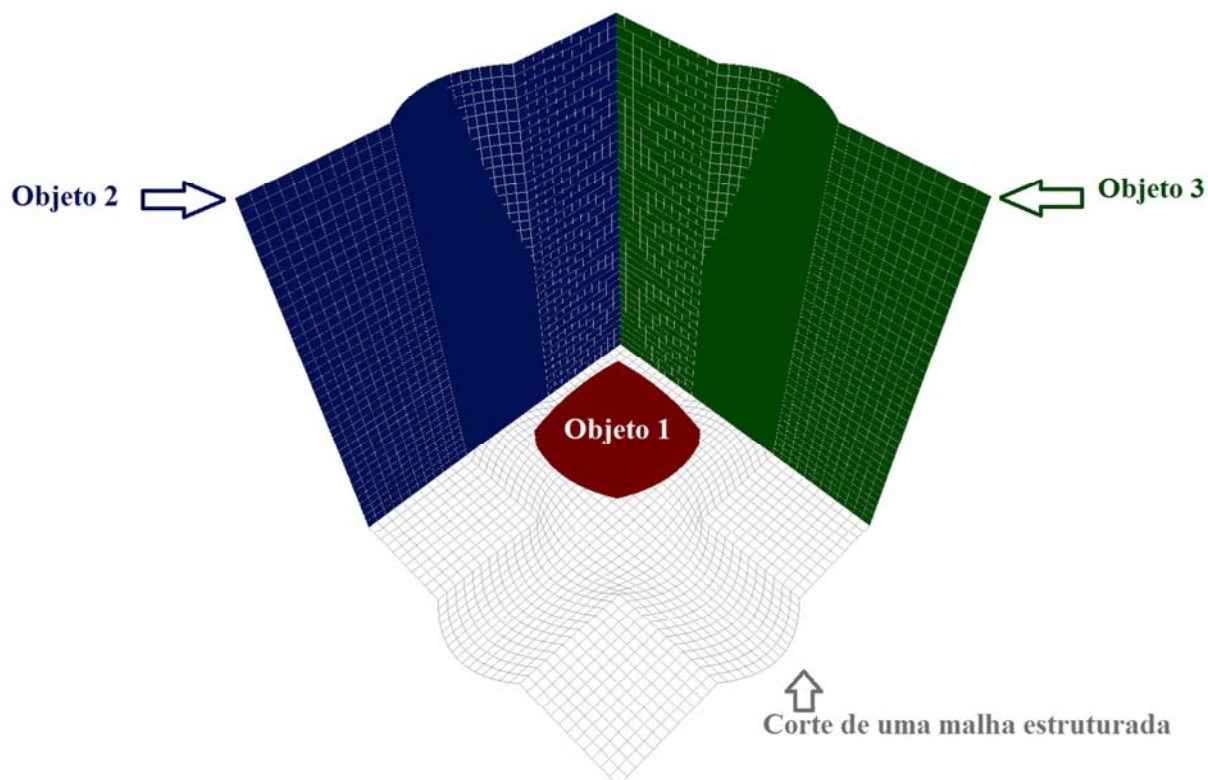


Figura 2.15: Cenário gráfico tridimensional, criado a partir da leitura de um arquivo de entrada.

Na Figura 2.15, têm-se um exemplo de cenário criado a partir de um arquivo de entrada (malha estruturada), o qual contém as coordenadas e seus pontos (relativas ao sistema Cartesiano / SRU). A partir de tais pontos, podem ser construídos os objetos, respeitando-se os limites do domínio de análise, os quais são especificados no arquivo de entrada. Como a malha é estruturada, a visualização da mesma é gerada simplesmente ligando-se os pontos vizinhos do domínio computacional (Figura 2.6), o que é realizado através de vetores nas direções  $\vec{a}_1$ ,  $\vec{a}_2$  e  $\vec{a}_3$ , tal como ilustrado na Figura 2.1. Para representar um objeto como os da Figura 2.15, o volume de cada célula computacional é preenchido por uma cor, dando a ideia de que o contorno dos objetos acompanha as curvas da malha. Dessa forma, é possível ter a perfeita noção de onde o objeto será posicionado na malha e de sua forma geométrica.

## Capítulo 3 - Desenvolvimento do Software

Atualmente os *softwares* estão em todos os lugares: cartões de crédito, em uma cafeteira, celulares e em muitos outros equipamentos eletrônicos. Todos esses dispositivos apresentam algum tipo de *software*. Os *softwares* são utilizados para ajudar nas operações das indústrias, escolas, universidades e etc. Muitas pessoas os utilizam para diversos fins, como, por exemplo, entretenimento e educação. A especificação, o processo de desenvolvimento, o gerenciamento e a evolução desses sistemas de *software* estão diretamente ligados a Engenharia de *Software*. Mesmo *softwares* simples possuem uma alta complexidade inerente e, com isso, os princípios da engenharia devem ser empregados em seu desenvolvimento [18]. Os processos de desenvolvimento de *software* são complexos, pois, todo processo depende do julgamento e da criatividade humana. Com isso, há uma enorme diversidade na forma de como realizar um processo de desenvolvimento de *software*; não há um processo padrão ou ideal e cada processo é organizado de diferentes formas de acordo com o *software* a ser desenvolvido e com as necessidades do usuário que irá utilizar o *software* [9] [10].

Apesar de existirem muitos processos de desenvolvimento de *software* diferentes, todos possuem etapas comuns tal como: Definição de requisitos, Especificação de Software, Arquitetura, Implementação ou Codificação, Testes e Validação, Documentação, Suporte e Treinamento e, por fim, Manutenção [18].

O engenheiro de *software* não precisa seguir todas estas etapas. Ele deve adequá-las a sua necessidade e a do usuário que utilizará o *software*, gerando assim um produto final com qualidade, baixo custo e entrega no prazo.

Baseado no contexto descrito acima, neste capítulo é descrito o desenvolvimento do *software* GUI LN-FDTD Coordenadas Gerais 3D e suas etapas.

### 3.1. Etapas de Desenvolvimento

O processo de desenvolvimento do *software* deste trabalho foi dividido em: Análise de Requisitos, Estudo da Tecnologia, Desenvolvimento do Protótipo e Testes e Validação. Estas etapas foram organizadas por um modelo de processo de *software* baseado em protótipos, conforme ilustrado na Figura 3.1.

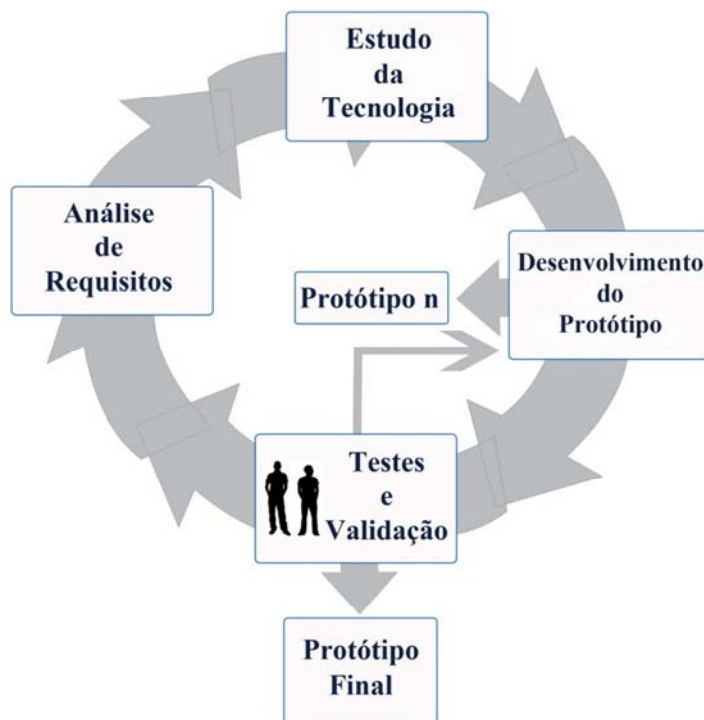


Figura 3.1: Etapas do Processo de Desenvolvimento

Na Análise de Requisitos (Figura 3.1), o usuário especialista da área define um conjunto de objetivos específicos para o *software* que é repassado ao responsável de realizar as etapas do processo de desenvolvimento. Somando esses objetivos, têm-se o objetivo geral que definirá o funcionamento do protótipo final, ou seja, o *software* pronto para uso. Em Estudo da Tecnologia é estudada qual tecnologia irá atender às necessidades de forma adequada para se cumprir o objetivo específico.

Desenvolvimento do Protótipo (Figura 3.1) é a etapa em que o protótipo é gerado.

Em Testes e Validação, é verificado se o protótipo cumpre o objetivo específico da análise de requisitos. Neste caso, o *software* é validado e passa novamente para a Análise de Requisitos, e um novo objetivo específico é definido. Caso contrário, o protótipo volta à etapa de Desenvolvimento do Protótipo para ser corrigido até que cumpra o objetivo específico. A validação do protótipo é feita por usuários da área, que relatam possíveis dificuldades ou problemas com o protótipo.

O processo fica em *loop*, conforme indicado pela Figura 3.1, até que sejam alcançados todos os objetivos. Com isso, na etapa de Testes e Validação, após serem alcançados todos os objetivos específicos, é gerado então o protótipo final.

### 3.1.1. Análise de Requisitos

Análise de Requisitos é a primeira etapa do processo de desenvolvimento e uma das mais importantes, pois, nela são identificadas as funcionalidades do *software* e, com isso, todos os problemas que poderão surgir na implementação, assim como, a tecnologia que será utilizada para codificação do *software*. Todas as informações relevantes ao processo de desenvolvimento do *software* são obtidas através de entrevistas com o usuário especialista da área, que irá utilizá-lo.

O *software* desenvolvido é uma interface gráfica de usuário, que funciona na plataforma GNU/Linux com ambiente *desktop* KDE. Além da interface, foi desenvolvido um simulador numérico, que tem como base o método numérico LN-FDTD em coordenadas gerais e também um visualizador de malha estruturada 3D (tridimensional) [3,14].

O *software* é composto por três módulos distintos, porém interligados conforme a Figura 3.2, seguindo o modelo desenvolvido em [14].

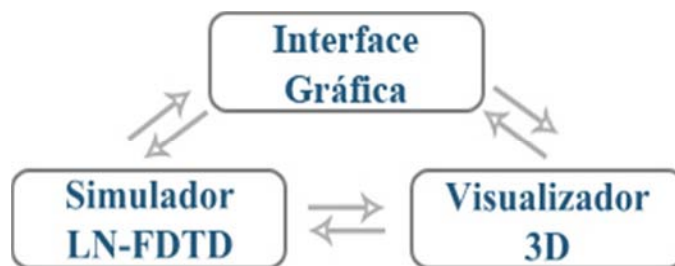


Figura 3.2: Conexão entre os módulos do software.

#### 3.1.1.1. Interface Gráfica de Usuário GUI

Interface Gráfica de Usuário GUI (*Graphical User Interface*) pode ser definida como a forma de interação do usuário com um programa, por meio de uma representação gráfica.

Para construção da interface gráfica, é aqui utilizada a ferramenta de desenvolvimento de interfaces Qt [12], que é multi-plataforma desenvolvida em C++. A mesma é detalhada no tópico 3.1.2.1.

O objetivo principal da construção da interface gráfica é facilitar a realização de simulações baseadas no método LN-FDTD em coordenadas gerais. Todas as suas funções são

projetadas com base nas necessidades do usuário da área de engenharia, o que deixa a interface gráfica intuitiva e de fácil uso.

### **3.1.1.2. Simulador Numérico LN-FDTD**

O simulador numérico deve conter toda a formulação matemática do método numérico LN-FDTD em coordenadas gerais e aceitar arquivos de entrada oriundos da interface gráfica fornecida pelo usuário. Os dados de saídas são gerados na forma de gráficos, vídeos e texto (.dat). O usuário pode decidir se usará ou não processamento paralelo em sua simulação.

Devido ao fato do simulador realizar cálculos matemáticos muito complexos, tempo de processamento alto e grande quantidade de memória RAM são exigidos (algumas vezes mais do que um único computador suportaria, se utilizado o processamento sequencial). Mas isso está intimamente ligado ao número de células usadas para solucionar o problema a ser simulado e cabe ao usuário decidir quantos processadores (se opção de processamento paralelo estiver disponível) serão usados na simulação.

### **3.1.1.3. Visualizador Tridimensional**

O visualizador tridimensional deve possuir a capacidade de ler uma malha estruturada e representá-la no monitor, possibilitar a inserção de objetos no cenário 3D pelo usuário através da interface gráfica e, o mais importante, o visualizador 3D deverá estar em total sincronia com o simulador numérico, de forma que o que for mostrado pelo visualizador 3D deverá ser exatamente o que vai ser simulado pelo simulador numérico [14].

## **3.1.2. Estudo da Tecnologia**

A tecnologia utilizada para o desenvolvimento do *software* está diretamente ligada às necessidades do *software* repassadas pelo usuário especialista. Como nem todos os desenvolvedores conhecem todas as tecnologias existentes, houve a necessidade de definir a tecnologia mais adequada às funcionalidades e parâmetros para a implementação do *software*, e a partir daí obter o domínio da tecnologia a ser utilizada.

A seguir é detalhada a etapa de Desenvolvimento do Protótipo.

### 3.1.3. Desenvolvimento do Protótipo

Nesta etapa são gerados os protótipos dos módulos que foram definidos na etapa de Análise de Requisitos. Com isso, é descrito como foi realizado o desenvolvimento dos módulos propostos.

Os três módulos são apresentados e demonstrações de suas funcionalidades são descritas passo a passo, para que na etapa de Teste e Validação o usuário possa validar o *software* de forma adequada.

#### 3.1.3.1. Interface Gráfica de Usuário GUI

Para o desenvolvimento da GUI deste trabalho foi utilizada uma ferramenta e linguagem para implementação de *software* livre (*Open Source*) chamada Qt versão 3.3 [12]. A interface foi construída para ser utilizada na plataforma GNU/Linux com ambiente *desktop* KDE 3 ou superior.

Qt possui um conjunto de ferramentas específicas para a implementação de interfaces gráficas com profissionalismo, eficiência, portabilidade e com grande facilidade na implementação. Qt foi criada pela Trolltech, empresa de *software* internacional com sede em Oslon, capital da Noruega, com escritórios em Brisbane, na Austrália, e em Palo Alto na Califórnia.

Em janeiro de 2008, a Trolltech foi comprada pela Nokia, uma grande empresa de celulares e dispositivos móveis do mercado internacional.

- ***Qt Designer***

No kit de ferramentas que acompanha a multi-plataforma Qt, existe a ferramenta *Qt Designer* (Figura 3.3) que é utilizada para a concepção e implementação de interfaces de usuário. Ela torna mais fácil a realização do *design* da interface do usuário com suas ferramentas de *layout*, que realizam facilmente a posição e redimensionamento de seus

*widgets* (controles na terminologia do *Windows*) automaticamente em tempo de execução [12].

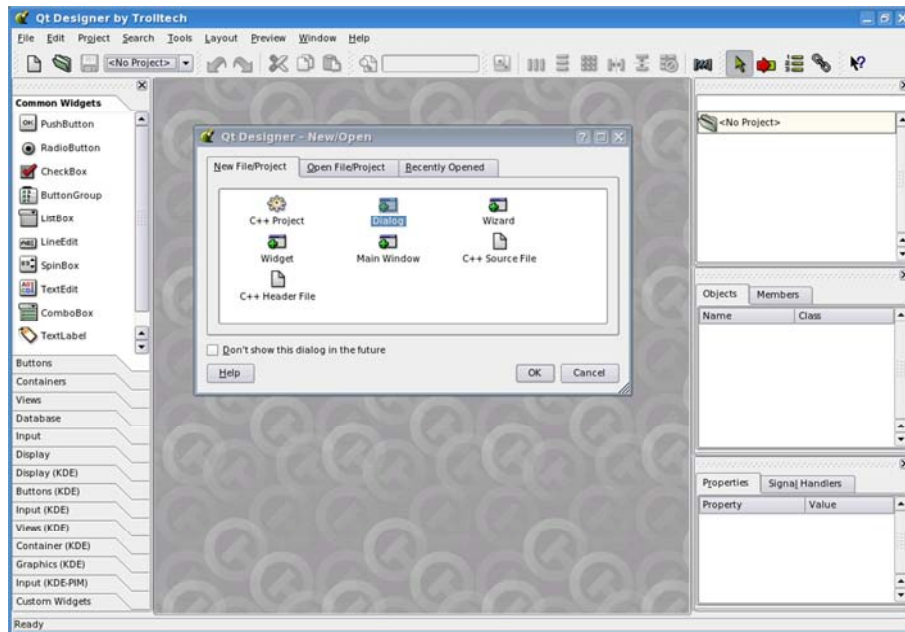


Figura 3.3: Qt Designer

No Apêndice A, é encontrado o passo a passo para construção de uma interface gráfica usando o *Qt Designer*, tendo o passo a passo como base, fica subentendido como foi feita a interface gráfica deste trabalho, que será apresentada neste capítulo.

- **Interface Gráfica Desenvolvida**

Na Figura 3.4, é observada a tela de abertura do *software* desenvolvido.



Figura 3.4: Tela de abertura do *software*.

Na Figura 3.5, pode ser observada a interface do *software*. A estrutura da interface foi dividida em seis áreas: *Menu Principal*, *Barra de Atalhos*, *Guia do Método Numérico*, *Parâmetros da Simulação e Estruturas*, *Ações e Resultados*, como pode ser observado em suas descrições abaixo.

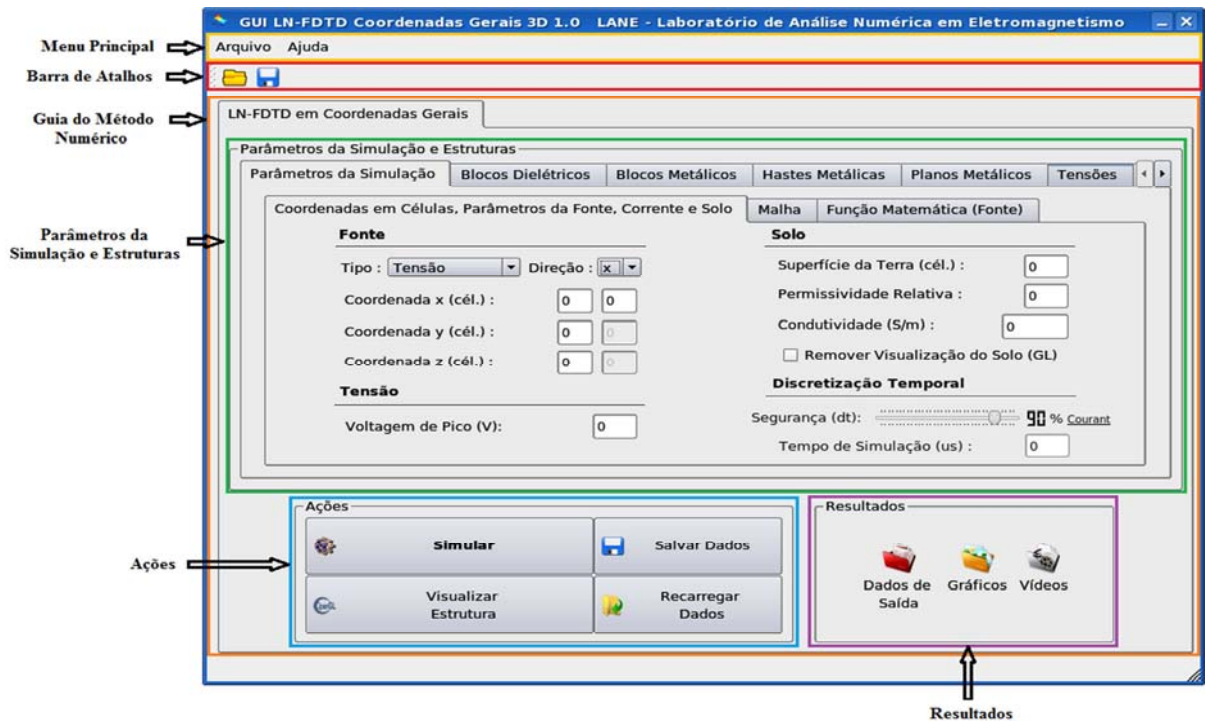
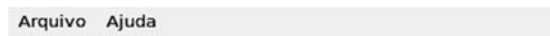


Figura 3.5: Interface Gráfica do *software* e suas áreas.

- **Áreas da Interface Gráfica**

O *Menu Principal* (Figura 3.6a) possui o submenu *Arquivo*, que contém as opções *abrir*, *salvar*, *salvar como* e *sair*. Essas opções estão relacionadas aos arquivos de entrada dos parâmetros da simulação e estruturas. O submenu *Ajuda* possui a opção *Manual* (Apêndice C) e *Sobre*. A opção *Manual* contém todas as informações sobre como utilizar o *software* corretamente e a opção *Sobre* contém informações sobre os desenvolvedores do *software* (Figura 3.6b).



(a)





(b)

Figura 3.6: (a) Menu Principal; (b) Sobre.

A *Barra de Atalho* (Figura 3.7) contém os atalhos para abrir e salvar os arquivos de entrada. É um acesso mais rápido às funções do *Menu Principal* com exceção da opção de *salvar como e sair*.

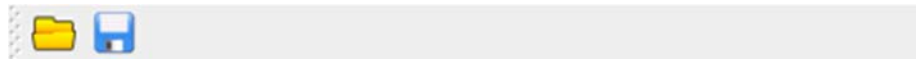


Figura 3.7: Barra de Atalhos.

A *Guia do Método Numérico (LN-FDTD em Coordenadas Gerais)* (Figura 3.8) foi colocada a fim de que, no futuro, agreguem-se novos métodos numéricos enriquecendo as funcionalidades da interface, com isso prevendo possibilidades de expansão do *software*.



Figura 3.8: Guia do Método Numérico (LN-FDTD em Coordenadas Gerais).

- **Parâmetros da Simulação e Estruturas**

*Parâmetros da Simulação e Estruturas* (Figura 3.8 e Figura 3.9) são responsáveis pela inserção dos *Parâmetros da Simulação*, *blocos dielétricos*, *blocos metálicos*, *hastes metálicas*, *Planos Metálicos*, *Tensões* (nas abas da parte superior da Figura 3.9a), *Correntes*, *Distribuição de Campo*, *Campo Elétrico e Processamento Paralelo* (nas abas parte superior da Figura 3.9b).



(a)



(b)

Figura 3.9: (a) Área Parâmetros da Simulação e Estruturas; (b) Área Parâmetros da Simulação e Estruturas.

Na aba *Parâmetros da Simulação* (Figura 3.10), são encontrados os parâmetros do tipo da fonte (fonte de tensão, fonte de corrente ou fonte de tensão pontual), as coordenadas da fonte, os parâmetros do solo, a margem de segurança para  $\Delta t$  (Courant), o tempo de simulação, a voltagem de pico (V) (se for selecionada a opção para fonte de tensão) e a corrente de pico (A) (se for escolhida a fonte de corrente).



Figura 3.10: Aba Parâmetros da Simulação.

A aba *Coordenadas em Células, Parâmetros da Fonte e Solo* são compostas por:

- **Fonte**

A fonte poderá ser de dois tipos: tensão ou corrente.

Em ambos os casos, o usuário poderá escolher a direção da fonte nas três opções ( $x$ ,  $y$  ou  $z$ ), além do seu valor de pico.

- **Solo**

Os parâmetros do solo são compostos pelo plano que define a sua superfície (coordenada z em célula), a permissividade elétrica relativa  $\epsilon_r$  do solo e a sua condutividade elétrica  $\sigma$ .

A opção de *Remover a Visualização do Solo (GL)* é usada somente pelo visualizador tridimensional, em que o usuário pode decidir se quer ou não visualizar o solo no ambiente tridimensional. Para o simulador numérico, o solo não será alterado.

- **Discretização Temporal**

Existem mais dois parâmetros que são: *Segurança (dt)* Figura 3.10 em que o usuário pode ajustar, para realizar a sua simulação, a margem de segurança do seu  $\Delta t$  (Percentual do limite de Courant, dado por (2.12)); e o *Tempo de Simulação (us)*, que é definido pelo usuário de acordo com a sua simulação. Esse tempo é expresso em microssegundos ( $\mu s$ ).

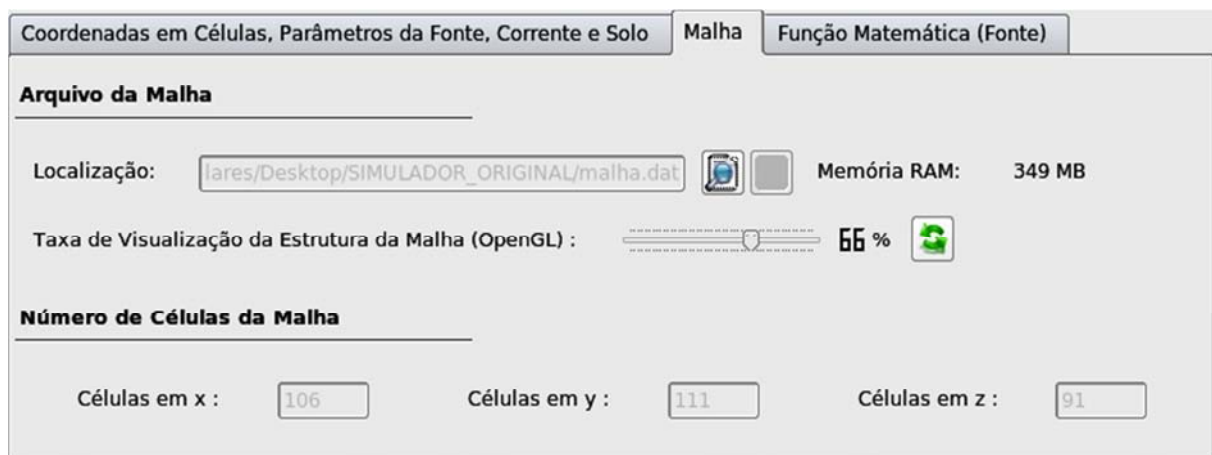



Figura 3.11: Aba Malha.

Na aba *Malha*, o usuário pode abrir o arquivo contendo os dados da malha, referente à sua simulação, utilizando o botão e selecionando o  arquivo da malha, que deverá ter sido criado externamente.

Vale ressaltar que o *software* requer uma malha estruturada como entrada. Esta malha, uma vez criada, deverá ser salva seguindo o modelo do código (Linguagem C) abaixo.

```
//escrita do arquivo da malha.dat
FILE *f_malha = fopen("malha.dat", "w+");
rewind(f_malha);
fprintf(f_malha, "%d %d %d\n", taxa, taxa, taxa);
fprintf(f_malha, "%d %d %d %f\n", nx, ny, nz, dx);
nx1=nx+1; ny1=ny+1; nz1=nz+1;
for(k=1;k<=nz1;k++){
for(j=1;j<=ny1;j++){
for(i=1;i<=nx1;i++){
fprintf(f_malha, "%f %f %f\n", x[i][j][k], yt[i][j][k], z[i][j][k]);


}}}
fclose(f_malha);
```


O bloco de código acima demonstra o padrão de escrita do arquivo malha.dat. É importante frisar a ordem em que são realizados os três laços *for*, pois o visualizador 3D e o simulador numérico realizam a leitura da malha nesta mesma sequencia. Portanto, se a malha for escrita em disco de outra maneira, a simulação não irá funcionar de forma adequada.

Na aba *Malha*, existe a informação chamada *Memória RAM*. Após o usuário abrir a malha, será calculada a quantidade de memória RAM necessária para que a simulação possa ser realizada. Com isso o usuário poderá tomar a decisão se irá ou não utilizar o processamento paralelo se o *software* oferecer esta opção.

O parâmetro *taxa*, serve para ajustar a visualização do cenário tridimensional no visualizador 3D (escala relativa ao nível de detalhes de visualização no cenário virtual gerado a partir da malha), sendo seu valor máximo igual a 99.

Os parâmetros *nx*, *ny* e *nz* são parâmetros do tipo inteiro que determinam a quantidade de células que a malha estruturada possui em cada dimensão; *dx* ( $\Delta x$ ) é uma variável do tipo *float* (real), que é o tamanho médio das arestas das células usadas para a criação da malha.

Voltando a Figura 3.11, após a leitura da malha, o *sliderbar* da taxa de visualização irá mostrar o valor do parâmetro *taxa* definida na criação da malha, e o usuário poderá modificar o valor da taxa conforme achar necessário. Feito isso, o usuário deverá clicar no botão salvar  que irá ser habilitado após a leitura da malha.

Se o usuário achar necessário alterar a taxa de visualização, ele poderá utilizar o botão alterar  e com isso, ele poderá efetuar a alteração e clicar no botão salvar que irá ser disponibilizado no lugar do botão alterar.

A última informação da aba *Malha* são os números de células da malha. Tais dados são mostrados assim que o usuário abre a malha com o botão abrir. A interface não deixa o usuário alterar tais informações, pois as mesmas são definidas durante o processo de criação da malha.

O *software* parte do pressuposto que a malha informada na interface está correta e o usuário não a altera em nenhum momento. O único parâmetro do arquivo da malha que pode ser alterado pelo *software* é a taxa de visualização. Porém, este parâmetro não é utilizado pelo simulador numérico, mas apenas pelo visualizador tridimensional. Com isso, a estrutura da malha não é alterada.

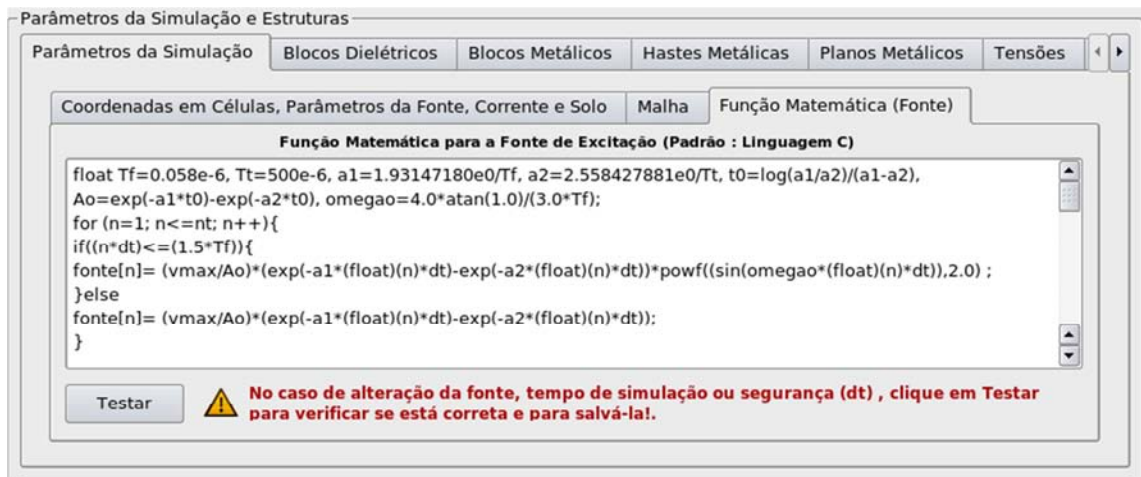


Figura 3.12: Aba Função Matemática (Fonte).

A Figura 3.12 demonstra a aba *Função Matemática (Fonte)*, na qual a função relativa à fonte de excitação usada pela a simulação ser realizada é definida através de um código no padrão da linguagem C, o que garante grande flexibilidade ao software.

- **Blocos Dielétricos**

Na aba *Blocos Dielétricos* (Figura 3.13), são inseridas as informações, tais como: os parâmetros eletromagnéticos  $\mu$ ,  $\epsilon$  e  $\sigma$ , relativos aos objetos que comporão o cenário de simulação, além de suas dimensões e posicionamento na malha computacional.

Cada linha da tabela da Figura 3.13 corresponde a um objeto inserido e sua respectiva cor (padrão RGB) deverá ser fornecida. Esses parâmetros serão utilizados tanto pelo

simulador como pelo visualizador tridimensional, com exceção dos parâmetros de cores, usados somente para representar os objetos no cenário tridimensional.

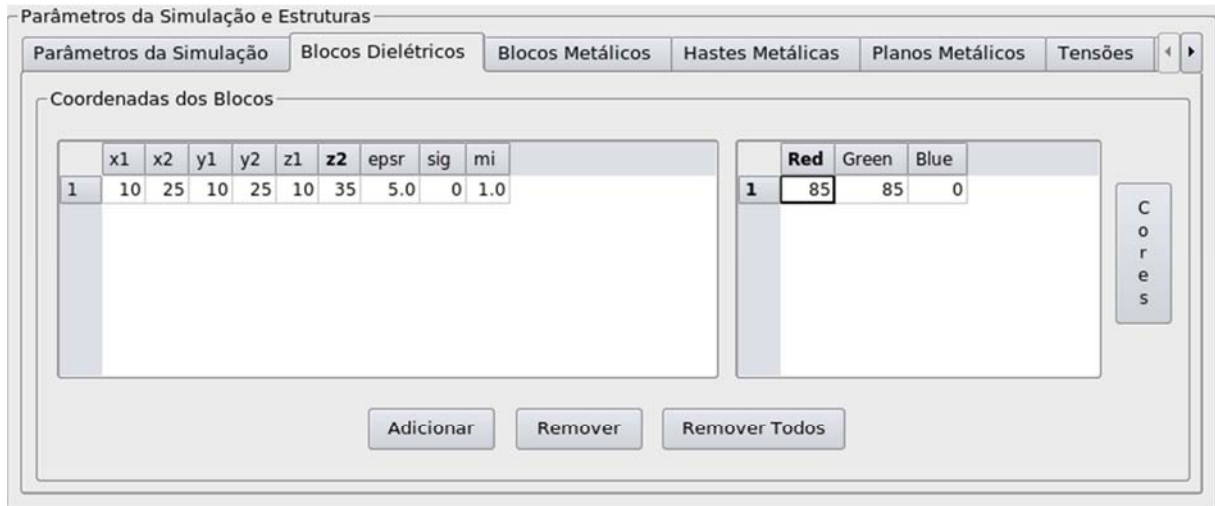


Figura 3.13: Aba Blocos Dielétricos.

É importante ressaltar que, cada linha da tabela da Figura 3.13, corresponde a um bloco dielétrico que será representado no ambiente de simulação, tanto para o simulador numérico, quanto para o visualizador tridimensional com exceção dos parâmetros de cores que será utilizado somente pelo visualizador 3D.

- **Blocos Metálicos**

A aba *Blocos Metálicos* (Figura 3.14) segue as mesmas características dos *Blocos Dielétricos*, porém, diferem na cor, cujo padrão é vermelho no visualizador 3D para todos os blocos metálicos definidos e, também diferem na ausência dos parâmetros eletromagnéticos, em função de que são assumidos que os blocos metálicos são considerados condutores perfeitos.

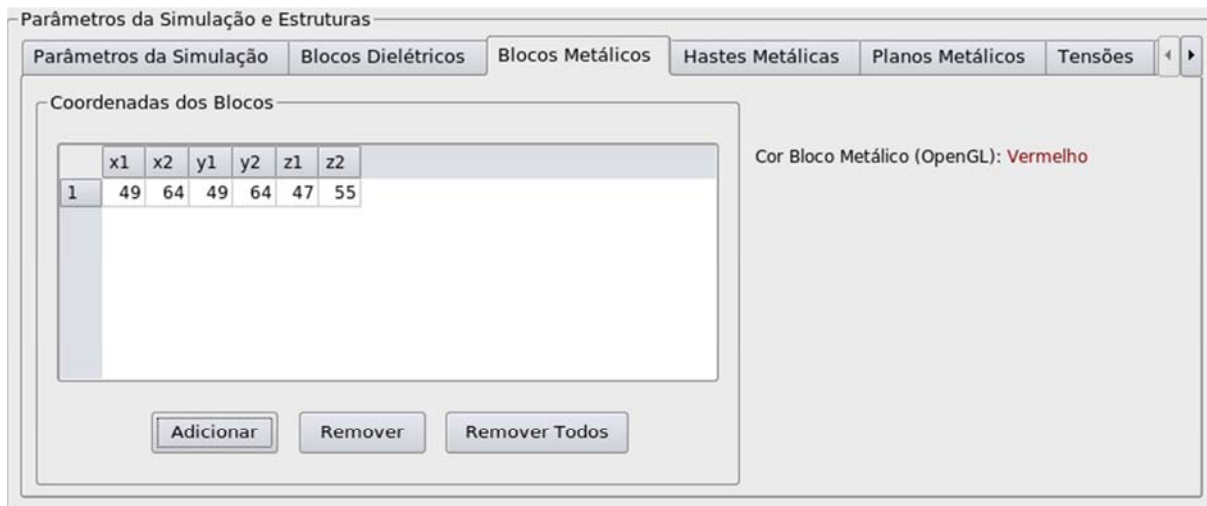


Figura 3.14: Aba Blocos Metálicos.

- **Hastes Metálicas**

Na aba *Hastes Metálicas* (Figura 3.15) o usuário poderá inserir as coordenadas das hastes metálicas finas que comporão a estrutura a ser analisada. Tais hastes possuirão raios menores que as arestas das células, conforme estabelecido em [19,20].



Figura 3.15: Aba Hastes Metálicas.

- **Superfícies Metálicas**

A aba *Planos Metálicos* (Figura 3.16) segue o mesmo padrão dos *Blocos metálicos*, com exceção da sua cor no cenário 3D, que é azul. Inserindo-se zero (0) em uma das coordenadas com índice dois (2), define-se o tipo de plano criado (por exemplo,  $x_2 = 0$  significa plano  $y-z$ ).





Figura 3.16: Aba Planos Metálicos.

- **Tensões**

Na aba *Tensões* (Figura 3.17), o usuário poderá calcular as tensões entre pares de pontos. Após o término da simulação, as tensões poderão ser visualizadas na forma de gráficos, que serão gerados automaticamente pelo simulador numérico.

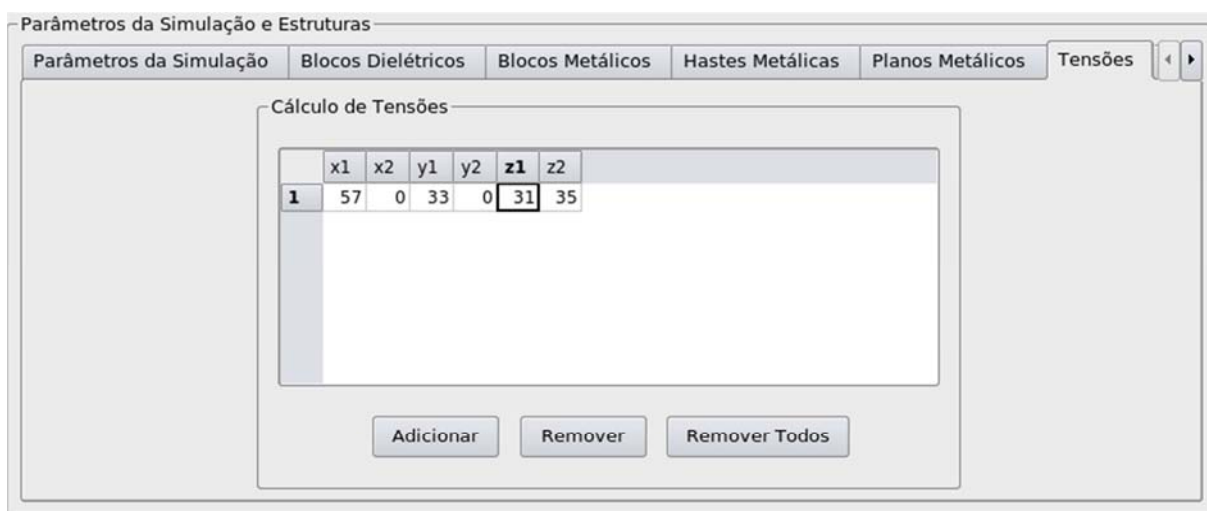


Figura 3.17: Aba Tensões.

- **Correntes**

A aba *Correntes* (Figura 3.18) irá realizar o cálculo de correntes entre pontos escolhidos pelo usuário. No momento em que o usuário for adicionar um cálculo de corrente, a interface irá pedir que o mesmo informe em qual direção ( $x$ ,  $y$  ou  $z$ ) o usuário deseja calcular a corrente, conforme a Figura 3.19.



Figura 3.18: Aba Correntes.

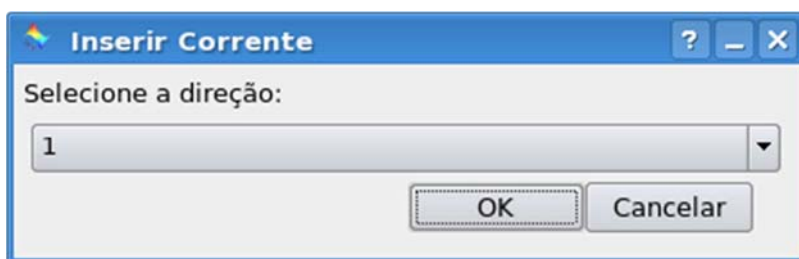


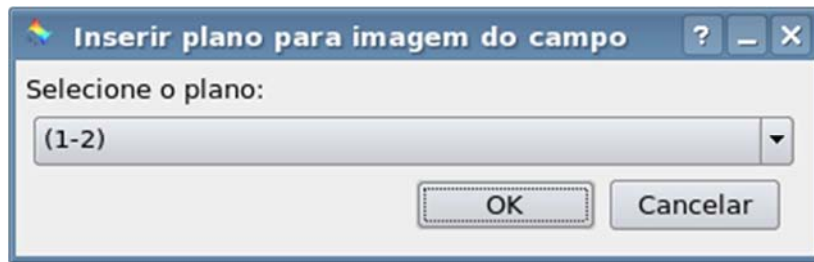
Figura 3.19: Escolha da Direção do Cálculo de Corrente.

- **Campo Elétrico**

Na aba *Campo Elétrico* (Figura 3.20(a)), o usuário poderá gerar imagens e vídeos da propagação do campo elétrico da sua simulação. No momento em que o usuário selecionar o botão *Adicionar*, a interface irá pedir que o mesmo informe em qual plano ((1-2), (1-3) ou (2-3)) o usuário deseja visualizar o campo elétrico, conforme a Figura 3.20(b).



(a)



(b)

Figura 3.20: (a) Aba Campo Elétrico; (b) Escolha do plano para imagem do campo.

O usuário deverá fornecer em qual célula será gerado o campo elétrico e o intervalo de iterações, e se a opção *Gerar Vídeo* estiver habilitado, será gerado ao final da simulação o vídeo da propagação do campo elétrico com as informações retiradas da tabela da Figura 3.20(a).

- **Ações**

A área *Ações* (Figura 3.21) é composta por quatro botões, *Simular*, *Visualizar Estrutura*, *Salvar Dados* e *Recarregar Dados*.

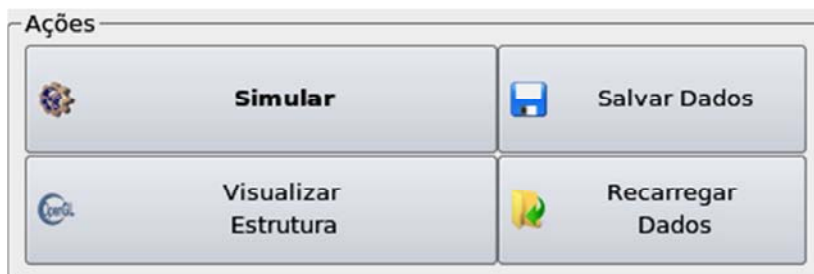


Figura 3.21: Área Ações.

O botão *Simular* irá executar o simulador numérico em outra janela. O botão *Visualizar* irá chamar a execução do visualizador tridimensional (*glmesh*) em outra janela. O botão *Salvar Dados* irá salvar todos os dados de entrada (Parâmetros de Simulação, Blocos Dielétricos, Blocos Metálicos, Hastes Metálicas, Planos Metálicos, Tensões, Correntes, Distribuição de Campo e Campo Elétrico) todos de uma vez. O último botão *Recarregar Dados* será usado quando o usuário editar os arquivos de entrada manualmente, e desejar carregar esses arquivos para a interface sem ter que abrir um arquivo por vez.

- **Resultados**

A área *Resultados* possui as pastas *Gráficos*, *Dados de Saída* e *Vídeos* (Figura 3.22).



Figura 3.22: Área Resultados.

A pasta *Dados de Saída* contém todos os arquivos de saída gerados pelo simulador. A pasta *Gráficos* contém todos os gráficos gerados pela simulação. Finalmente, a pasta *Vídeos* contendo os vídeos gerados pela simulação.

Por fim, é mostrada pela Figura 3.23 uma visão geral do sistema desenvolvido através de um fluxograma. O *software* é composto por: interface (GUI), visualizador (3D) e simulador (LN-FDTD). A seguir, cada elemento é descrito em mais detalhes.

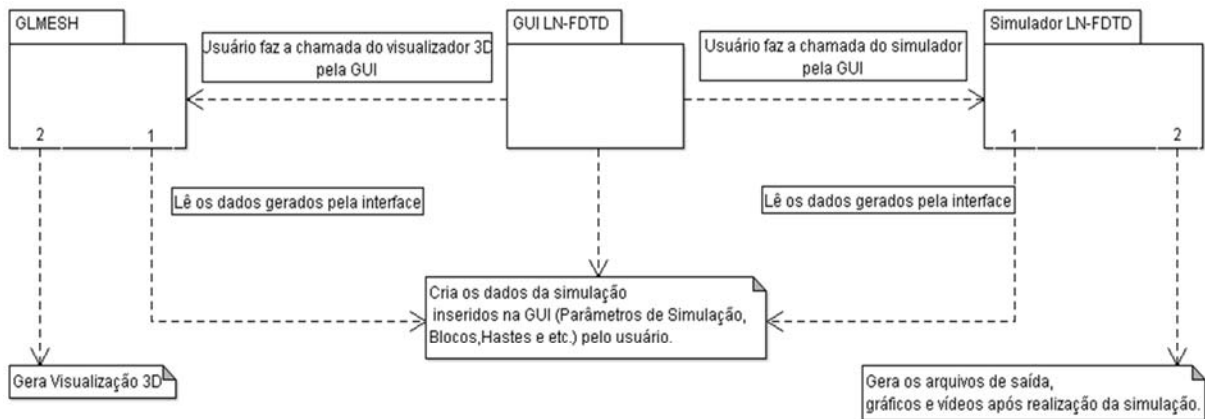


Figura 3.23: Visão geral do sistema desenvolvido.

### 3.1.3.2. Fluxograma do simulador numérico baseado no método LN-FDTD

Para o melhor entendimento da implementação do simulador numérico desenvolvido, é apresentado um fluxograma (Figura 3.25) demonstrando o algoritmo do mesmo.

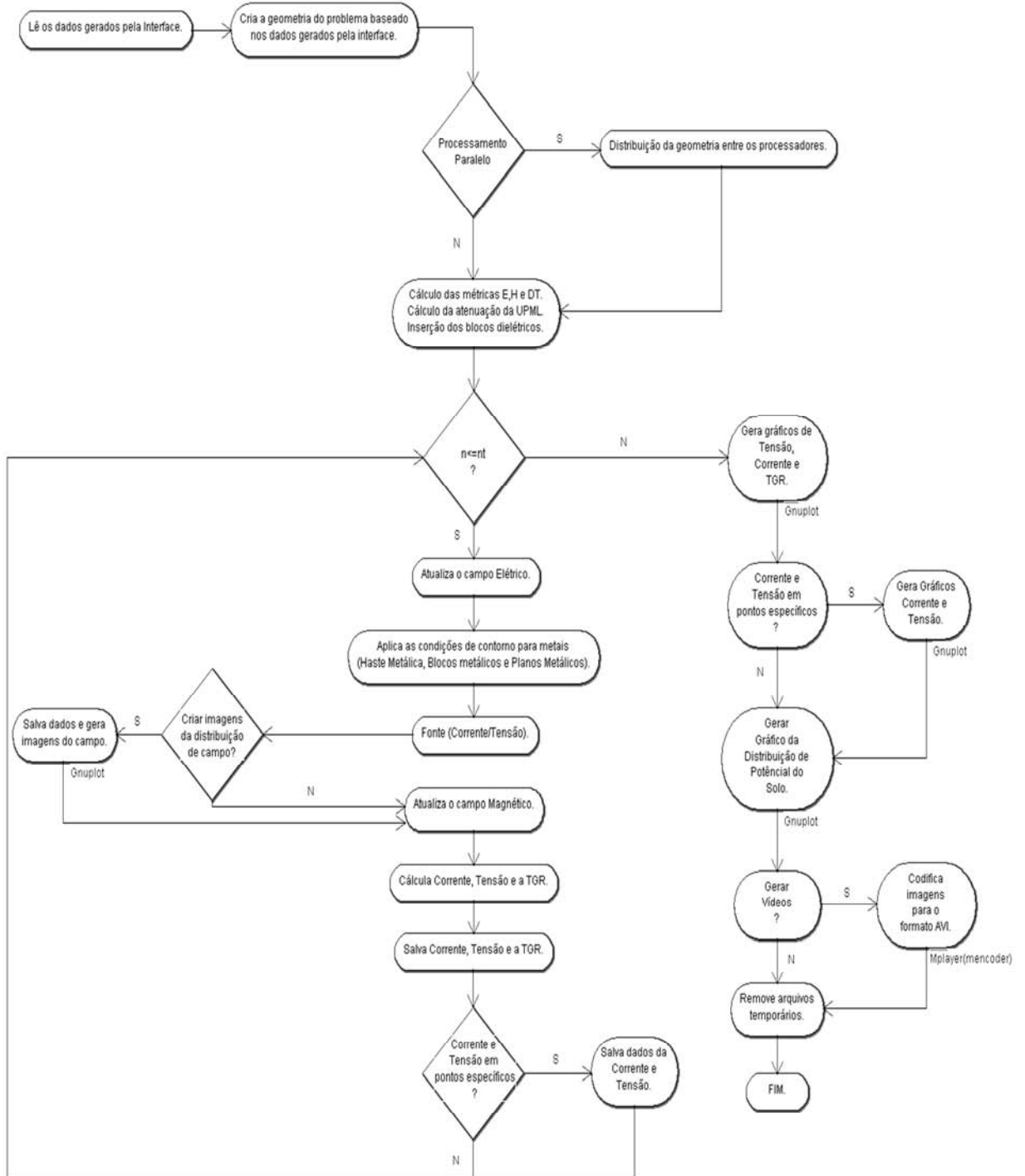


Figura 3.24: Algoritmo do simulador LN-FDTD.

Vale ressaltar que o simulador numérico foi desenvolvido utilizando a linguagem de programação C. Ele utiliza o *software* Gnuplot para geração de gráficos 2D e 3D. Também é utilizado o *software* MPlayer, que é não só um *player* multimídia, mas também possui funcionalidades para a criação de vídeos. Ambas as ferramentas complementares utilizadas pelo simulador são *software* livre.

### 3.1.3.3. Visualizador Tridimensional

Há alguns anos, o desenvolvimento de aplicações gráficas de alto desempenho era restrito a instituições de pesquisas ou de empresas que dispunham de *hardware* veloz e específico para aplicações gráficas, além de programadores experientes. Atualmente essa situação mudou devido ao surgimento das placas gráficas para computadores pessoais e ao desenvolvimento de bibliotecas gráficas que não exigem conhecimento extensivo de programação gráfica ou sobre o *hardware* utilizado. Uma dessas bibliotecas, hoje padrão da indústria para aplicações gráficas profissionais, é a *OpenGL* [13].

Em [13] *OpenGL* é definida como uma biblioteca gráfica aberta e multi-plataforma, que possui rotinas gráficas e de modelagem utilizada para o desenvolvimento de aplicações de computação gráfica, tais como jogos, sistemas de visualização, criações de ambientes tridimensionais entre outros.

A *OpenGL* é gerenciada por um grupo independente formado em 1992, o ARB (*Architecture Review Board*). Este grupo hoje em parceria com a *Khronos Group*, grupo fundado em janeiro de 2000 por uma série de empresas líderes na área, entre as quais estão a *3Dlabs*, *ATI*, *Discreto*, *Evans & Sutherland*, *Intel*, *NVIDIA*, *SGI* e *Sun Microsystems*, são responsáveis pela criação e aprovação de novas funcionalidades, versões e extensão de *OpenGL* [21]. A *OpenGL* está atualmente na versão 4.1 e está disponível para *download* em [21].

Em síntese, *OpenGL* é uma poderosa API (*Application Programming Interface*) para desenvolvimento de aplicações gráficas bidimensionais (2D) e tridimensionais (3D) para diferentes plataformas. Seu funcionamento é semelhante ao de uma biblioteca da linguagem de programação C. Logo, quando se diz que um programa é baseado em *OpenGL* ou é uma aplicação *OpenGL*, significa que foi escrito em alguma linguagem de programação (C, C++, Java, dentre outras), e que no programa está sendo utilizada uma ou mais bibliotecas *OpenGL* [13].

No Apêndice B é descrito como desenvolver uma aplicação gráfica 3D utilizando *OpenGL*. O exemplo no Apêndice B serve como base para desenvolver qualquer aplicação 3D utilizando a biblioteca *OpenGL*, seja esta simples ou complexa.

- **Glmesh**

O visualizador tridimensional desenvolvido foi chamado de *gmesh*, pois além deste

possibilitar a visualização das estruturas, possibilita a visualização de malhas computacionais ou de ambas, simultaneamente. Durante a configuração dos parâmetros de simulação e após a abertura do arquivo da malha pela interface, o usuário poderá visualizar a estrutura a ser simulada clicando no botão *Visualizar Estrutura* (Figura 3.21).

Como exemplo de visualização de uma estrutura, é mostrada uma malha retangular de  $80 \times 80 \times 80$  células. O tamanho da aresta de uma célula será igual a 0,5 m (Figura 3.25). Na Figura 3.25, podemos observar a parte externa da malha e a visualização do solo: essa será a visão inicial da malha recém-construída mostrada pelo glmesh.

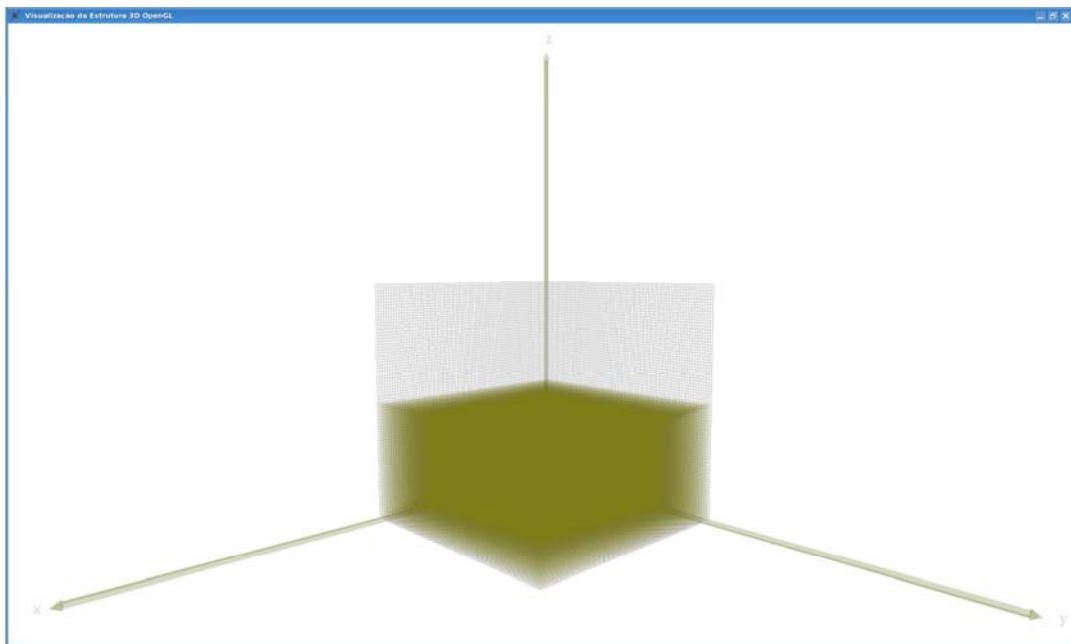


Figura 3.25: Glmesh visão da casca da malha.

O glmesh possui funcionalidades para que o usuário possa inserir seus objetos no cenário 3D, de acordo com suas necessidades; todas as funcionalidades são ativadas por teclas do teclado:

Tecla *m*: Ativa ou desativa a visualização da parte externa da malha.

Teclas *i*, *j* e *k*: Retiram uma linha da parte inicial da malha conforme a direção escolhida, tecla *i* = direção *x*, tecla *j* = direção *y*, tecla *k* = direção *z*.

Teclas *Shift + i*, *Shift + j* e *Shift + k*: Esconde uma linha da parte final da malha conforme a direção escolhida, tecla *i* = direção *x*, tecla *j* = direção *y*, tecla *k* = direção *z*. Isto permite que o usuário visualize e analise parte do volume da malha. A Figura 3.26 ilustra esta funcionalidade.

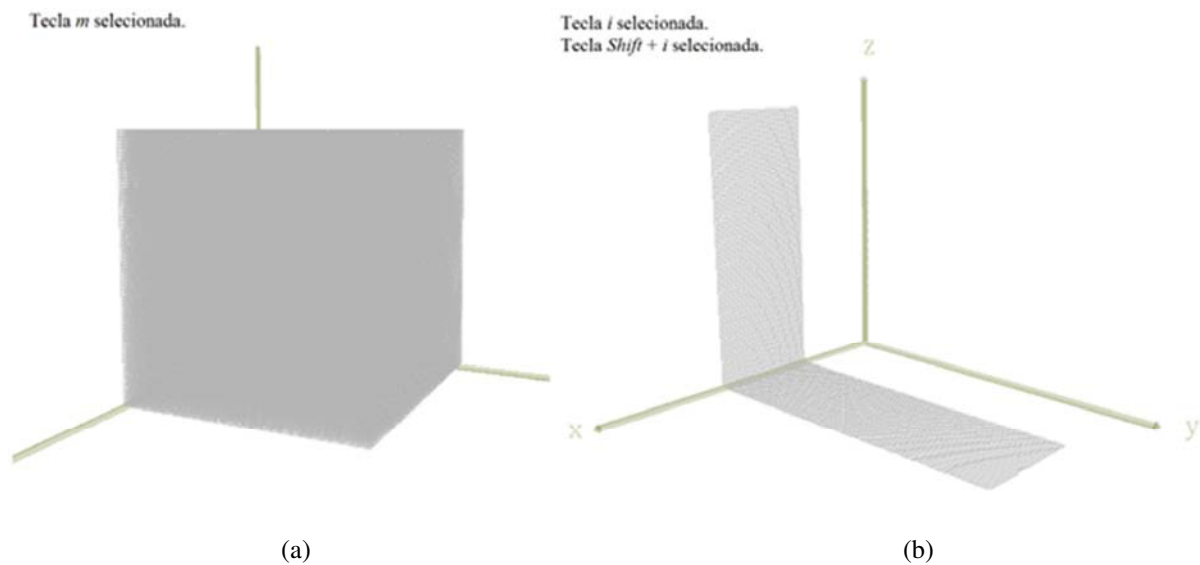


Figura 3.26: (a) Demonstração da funcionalidade da tecla *m* ativada e (b) funcionalidades das teclas *i* e *Shift + i*.

Quando associada às teclas *x*, *y* e *z*, a tecla *s* (= *select*), se selecionada pelo usuário, mostra apenas uma superfície da malha e a percorre superfície por superfície na direção escolhida. Quando a tecla *shift* é ativada, a malha é varrida em sentido contrário. A Figura 3.27 demonstra a funcionalidade das funções descritas acima para a tecla *x*. Isto permite que o usuário visualize a malha superfície a superfície.

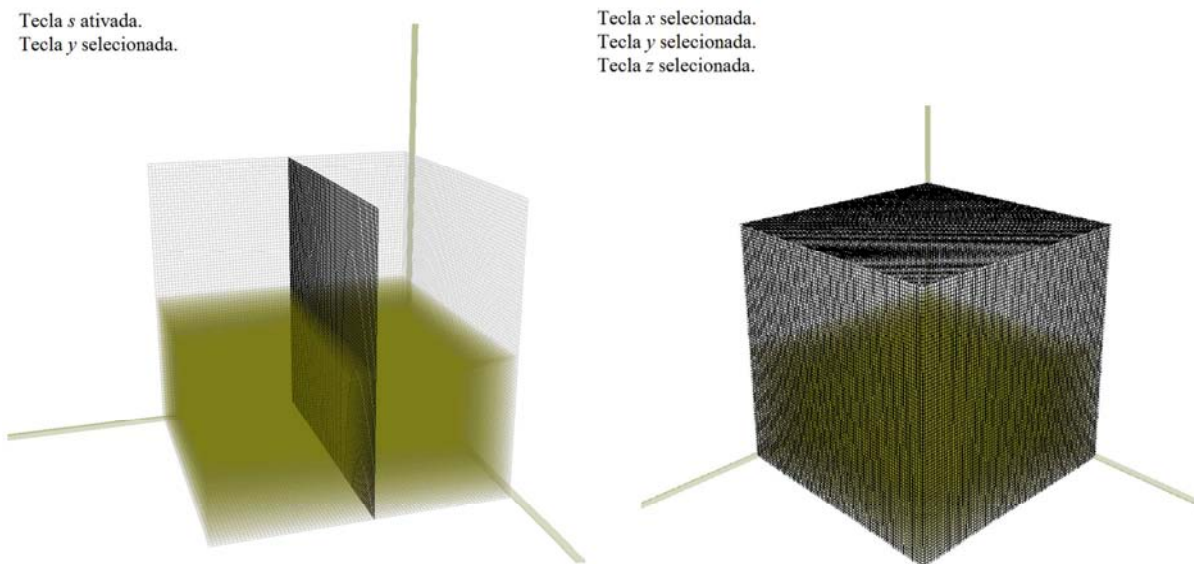


Figura 3.27: Funcionalidades das tecla *s* associada à tecla *x*.

A última funcionalidade do *software* *glmesh* é a tecla *r*. Se o usuário esconder linhas da malha com as teclas *i*, *j* ou *k* ou com *Shift + i*, *j* ou *k*, basta pressionar a tecla *r* para que as



linhas da malha voltem ao seu estado de visibilidade inicial, conforme mostrado na Figura 3.28.

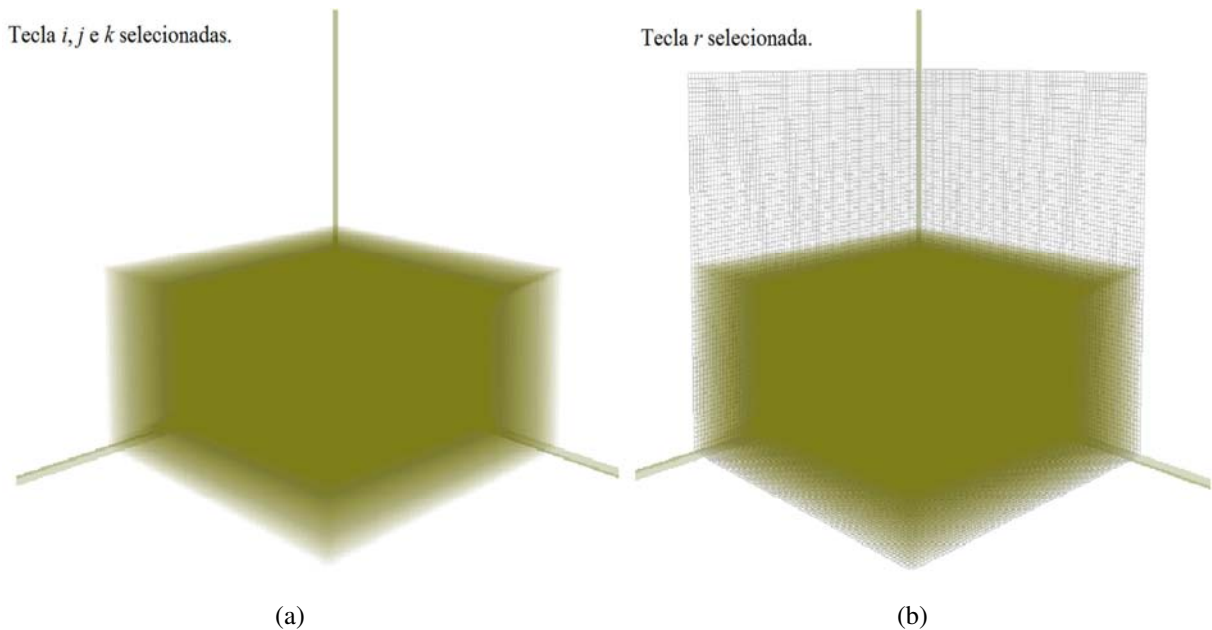


Figura 3.28: Funcionalidade da tecla  $r$ : (a) todas as linhas da malha foram escondidas; (b) visualização restaurada da malha.

Além das funcionalidades que envolvem teclado, o glmesh possui funcionalidades que envolvem o *mouse*. O usuário pode investigar cada detalhe do cenário 3D utilizando funcionalidades como *zoom* (com o botão direito do *mouse* ou com o seu *scroll*), rotação (com o botão esquerdo do *mouse*) e translação (botão do meio).

Além das funcionalidades de visualização da malha, apresentadas acima, o módulo glmesh representa objetos no cenário 3D, os quais são inseridos através da interface construída em Qt (GUI) utilizando as tabelas relativas aos blocos (dielétricos e metálicos), hastes metálicas e planos metálicos.

A Figura 3.29 mostra um conjunto de hastes metálicas utilizando a malha computacional mostrada pela Figura 3.28(b).

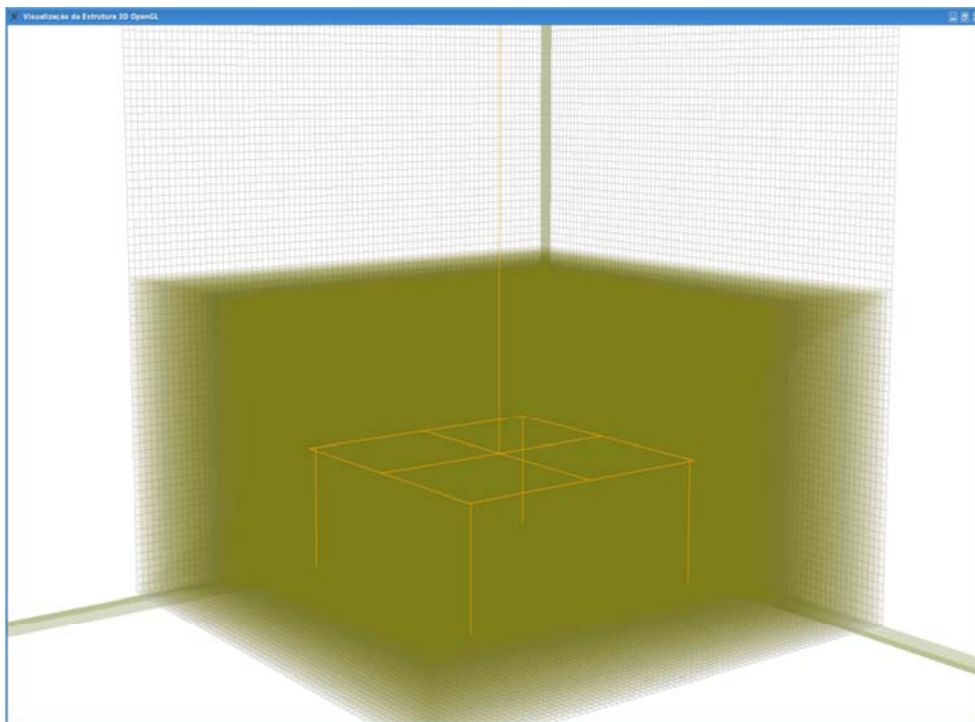


Figura 3.29: Conjunto de Hastes Metálicas.

Na Figura 3.30, é mostrado um bloco metálico (em vermelho) com uma haste metálica fina (em amarelo) que passa pelo seu centro, ainda utilizando a malha computacional mostrada pela Fig. 3.28(b).

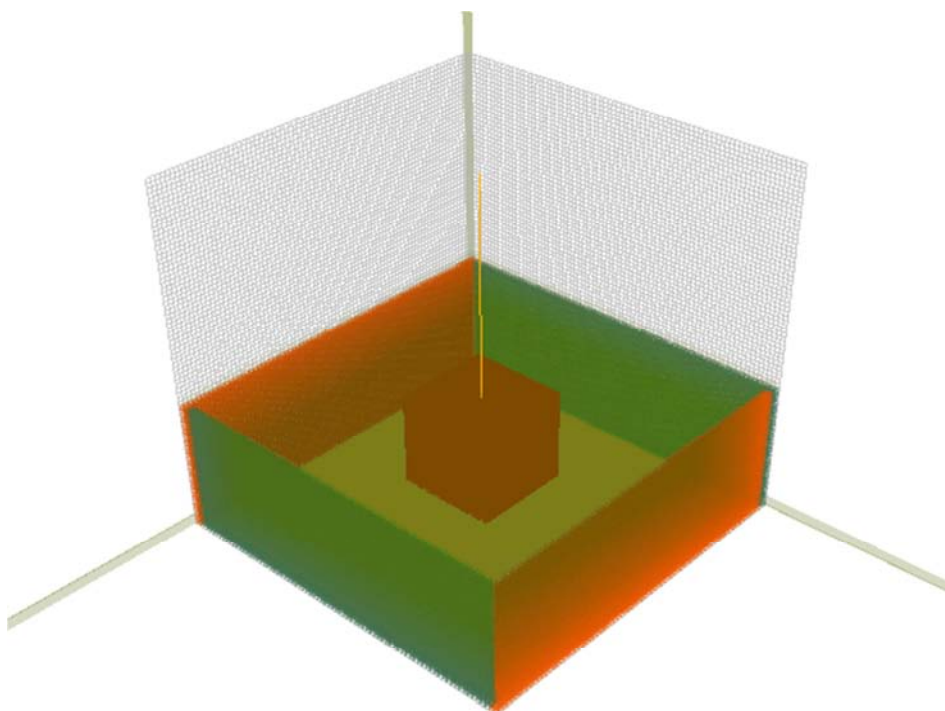


Figura 3.30: Bloco Metálico e Blocos Dielétricos.

A Figura 3.31 mostra dois planos metálicos (em azul) interligados por uma haste metálica (em amarelo).

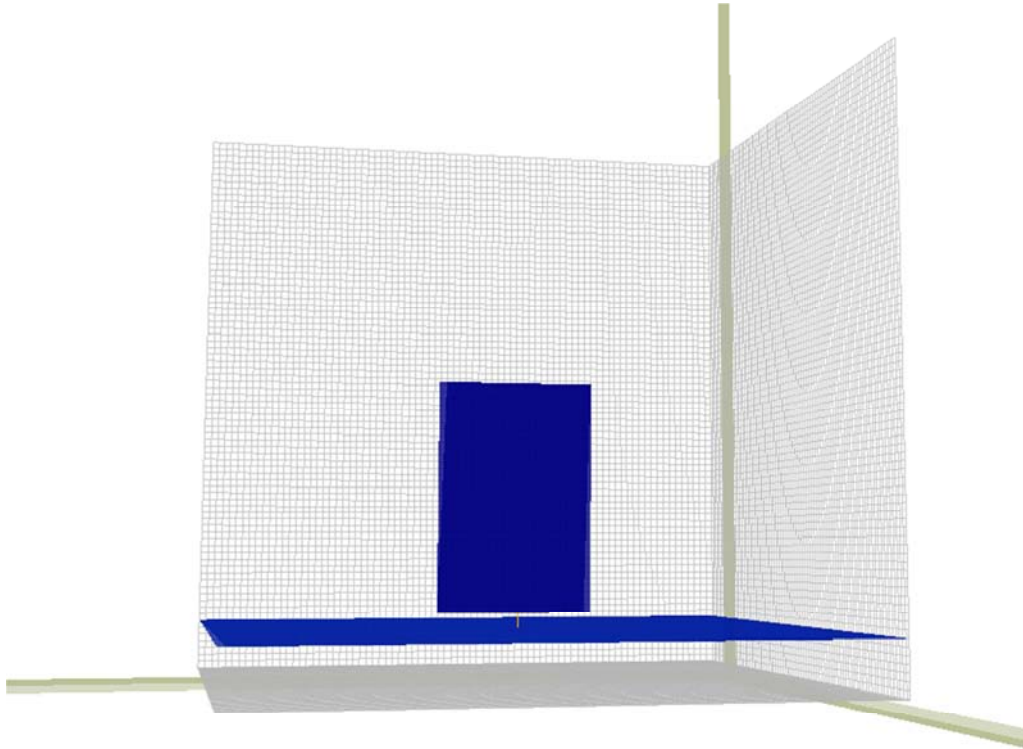


Figura 3.31: Planos Metálicos interligados por uma Haste Metálica.

### 3.1.4. Testes e Validação

A última etapa do processo de desenvolvimento do *software*, Testes e Validação, também é uma das mais importantes, pois, nesta etapa, o protótipo gerado é testado de todas as formas possíveis, a fim de que se solucionem problemas não detectados anteriormente.

Os testes são realizados pelo usuário especialista da área e por uma equipe montada pelo mesmo, composta por alunos de graduação e mestrado do curso de Engenharia Elétrica. Se for reportado algum erro ou problema na utilização, o protótipo volta à etapa anterior para ser corrigido e é submetido a novos testes até que o mesmo seja validado.

Esta etapa é de grande importância, pois, garante a confiabilidade do *software*

desenvolvido e evidencia suas características de qualidade, que serão repassadas e discutidas pela equipe que irá testar o protótipo.

Para fins de comparação e validação, o resultado obtido com o método numérico LN-FDTD em coordenadas gerais o qual é a base deste *software*, é comparado ao resultado obtido com o método numérico FDTD em coordenadas retangulares. Vale ressaltar que em ambos os métodos são utilizados os mesmos parâmetros.

Após realizar vários ciclos da Figura 3.1, é obtido o produto final, o que não significa que o *software* não será mais alterado ou corrigido, pelo contrário, serão realizadas constantes melhorias em toda sua estrutura.

Para validar o *software*, o problema proposto em [22], por Tanabe, foi reproduzido (Figura 3.32). O solo é considerado isotrópico e é caracterizado pelos seguintes parâmetros:  $\sigma = 2,28$  mS/m,  $\epsilon_r = 50$  e  $\mu_r = 1$ . O problema consiste em um eletrodo vertical enterrado, com um comprimento de 3 m, cuja área da secção transversal é de  $0,5 \times 0,5$  m<sup>2</sup>. O eletrodo é alimentado por uma fonte tensão de surto conectada em série com um resistor [22]. O circuito de medição descrito em [22] também foi considerado. Os resultados obtidos apresentam excelente concordância com os obtidos experimentalmente por Tanabe, além de serem perfeitamente coincidentes com os resultados calculados em [22].

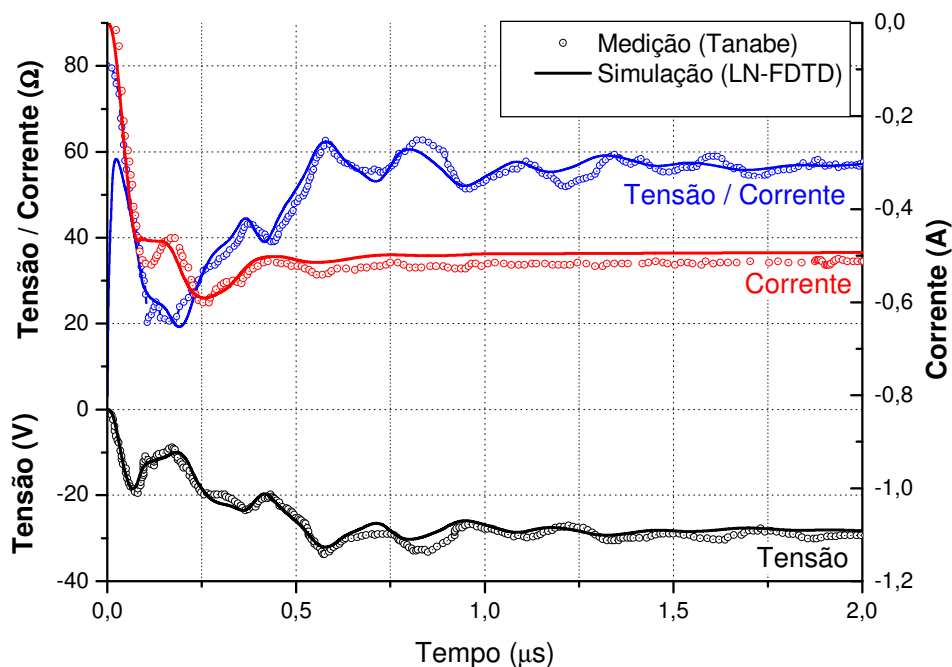


Figura 3.32: Comparações da resposta transiente obtida através do método LN-FDTD, para um eletrodo vertical medindo  $0,5 \times 0,5 \times 3,0$  m, com os dados obtidos em [22].

Para validar o *software* para realizar análises transitórias em coordenadas gerais, foi realizada a simulação de um condutor cilíndrico de aterramento. O solo é caracterizado pelos seguintes parâmetros:  $\epsilon_r = 10$ ,  $\sigma = 0,04$  S/m e  $\mu_r = 1$ . O problema consiste em duas hastes cilíndricas afastadas por 0,25 m, tal como é ilustrado na Figura 3.33. O condutor superior se estende de forma a penetrar na região absorvente UPML (seu comprimento pode ser considerado infinito). A haste de aterramento possui raio  $r = 50$  mm. A simulação calcula a tensão  $V(t)$  através do caminho de integração definido na Figura 3.33. O tempo total simulado é  $t = 0,25 \mu s$ . A excitação do problema é feita impondo-se uma tensão variante no tempo entre os cilindros condutores que obedece a função matemática para  $t \leq T_f$ :

$$V_s(t) = \frac{V_{max}}{T_f} t \quad (3.1)$$

para  $t > T_f$ :

$$V_s(t) = -\frac{V_{max} \cdot t}{10^{-4}} + 1010 \quad (3.2)$$

em que  $V_s(t)$  é a função tensão,  $T_f = 0,1 \mu s$ ,  $V_{max} = 1000$  V.

O problema aqui proposto foi simulado utilizando a técnica numérica FDTD (LANE SAGS) e LN-FDTD (LN-LANE SAGS). No caso da simulação em coordenadas retangulares (FDTD), os cilindros condutores foram modelados pela técnica de fio fino aperfeiçoada por Baba [20], a qual foi introduzida originalmente por Noda [19]. Neste caso,  $\Delta_x = \Delta_y = \Delta_z = 0,25$  m. Para a simulação baseada no sistema de coordenadas gerais, foi construída uma malha cuja secção transversal é mostrada pela Figura 3.34. Foram utilizadas seis células para modelar o diâmetro dos cilindros.

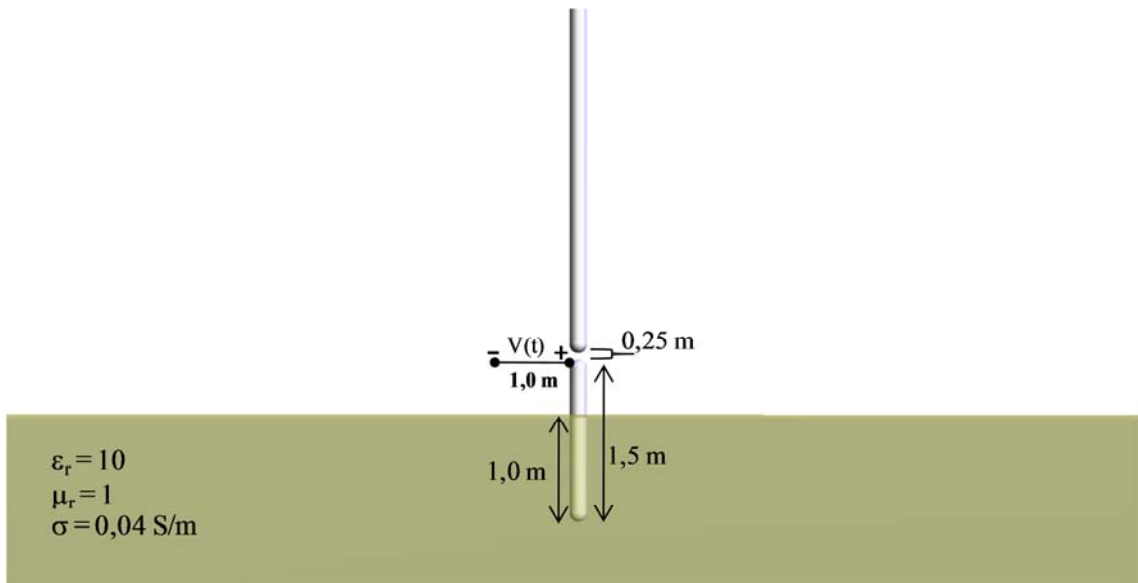


Figura 3.33: Problema proposto para validação do *software* desenvolvido.

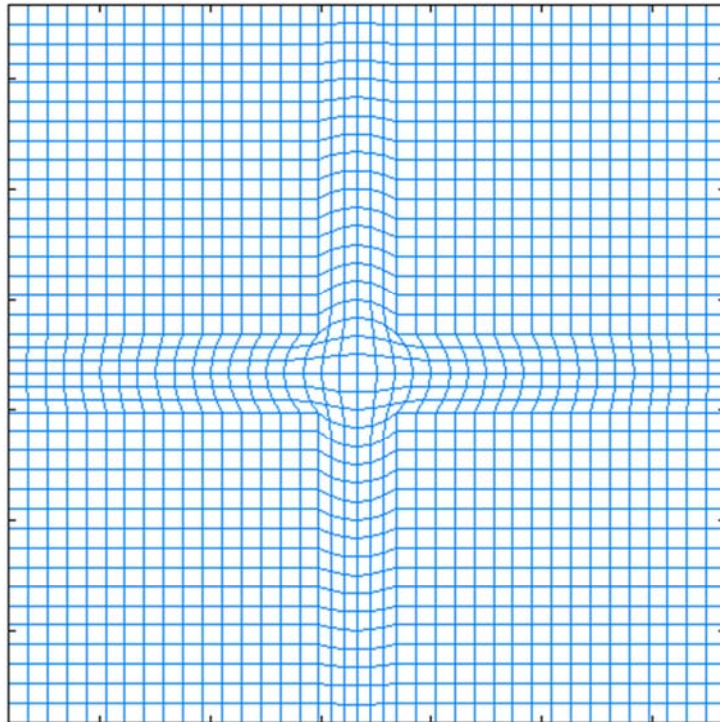


Figura 3.34: Secção transversal (superfície 1-2) da malha não-ortogonal gerada para modelar as hastes.

Os resultados apresentados pela Figura 3.35, obtidos através das técnicas FDTD com fio fino e LN-FDTD, concordam entre si. Vale ressaltar que a formulação de fio fino foi validada experimentalmente por T. Noda em [19]. Dessa forma, a implementação do método LN-FDTD realizada neste trabalho é validada através dos resultados mostrados na Figura 3.35 por comparação indireta.

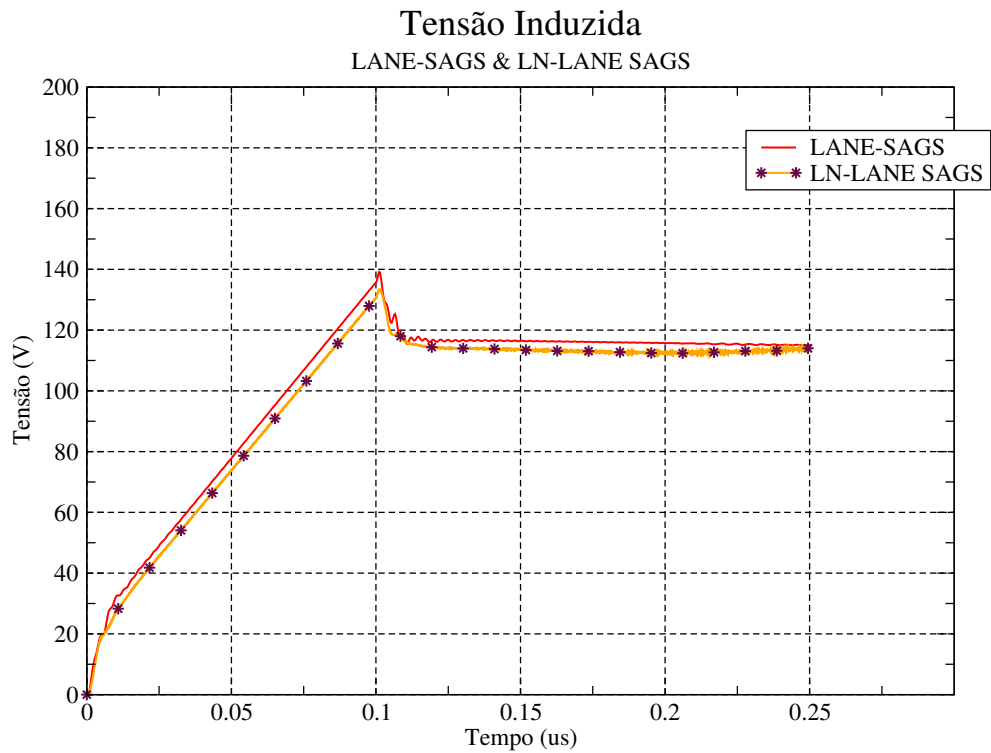


Figura 3.35: Tensão  $V(t)$ : métodos FDTD (fio fino) e LN-FDTD.

O *software* desenvolvido neste trabalho foi utilizado para gerar os resultados publicados em [23-25], os quais podem ser encontrados nos Apêndices C, D e E com exemplos adicionais de validação.

## Capítulo 4 - Resultados

Neste capítulo, são apresentados os resultados obtidos através das simulações realizadas utilizando o *software* desenvolvido neste trabalho, GUI LN-FDTD.

Inicialmente, é feita validação do *software* através de comparações com simulações realizadas utilizando o *software* LANE-SAGS [14], o qual é baseado no método numérico FDTD em coordenadas retangulares (validações relativas a problemas não retangulares são encontradas em [23, 24] e nos Apêndices C e D). Em seguida, são realizadas simulações em coordenadas gerais concebidas a partir do problema de validação. Os resultados obtidos são discutidos e analisados ao longo deste capítulo.

### 4.1. Validação do *Software* e Simulações

A validação do *software* desenvolvido é realizada através do Caso A, apresentado a seguir, a partir do qual os demais exemplos (em coordenadas gerais) são desenvolvidos. Validações relativas a problemas em coordenadas gerais (casos não retangulares) são encontradas em [23,24]. Tais referências estão disponíveis na íntegra nos Apêndices C e D.

#### 4.1.1 - Caso A

Inicialmente, para o Caso A, uma residência foi modelada no *software* GUI LN-FDTD em Coordenadas Gerais 3D (Figuras 4.1 – 4.3), o qual foi desenvolvido neste trabalho. Este modelo foi recriado na ferramenta LANE-SAGS, para efeito de comparação. A malha computacional relativa à estrutura da Figura 4.1 é retangular e possui  $420 \times 272 \times 185$  células, onde  $\Delta = 0,04\text{m}$  (células cúbicas). Tal estrutura é composta por blocos dielétricos ( $\epsilon_r = 5,0$ ,  $\sigma = 0,0$  e  $\mu_r = 1,0$ ) [26] que representam as paredes de concreto, com 0,60m de espessura e 3,88m de altura (Figuras 4.2 e 4.3). O telhado da residência é modelado com o mesmo material. A residência possui também blocos metálicos, que representam vigas de sustentação. Tais vigas são consideradas condutores elétricos perfeitos, possuem 0,20m de espessura e penetram um metro no solo (Figura 4.3).

Na parte exterior da residência, linhas de distribuição foram consideradas (Figuras 4.2 - 4.4). O condutor superior (neutro) foi posicionado a 4,6 m da superfície do solo e o condutor que representa a fase foi suspenso a 4,4 m do nível do terreno. O raio  $r_0$  desses condutores foi



modelado usando o conceito de raio intrínseco apresentado em [20], de tal forma que  $r_o = 0,23 \Delta$ . As extremidades desses condutores penetram na região absorvente UPML, de forma que seus comprimentos podem ser considerados infinitos, tal como proposto em [28]. Conforme indicado pela Figura 4.3, a distância das linhas de distribuição em relação à superfície de concreto mais próximo (pertencente à residência) é de 2,08 m. Ainda observando a Figura 4.3, observa-se que o condutor neutro foi aterrado através de uma haste que penetra um metro no solo. Conforme indicado pela Figura 4.2, os cabos que compõem a instalação elétrica da residência foram modelados com terminações que funcionam como tomadas. Estas tomadas foram posicionadas a um metro da superfície do solo (Figura 4.4) e seus terminais são afastados por 4 cm (Figura 4.5a). A norma NBR 14136 [29] define o afastamento entre fase e neutro como sendo de  $1,9 \text{ cm} \pm 0,2 \text{ mm}$  nas tomadas. Porém, dado o alto custo computacional do método FDTD, optou-se pelo afastamento de 4 cm para viabilizar a investigação.

A residência modelada possui um sistema de proteção contra descargas atmosféricas (SPDA) [32] [33], que consiste de um pára-raios. Este dispositivo está diretamente conectado a estrutura metálica da residência, tal como indicado pela Figura 4.3. A fonte de excitação, que modela a corrente relativa à descarga atmosférica, foi posicionada no pára-raios, a 4,16 m da superfície do solo. O canal da descarga atmosférica foi modelado por um condutor elétrico cilíndrico, posicionado verticalmente, que penetra na UPML (comprimento virtualmente infinito), tal como proposto por [28]. Esta configuração geométrica é ilustrada pela Figura 4.3. Deve-se ressaltar que as características eletromagnéticas da terra seguem [27], sendo estas dadas por  $\epsilon_r = 10$ ,  $\sigma = 0,04 \text{ S/m}$  e  $\mu_r = 1$ .

Neste trabalho, foram calculadas as tensões induzidas entre os terminais relativos a todas as oito tomadas da residência (Figuras 4.2 e 4.5(a)). Além disso, foram calculadas as tensões entre as linhas de distribuição e o solo, tal como indicado na Figura 4.5(b), conforme proposto originalmente por [32]. Vale ressaltar que, neste caso, o terra das tomadas não é considerado.

Como a estrutura da Figura 4.1 é retangular, os resultados obtidos em ambos os *softwares* (LANE SAGS e o simulador aqui desenvolvido) são idênticos, conforme mostrado pelas Figuras 4.6 – 4.10. Ressalta-se que o tempo simulado total é de  $t = 3 \mu\text{s}$ . Com isso, validam-se as sub-rotinas do *software* GUI LN-FDTD relativas à aplicação das condições de contorno, definição dos parâmetros  $\epsilon$ ,  $\sigma$  e  $\mu$ , além da implementação da fonte de excitação. Validações relativas a problemas em coordenadas gerais (casos não retangulares) são

encontradas em [23,24]. Tais referências estão disponíveis na íntegra nos Apêndices C e D.

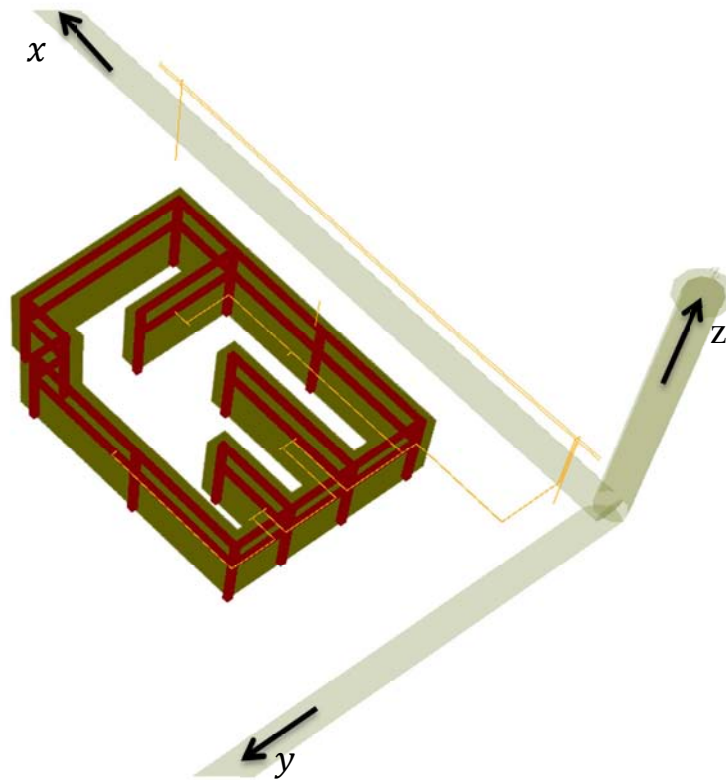


Figura 4.1: Estrutura da residência modelada na GUI LN-FDTD em Coordenadas Gerais 3D (visão sem o teto e o solo).

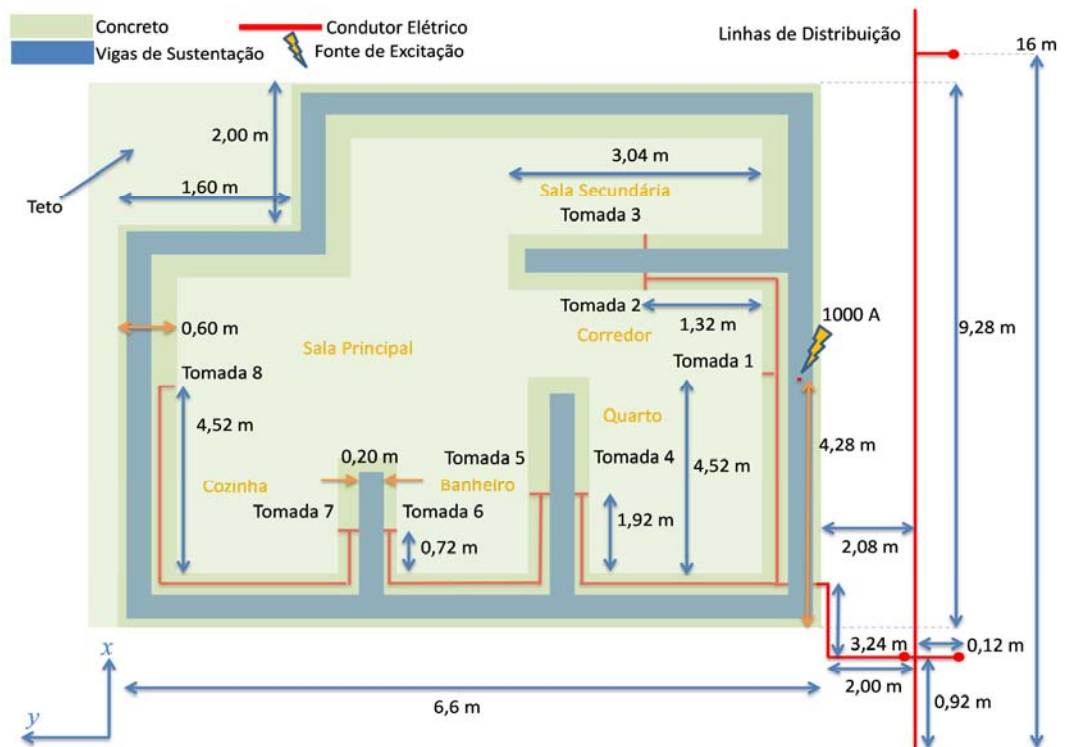


Figura 4.2: Visão superior da residência para o Caso A.

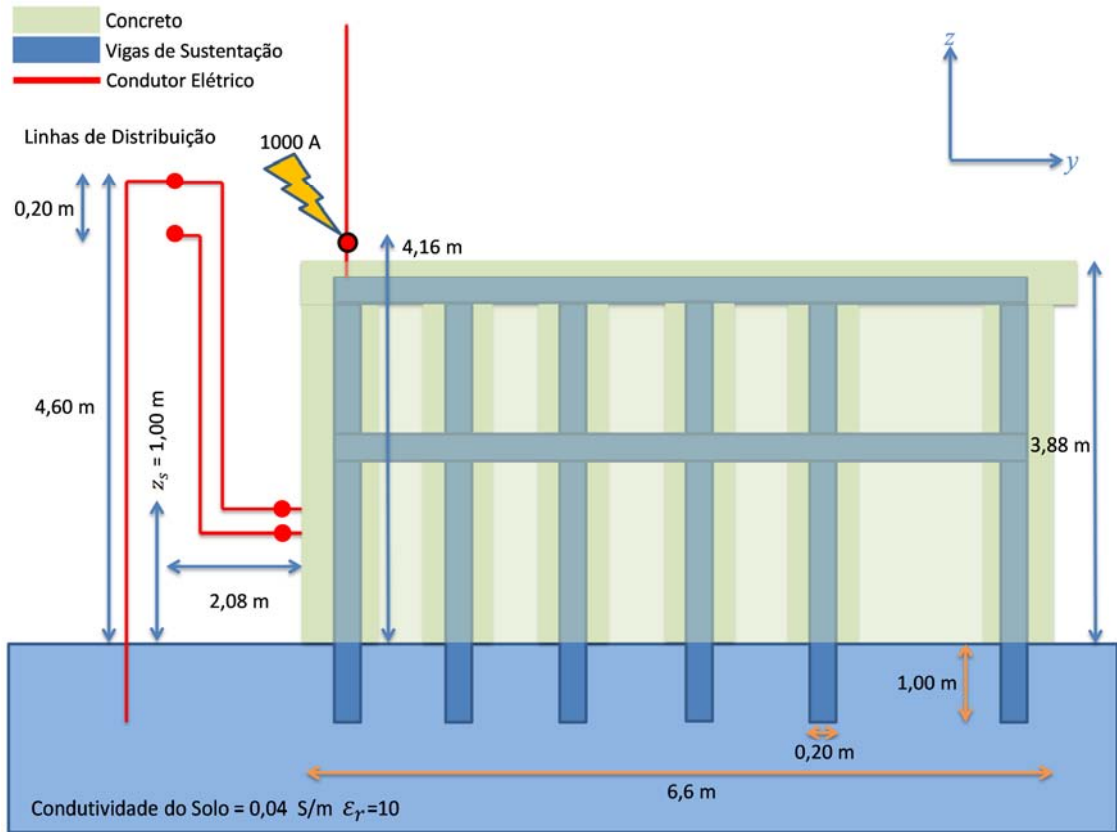


Figura 4.3: Visão lateral da residência para o Caso A.

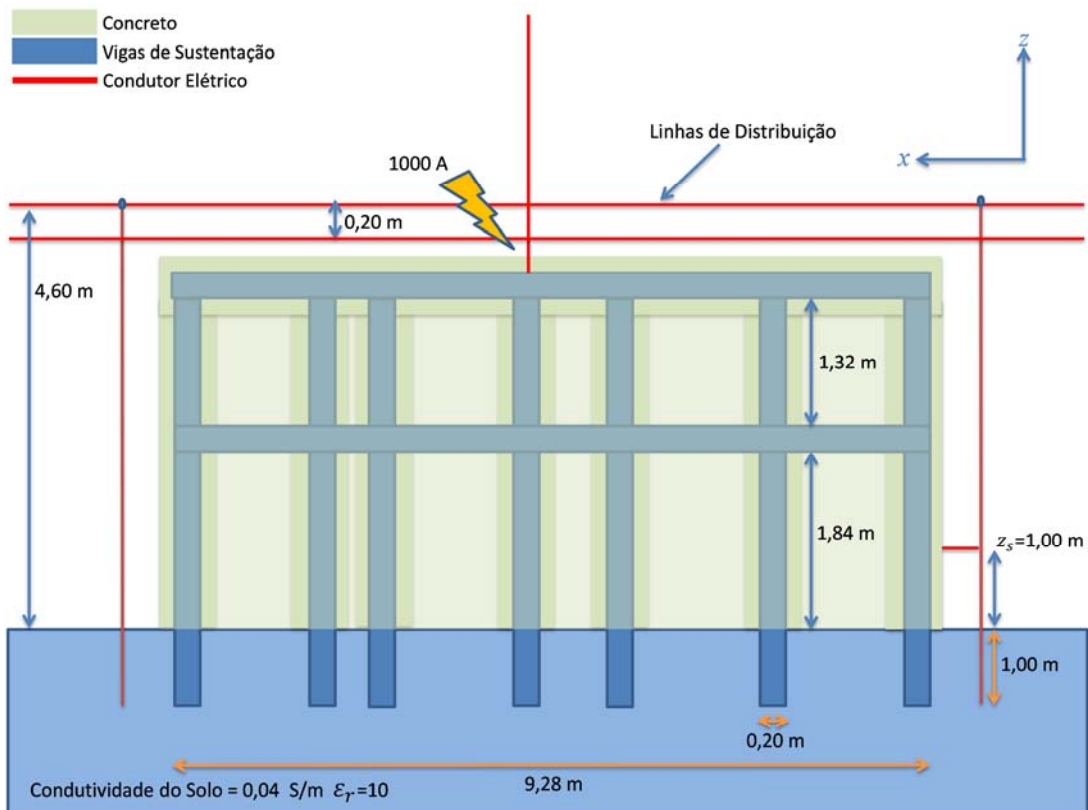


Figura 4.4: Visão frontal da residência para o Caso A.

A função matemática relativa à fonte usada na simulação para gerar a corrente de descarga atmosférica é idêntica à utilizada em [27, 32]. A função matemática da fonte pode ser descrita da seguinte maneira:

para  $t \leq T_f$ :

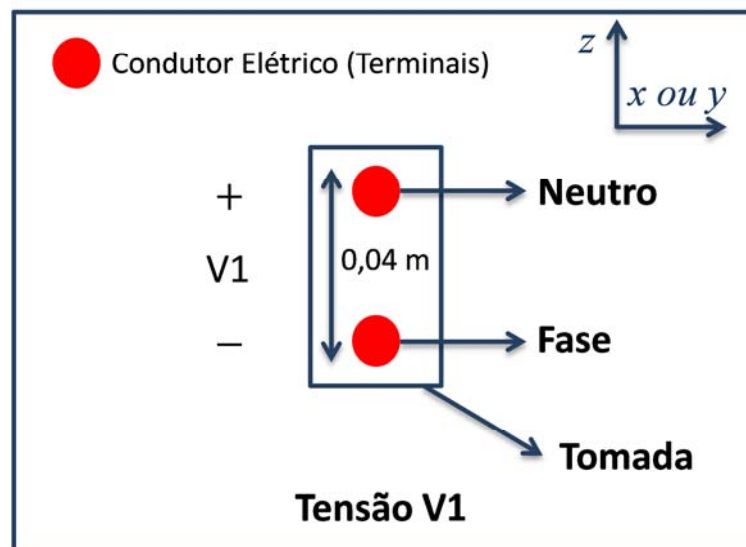
$$I_s(t) = \frac{I_{max}}{T_f} t \quad (4.1)$$

para  $t > T_f$ :

$$I_s(t) = -\frac{I_{max} \cdot t}{10^{-4}} + 1010 \quad (4.2)$$

em que  $I_s(t)$  é a função corrente,  $T_f = 10^{-6} s$ ,  $I_{max} = 1000$  A, gerando um pulso  $1/50 \mu s$  com pico de 1 kA. Internamente, esta função é implementada excitando-se as componentes covariantes do campo magnético [14], situadas no entorno do fio que representa o canal de descarga, na altura do ponto de *attachment* [32]. Esta função é utilizada também em todos os casos tratados a seguir.

### Tomada: Visão Frontal



(a)

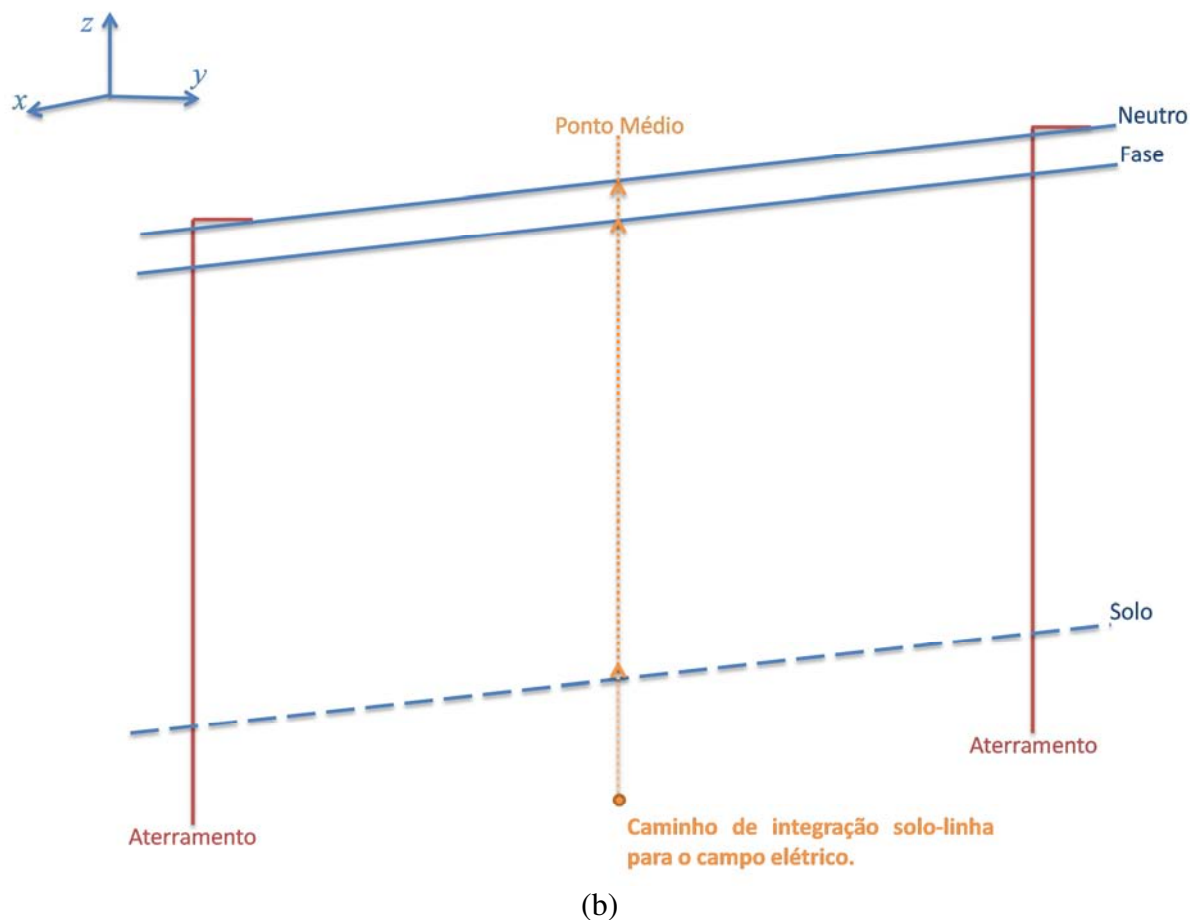


Figura 4.5: (a) Tensão  $V_1$  calculada na tomada; (b) Caminho de integração numérica utilizado no cálculo da tensão induzida na linha de distribuição (linha pontilhada).

A Figura 4.6a mostra a tensão induzida entre o solo e a fase (Figura 4.5b) para o problema apresentado pelas Figuras 4.1 – 4.5, utilizando-se (4.1) e (4.2) com 1000A de pico para a corrente relativa à descarga atmosférica. Observa-se uma voltagem máxima de aproximadamente 3540 V para  $t \approx 1 \mu\text{s}$ . As oscilações observadas para  $t > 0,1 \mu\text{s}$  são decorrentes das múltiplas reflexões de campo que ocorrem devido à edificação, ao solo e às próprias linhas de distribuição. A Figura 4.6b mostra a tensão induzida entre o solo e o condutor neutro (Figura 4.5b). Observa-se que até  $t \approx 0,1 \mu\text{s}$  o resultado obtido é similar ao da Figura 4.6a (para a fase), devido à incidência direta do campo e à pequena diferença de altura entre as linhas. Porém, para  $t > 0,1 \mu\text{s}$ , os resultados obtidos são diferentes, de forma que a tensão relativa ao neutro é consideravelmente menor do que a da tensão induzida na fase para todos os instantes de tempo considerados a partir de  $0,1 \mu\text{s}$ . O máximo valor absoluto de voltagem obtido para o neutro (1700 V) ocorre exatamente em  $t = 0,1 \mu\text{s}$ . Isto é atribuído ao

aterramento do condutor neutro (Figura 4.3b). Os resultados obtidos através do *software* LANE SAGS e do código aqui desenvolvido apresentam total concordância. Vale ressaltar que o aterramento do condutor neutro não é uma prática recomendável [31].

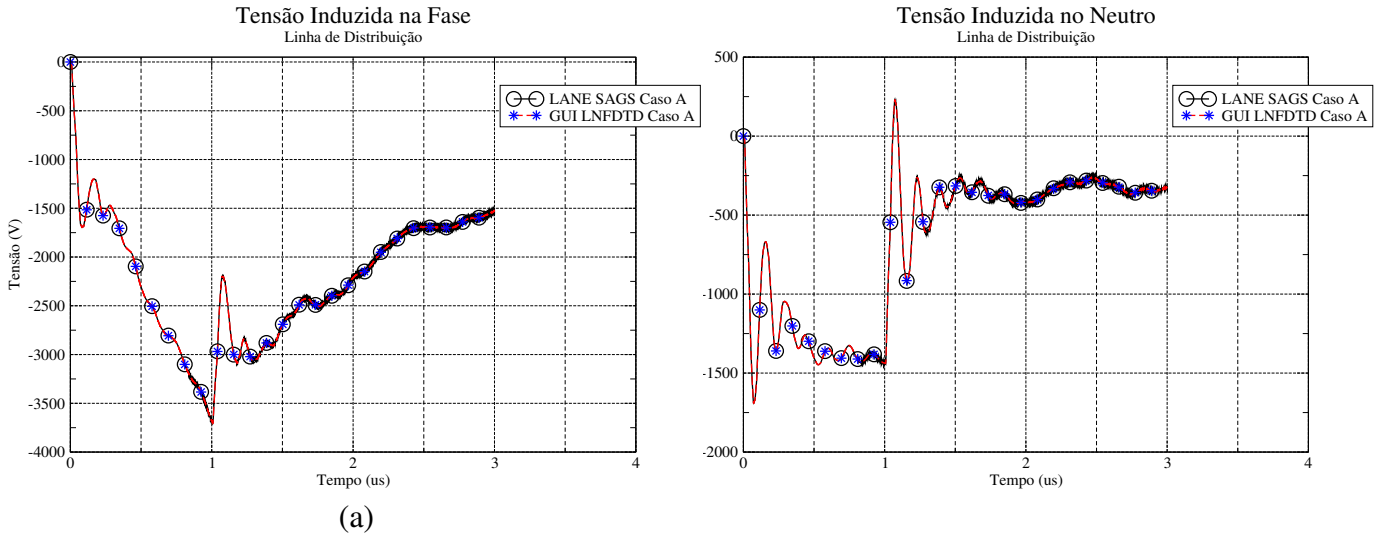


Figura 4.6: (a) Tensão induzida entre o solo e a fase; (b) Tensão induzida entre o solo e o neutro (Caso A).

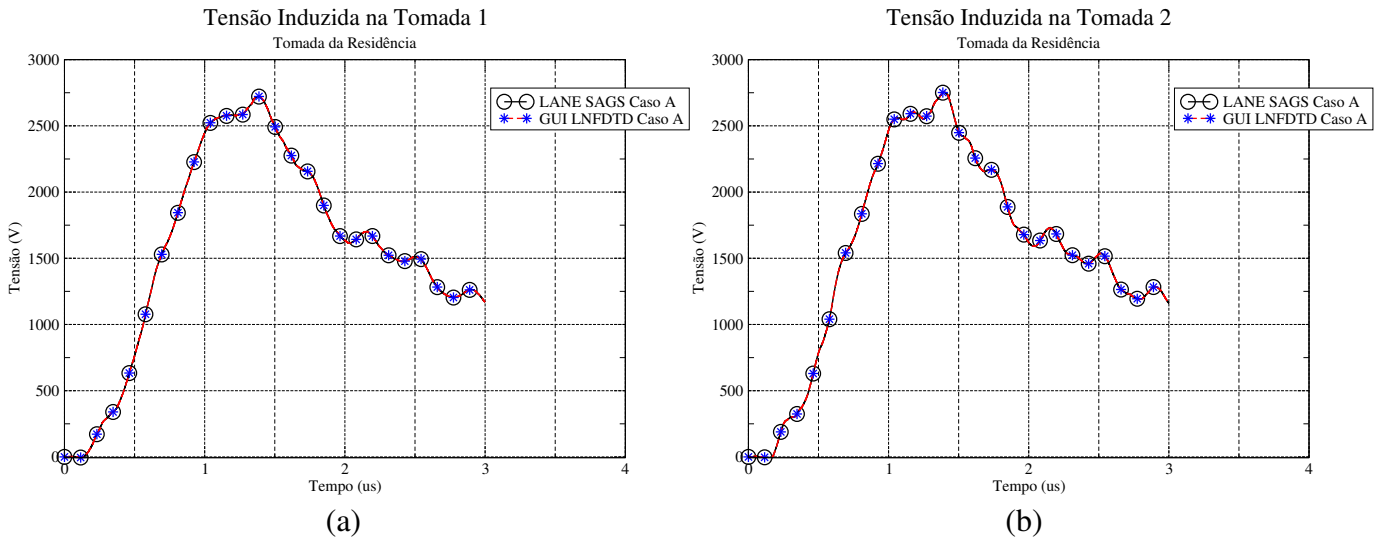


Figura 4.7: (a) Tensão induzida na tomada 1; (b) Tensão induzida na tomada 2 (Caso A).

As Figuras 4.7 - 4.10 mostram as tensões induzidas obtidas para as oito tomadas para o presente caso. Observa-se que a função tensão induzida para as tomadas é muito pouco afetada em relação aos cômodos do imóvel, de forma que todas elas apresentam seu valor máximo em torno de 2750 V para  $1,2 \mu s < t < 1,4 \mu s$ . Esta janela de tempo, na qual o máximo ocorre, está associada ao tempo de propagação do sinal para cada caso e às reflexões. Além

disso, os valores obtidos para as tensões induzidas nas tomadas são compatíveis com a diferença entre os sinais apresentados pelas Figuras 4.6.

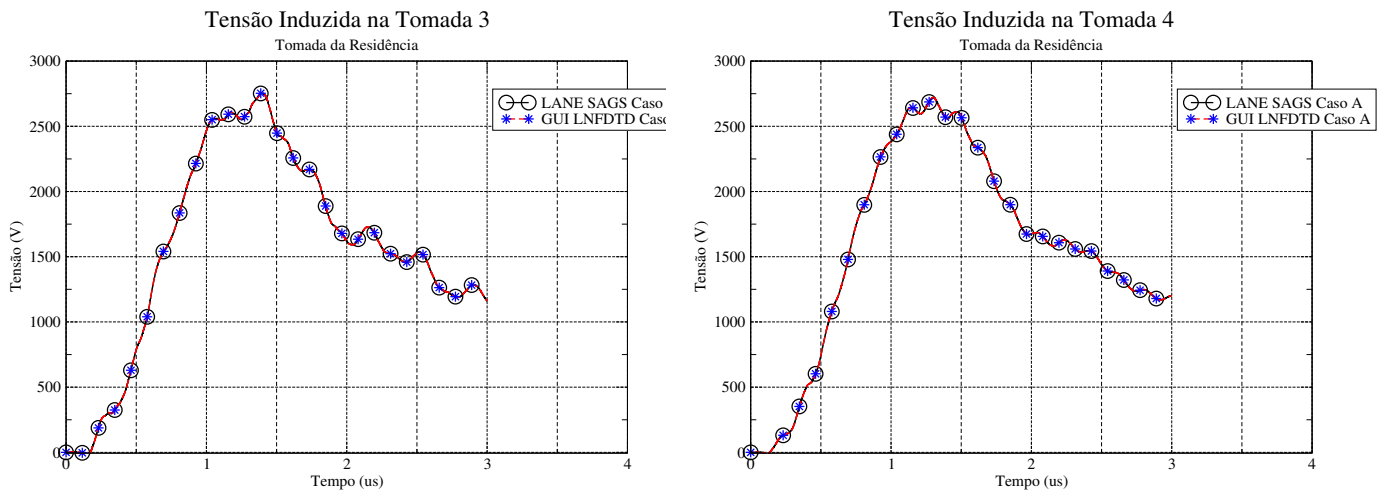


Figura 4.8: (a) Tensão induzida na tomada 3; (b) Tensão induzida na tomada 4 (Caso A).

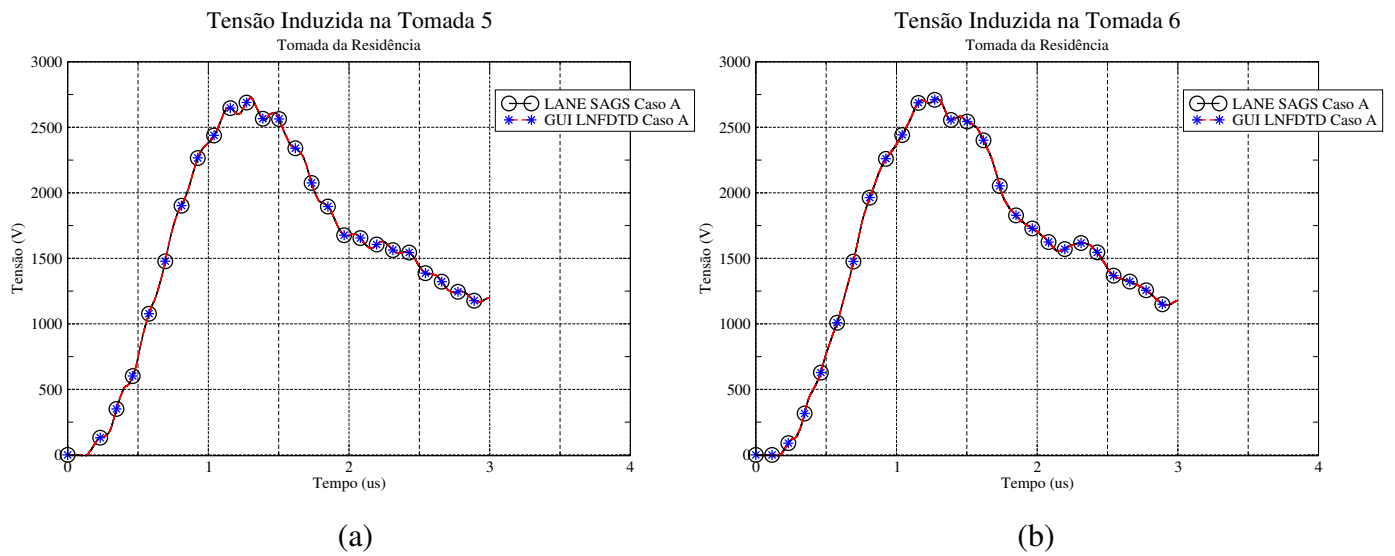


Figura 4.9: (a) Tensão induzida na tomada 5; (b) Tensão induzida na tomada 6 (Caso A).

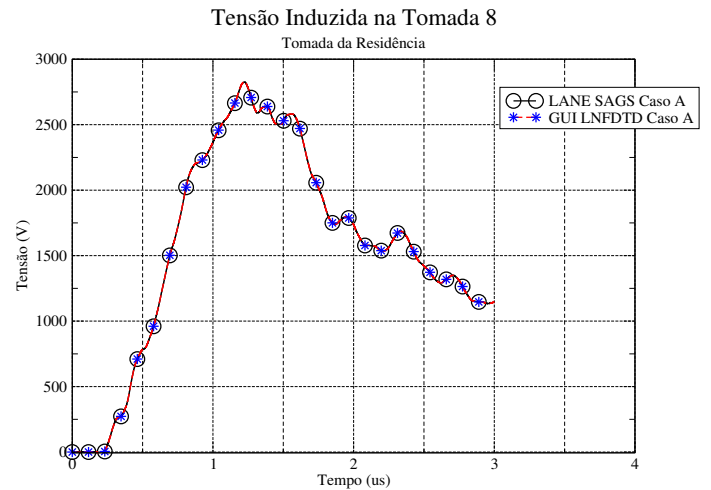
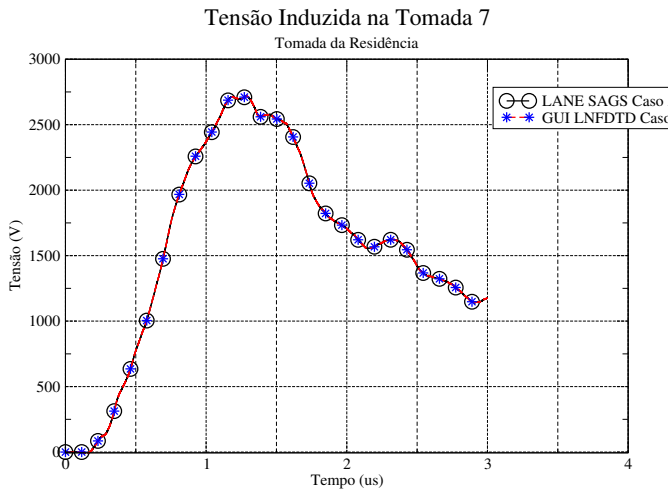


Figura 4.10: (a) Tensão induzida na tomada 7; (b) Tensão induzida na tomada 8 (Caso A).

Dada a grande similaridade entre as tensões obtidas para as tomadas, apenas a Tomada 1 será considerada nos demais casos.

Os gráficos da distribuição de  $|\vec{E}|$  para  $z = 2,88 \text{ m}$  ( $k=72$ ), que intercepta o neutro das tomadas, é mostrado pela Figura 4.11 para  $t = 1 \mu\text{s}$ ,  $t = 2 \mu\text{s}$  e  $t = 3 \mu\text{s}$ .

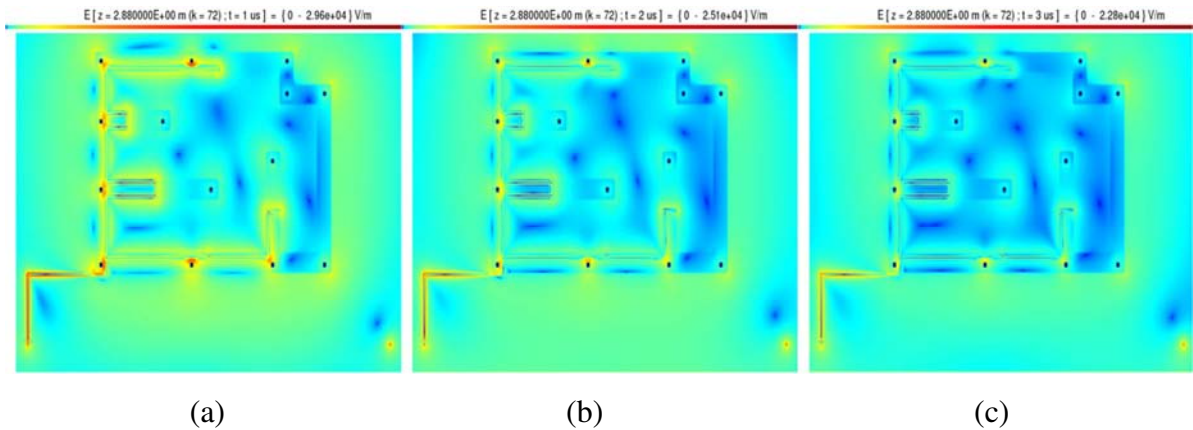


Figura 4.11: Distribuição espacial de  $|\vec{E}|$  para  $z = 2,88\text{m}$  ( $k=72$ ) e (a)  $t = 1\mu\text{s}$ ; (b)  $t = 2\mu\text{s}$ ; (c)  $t = 3\mu\text{s}$ .

Os gráficos da distribuição espacial de  $|\vec{E}|$  para  $y = 3,60\text{m}$  ( $j=90$ ), plano que intercepta a fonte, são mostrados pela Figura 4.12 para  $t = 1 \mu\text{s}$ ,  $t = 2 \mu\text{s}$  e  $t = 3 \mu\text{s}$ .



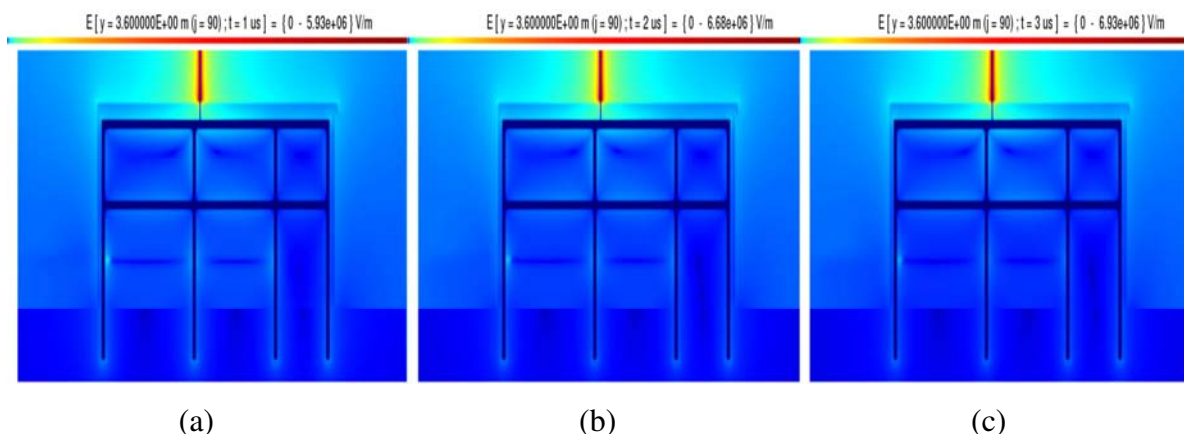


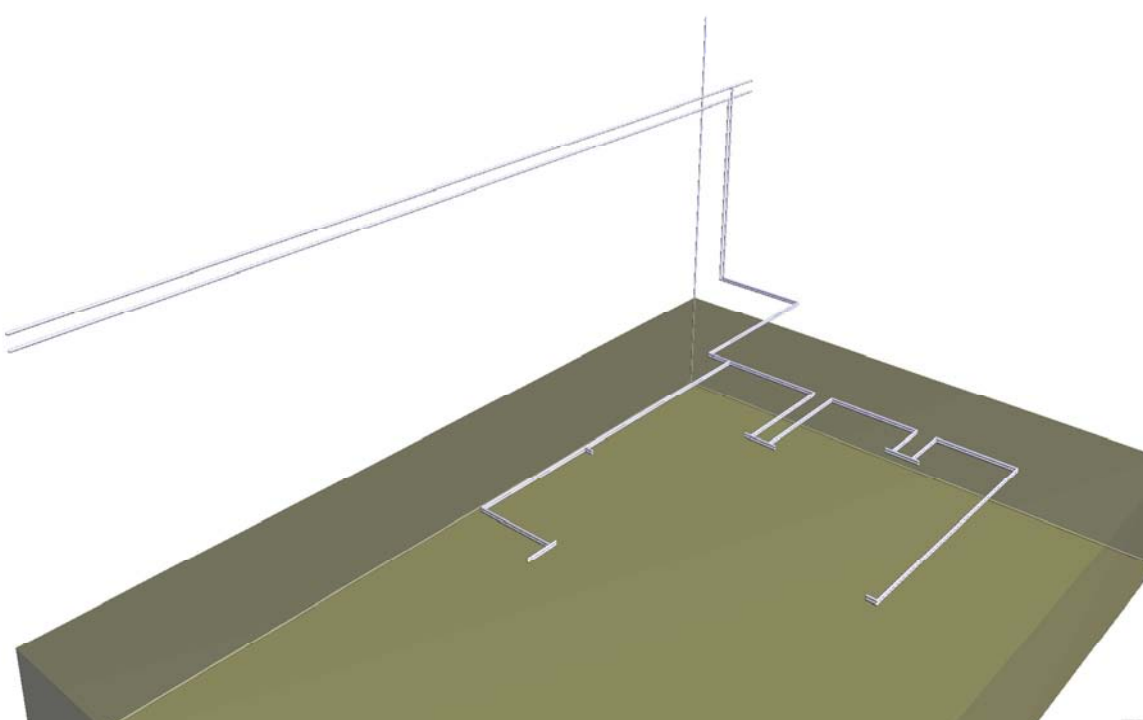
Figura 4.12: Distribuição espacial de  $|\vec{E}|$  para  $y = 3,60\text{m}$  ( $j=90$ ) e (a)  $t = 1\mu\text{s}$ ; (b)  $t = 2\mu\text{s}$ ; (c)  $t = 3\mu\text{s}$ .

Os dados apresentados graficamente pelas Figuras 4.11 e 4.12 mostram que a própria estrutura da residência blindada sua região interna, de forma que as maiores intensidades de campo são observadas no exterior da construção. No caso específico da Figura 4.11, é possível perceber que o campo induzido nas linhas de distribuição é levado para a parte interior da residência através principalmente do sistema elétrico da construção, que está conectado eletricamente às linhas de distribuição.

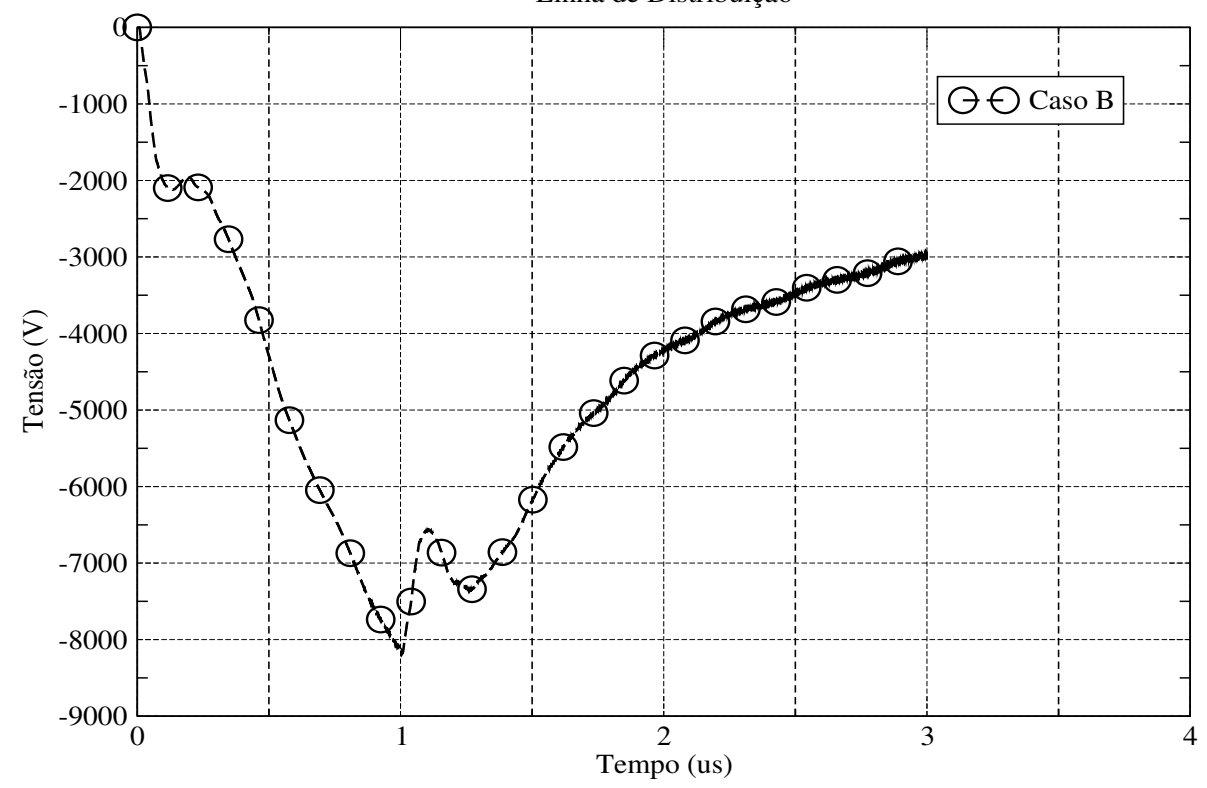
Os resultados gerados por ambos os *software* são exatamente iguais para a simulação do Caso A. Com isso, os resultados gerados pela GUI LN-FDTD para a simulação utilizando uma malha retangular são validados pela ferramenta LANE-SAGS.

#### 4.1.2 - Caso B

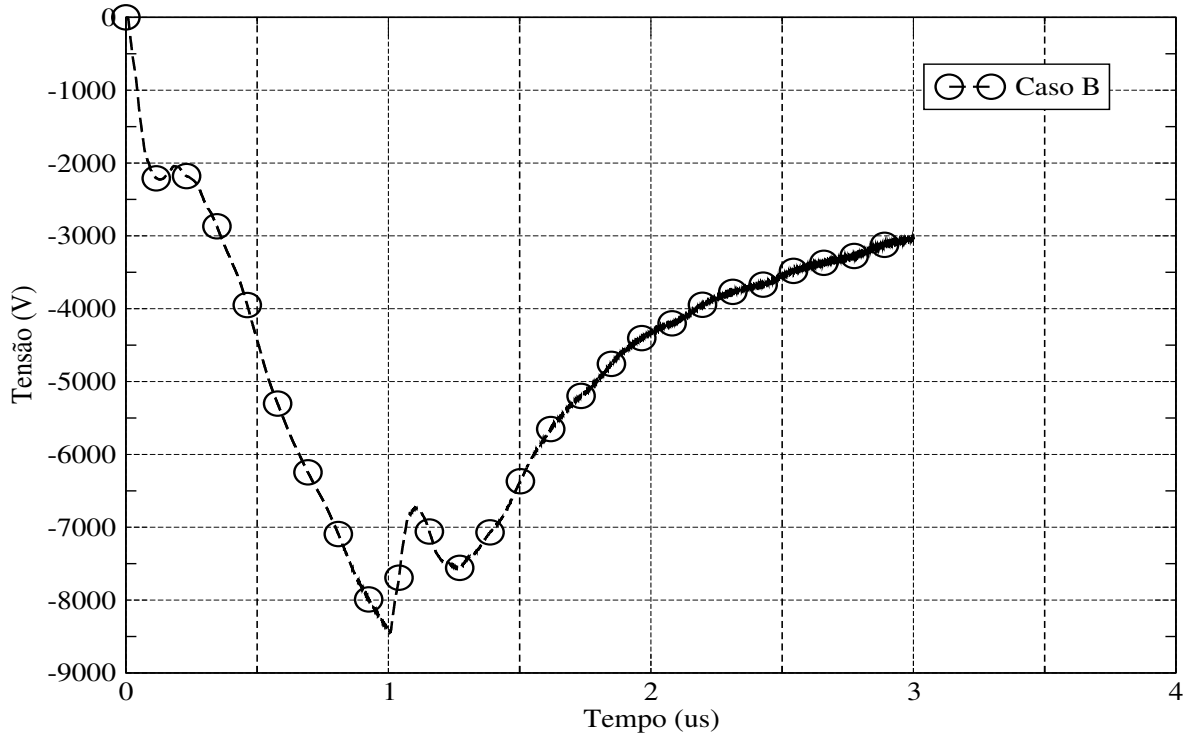
Este caso tem por objetivo avaliar a influência do aterramento do condutor neutro simulado no caso anterior. Para tanto, o aterramento deste condutor foi desfeito e todos os demais parâmetros da simulação anterior foram conservados. O problema é ilustrado pela Figura 4.13. Os resultados obtidos estão representados graficamente pelas Figuras 4.14, 4.15 e 4.16.



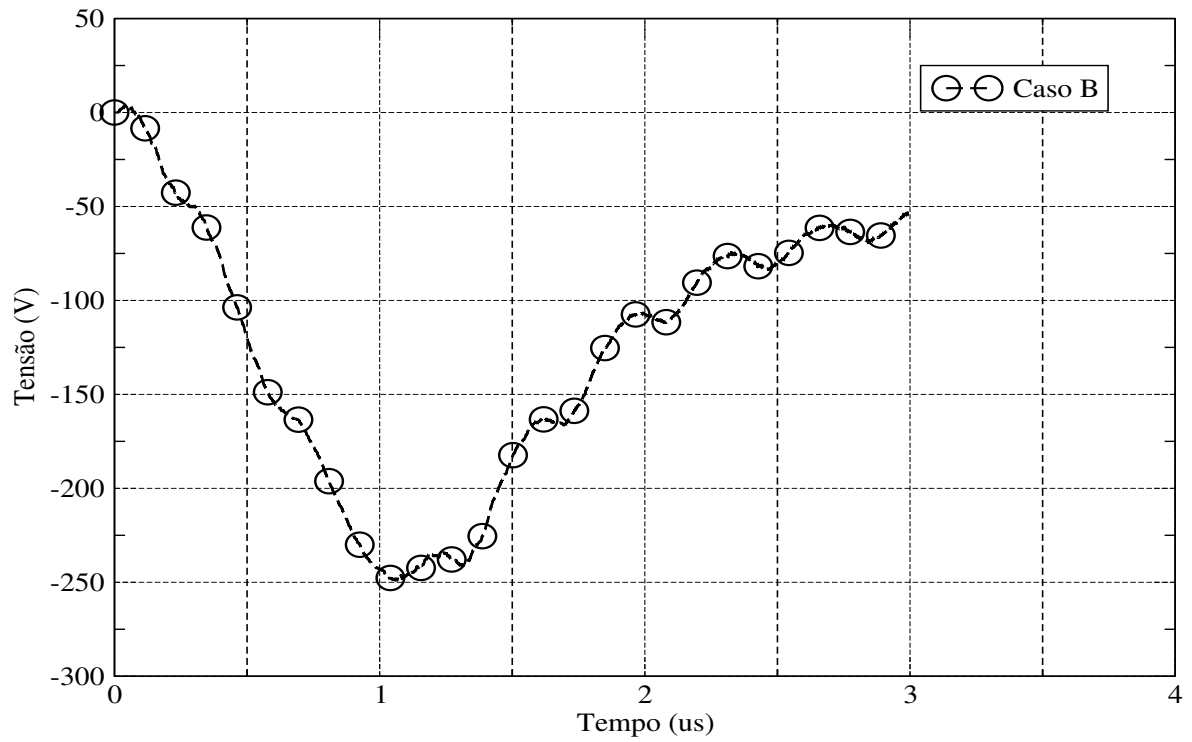
Tensão Induzida na Fase  
Linha de Distribuição



## Tensão Induzida no Neutro Linha de Distribuição



## Tensão Induzida na Tomada 1 Tomada da Residência



Ao se inspecionar as Figuras 4.14 e 4.15, é observado que as tensões obtidas para neutro e fase são levemente diferentes, confirmando a observação feita no caso anterior de que o aterramento do neutro é responsável pela diminuição da tensão induzida relativa a este condutor. Este efeito é refletido também pela tensão induzida na tomada 1 (Figura 4.16), onde observa-se um valor máximo de 250 V para  $t \approx 1,1 \mu\text{s}$ , para cada kA de corrente pico da descarga atmosférica. Comparando a Figura 4.9(a) (pico de  $\sim 2750$  V) e a Figura 4.16, observa-se que a eliminação do aterramento do condutor neutro promoveu uma redução de aproximadamente 90% para o módulo do pico da tensão induzida na tomada 1, o que confirma a recomendação de não aterrar o neutro. Reduções de ordem similar foram observadas para as demais tomadas (Apêndice E).

### **4.1.3 – Caso C**

A proposta deste caso é verificar a influência da geometria do sistema de proteção (pára-raios) nas tensões induzidas nas tomadas e entre o solo e as linhas de distribuição. A Figura 4.17 ilustra a geometria aqui analisada, na qual o sistema de captação da descarga não é conectado à estrutura metálica da edificação, mas é ligado eletricamente a uma haste de aterramento através de um cabo de descida, o qual é isolado eletricamente da residência através de afastadores dielétricos, separados entre si por 0,52 m. Vale ressaltar, que a fase neutro da linha de distribuição está aterrado.

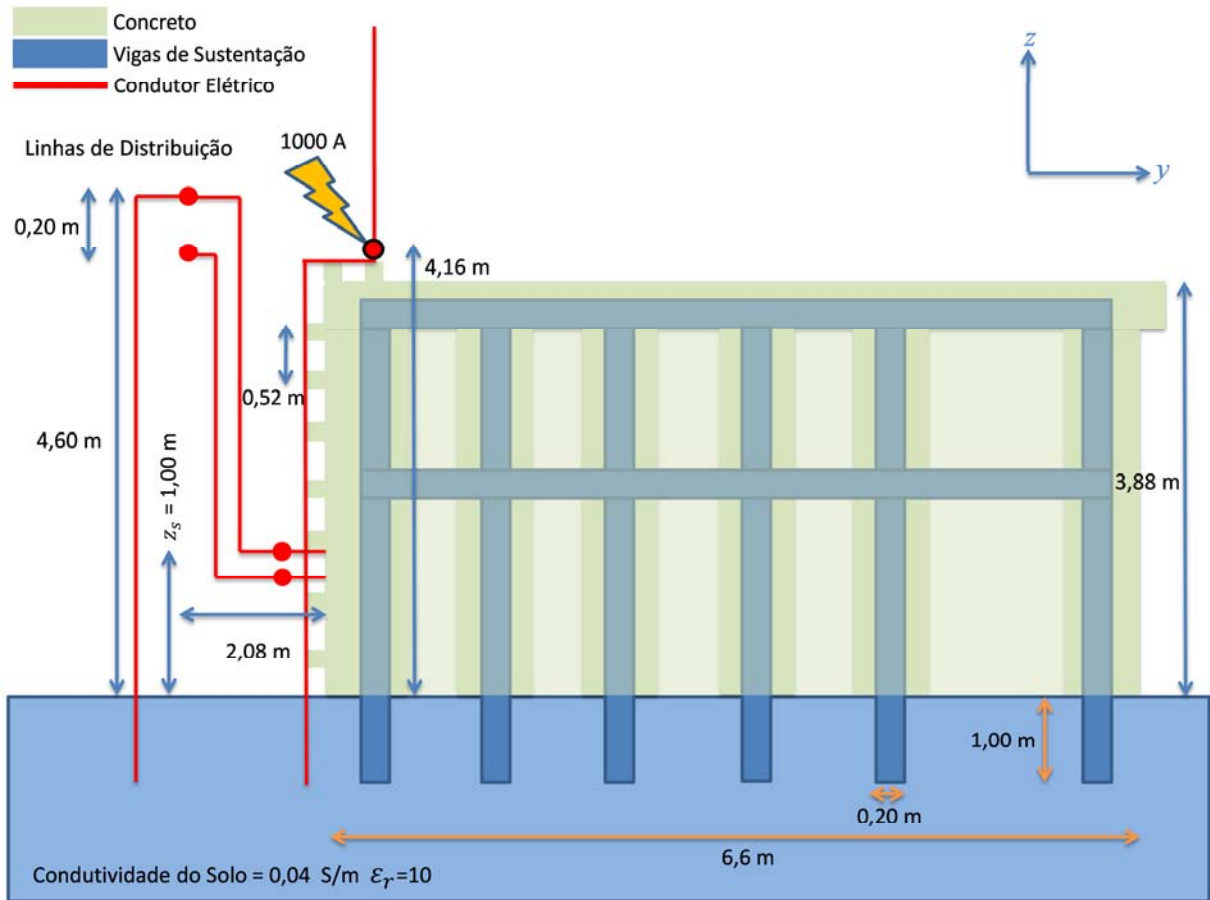


Figura 4.17: Nova estrutura de pára-raios para o Caso C.

Na Figura 4.18, o gráfico da distribuição de  $|\vec{E}|$  para  $x = 7,96$  m ( $i=199$ ) (plano da fonte), para  $t = 0.333\mu\text{s}$ ,  $t = 0.667\mu\text{s}$  e  $t = 1\mu\text{s}$ , demonstra que, novamente as maiores intensidades de campo são observadas no exterior da residência.

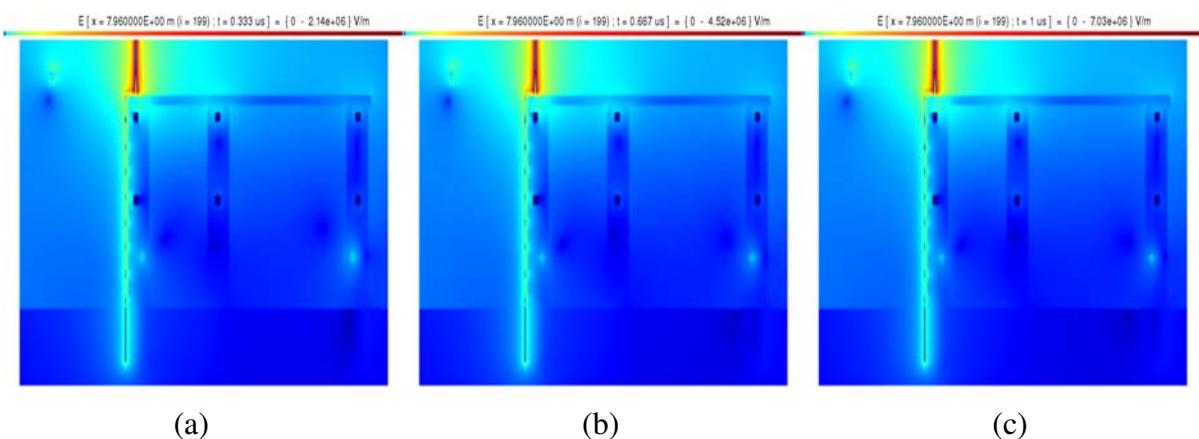
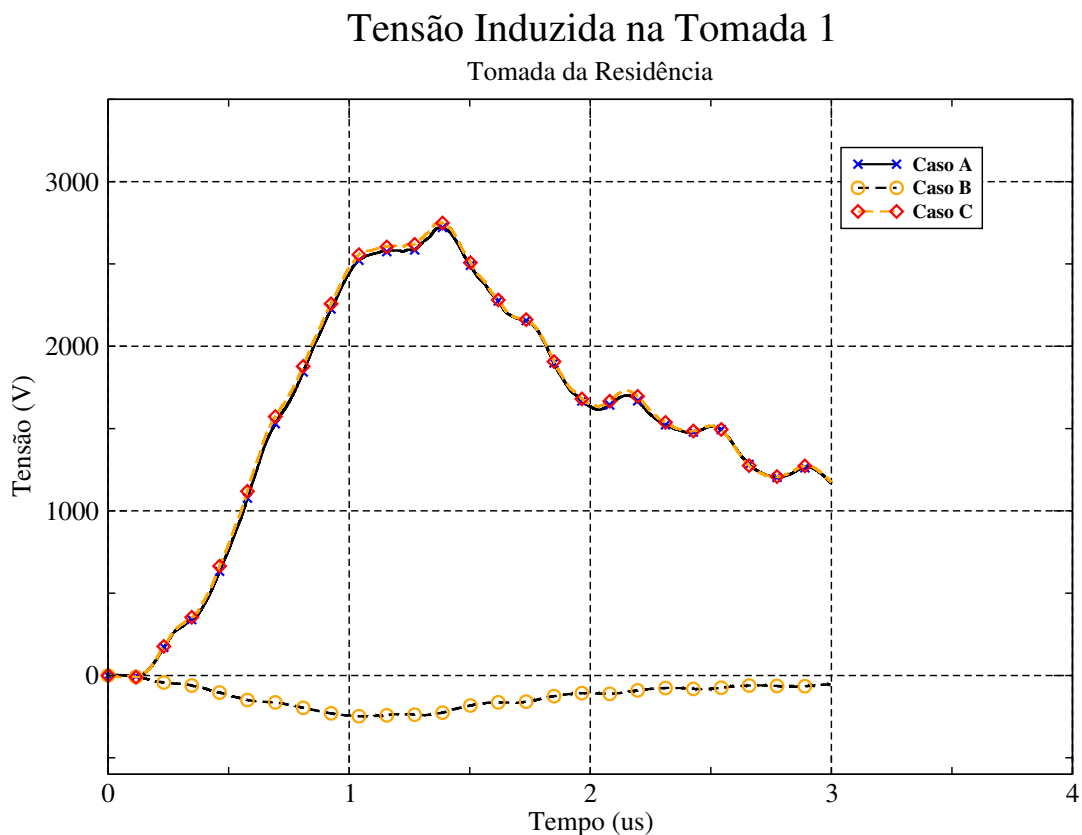
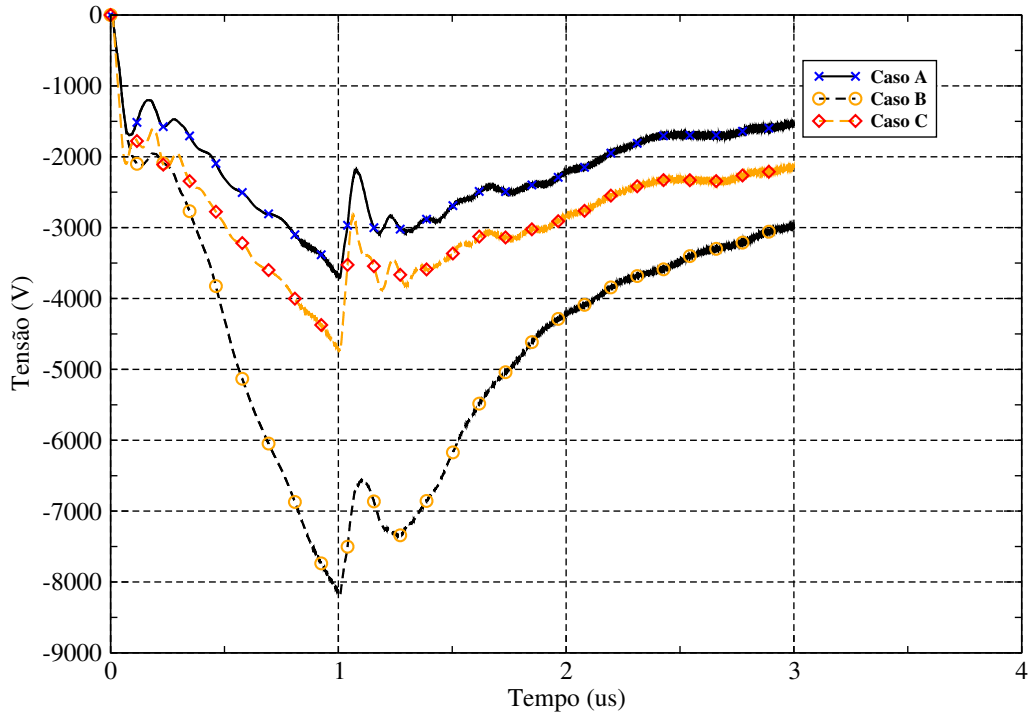


Figura 4.18: Distribuição espacial de  $|\vec{E}|$  para  $x = 7,96$  m ( $i=199$ ) após (a)  $0.333\mu\text{s}$ ; (b)  $0.667\mu\text{s}$ ; (c)  $1\mu\text{s}$  para o Caso C.

Na Figura 4.19 são apresentados os gráficos relativos às tensões induzidas na tomada 1, entre solo e fase e entre solo e neutro (Figura 4.5 (b)), obtidas para este caso e os casos anteriores. A Figura 4.19(a) revela que, para as tomadas, a tensão transitória induzida praticamente independe de forma como a corrente proveniente da descarga é conduzida ao solo (por um condutor externo ou através da estrutura metálica da residência). Porém, as tensões induzidas nas linhas de distribuição são afetadas, de forma que seus módulos são incrementados similarmente em relação ao que ocorre ao caso anterior (praticamente os mesmos valores são adicionados a cada instante para cada linha), conforme mostram as Figuras 4.19(b) e 4.19(c). Isto ocorre nas linhas de distribuição porque no Caso A, a corrente se divide pela estrutura metálica da edificação, diferentemente do que ocorre no presente caso, em que a corrente que flui pelo condutor que liga o pára-raios à haste de aterramento, aumentando a intensidade de campo na região próxima às linhas de distribuição. Isto é visível quando a Figura 4.12 é comparada à Figura 4.18. O comportamento observado para as tensões induzidas nas linhas de distribuição em função da forma como a corrente é conduzida ao solo é compatível com o que é mostrado em [33]. Porém, em [33], não são analisadas as tensões in

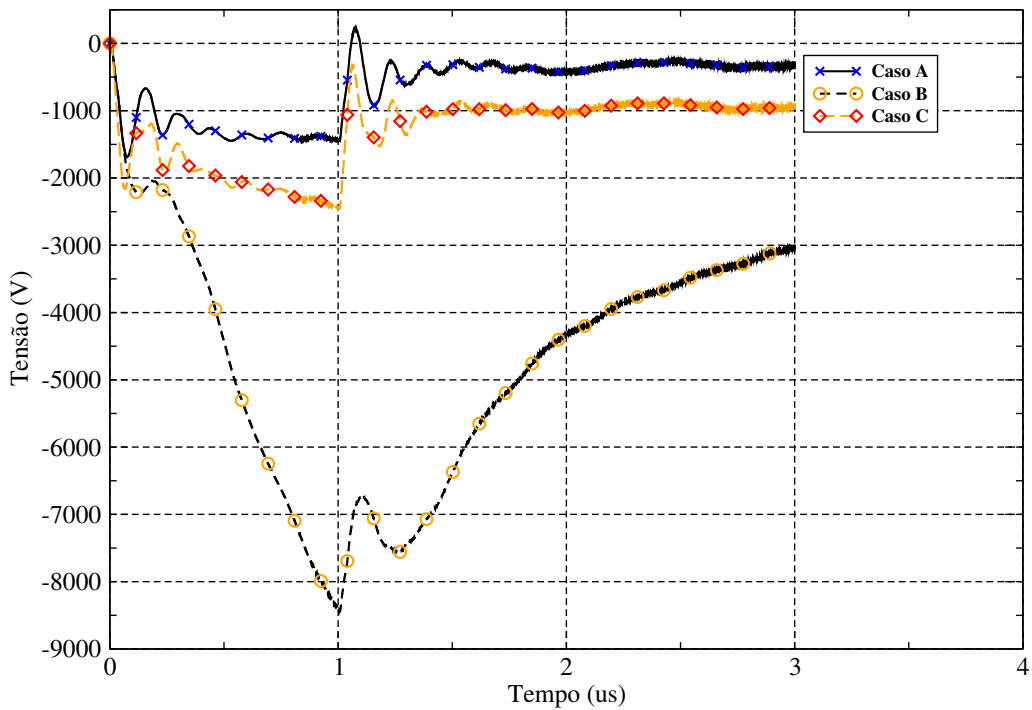


### Tensão Induzida na Fase Linha de Distribuição



(b)

### Tensão Induzida no Neutro Linha de Distribuição



(c)

Figura 4.19: Tensões induzidas para os Casos A, B e C para (a) Tomada 1; (b) Fase; (c) Neutro.

#### 4.1.4 – Casos D e E

As simulações até o momento foram realizadas utilizando uma malha computacional retangular. Para demonstrar que o *software* está preparado para realizar simulações envolvendo geometrias não compatíveis com o sistema Cartesiano, é modelada a catenária na linha de distribuição, relativa à ação da gravidade no condutor. O cenário é representado pela Figura 4.20. Para isto, desenvolveu-se uma malha computacional específica (não retangular) para representar a função que descreve tal tipo de curvatura na região onde as linhas são modeladas. Isto torna a simulação mais próxima do que ocorre na realidade. Um corte transversal da malha, relativo ao plano que contém as linhas de distribuição, é apresentado pela Figura 4.21. A função que descreve matematicamente a curvatura das linhas, causada pela ação da gravidade, é a função cosseno hiperbólico, tal como descrito em [34] e em [35]. Uma representação tridimensional da estrutura, gerada pelo *software* desenvolvido neste trabalho, é apresentada pela Figura 4.22. Aqui, o Caso D é relativo à conexão do pára-raios com a estrutura metálica da edificação (Figura 4.3). Já no Caso E, o pára-raios é conectado a uma haste de aterramento através de um condutor de descida, tal como ilustra a Figura 4.17. Foram modeladas duas situações diferentes para cada caso. Inicialmente modelou-se a linha com flecha  $\Delta f = 0,20$  m (Figura 4.20) e, posteriormente, a malha foi refeita de tal forma que  $\Delta f = 0,30$  m. Em ambos os casos, o afastamento entre as linhas foi mantido em 0,20 m.



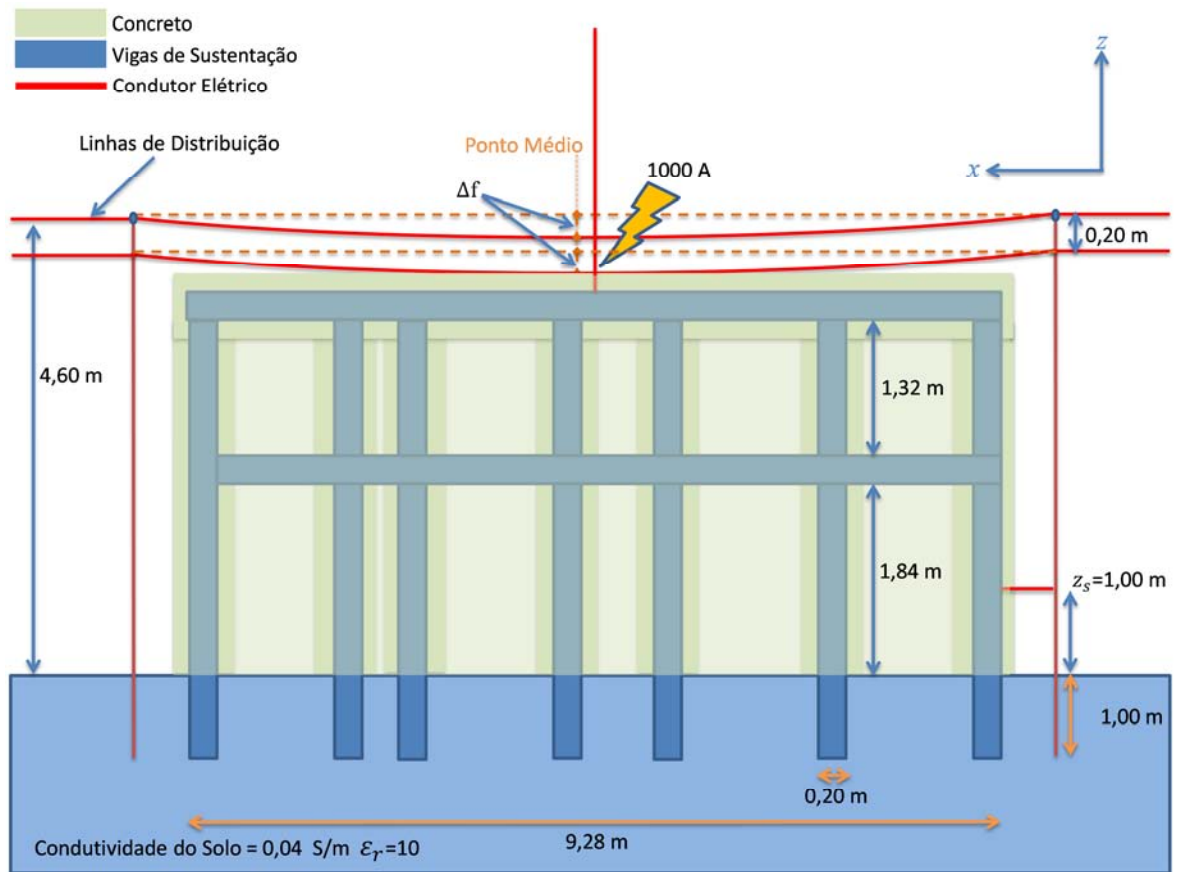


Figura 4.20: Visão frontal com a representação da catenária na linha de distribuição.

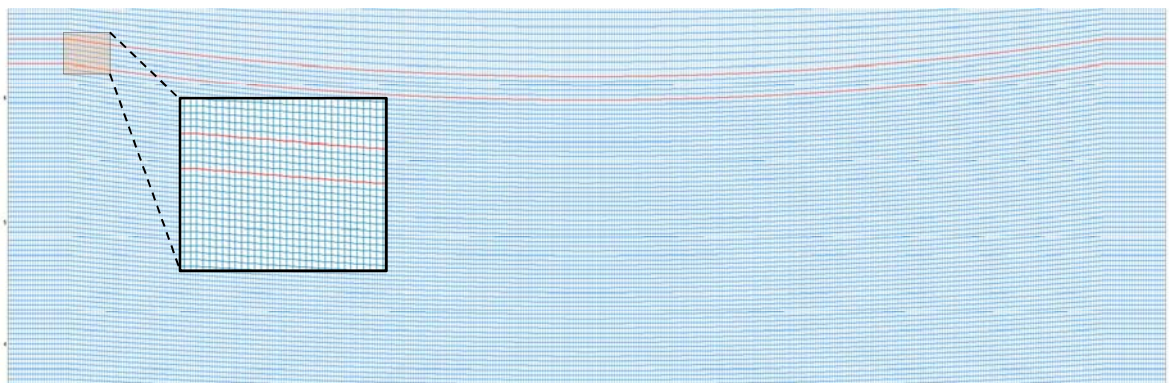


Figura 4.21: Corte transversal da malha desenvolvida para a simulação. Destacam-se as linhas curvadas.

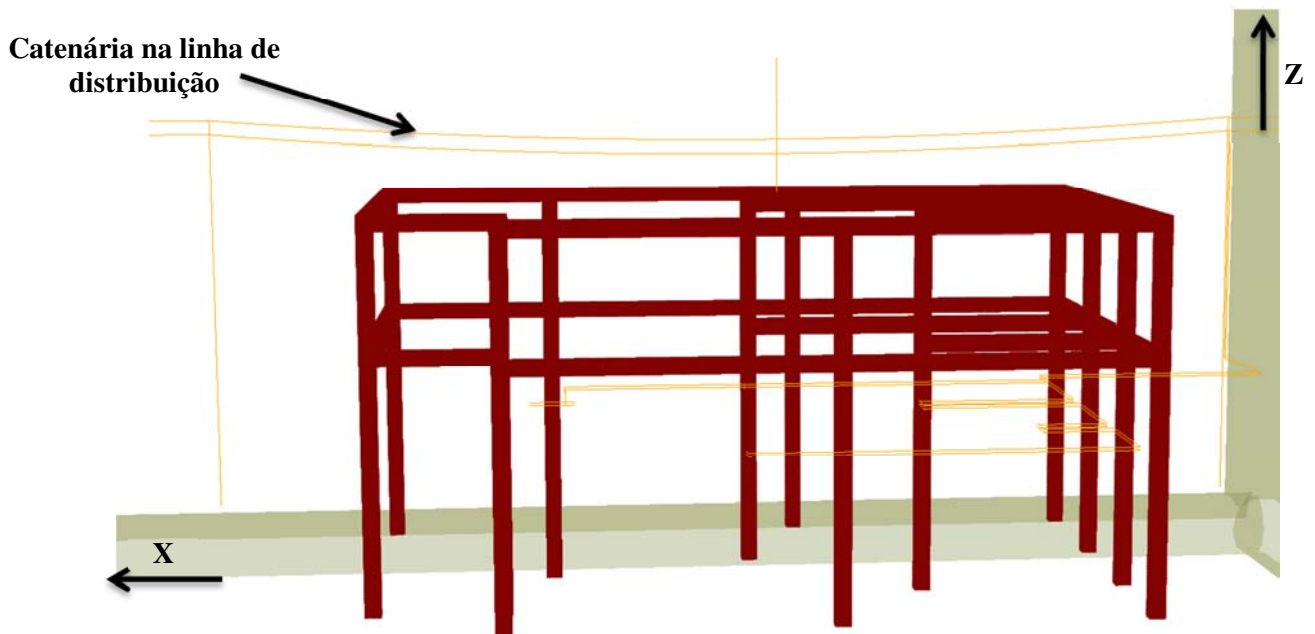


Figura 4.22: Estrutura metálica da residência e catenárias relativas às linhas de distribuição: visão em 3D sem os blocos dielétricos.

Na Figura 4.23, é apresentada a distribuição espacial da componente  $e_2$  do campo elétrico, em escala logarítmica, para  $t \approx 0.0247 \mu s$ . O campo “ilumina” as linhas que representam a fase e o neutro, de forma que podem ser observadas claramente as catenárias modeladas para os  $\Delta f$  distintos. O plano mostrado ( $j=33$ ) contém as linhas de distribuição.

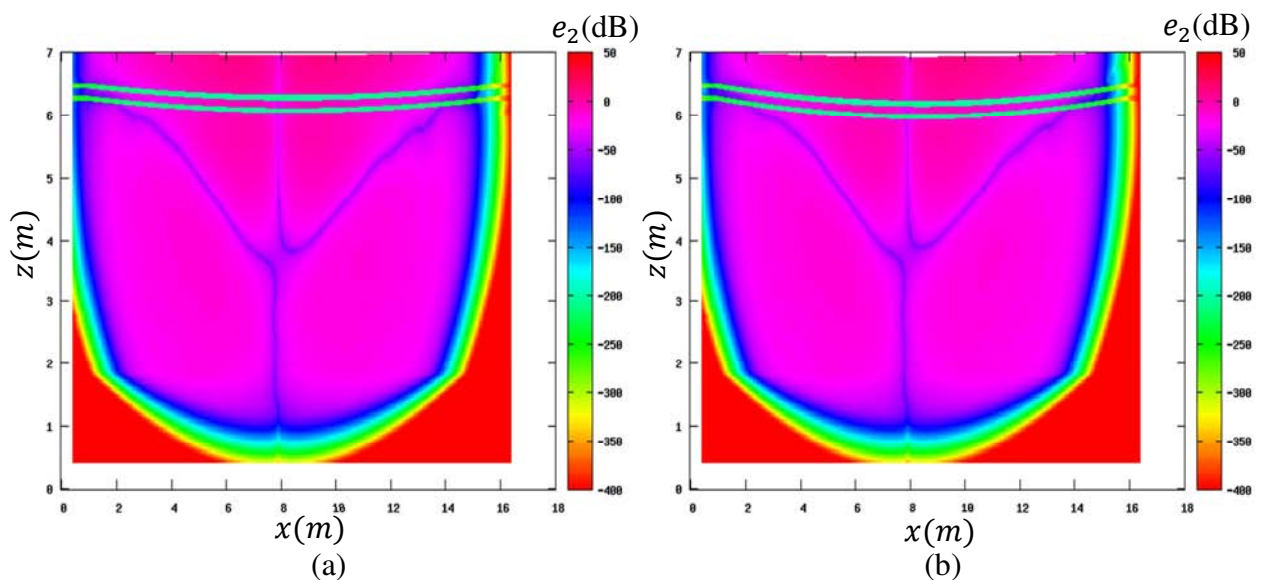
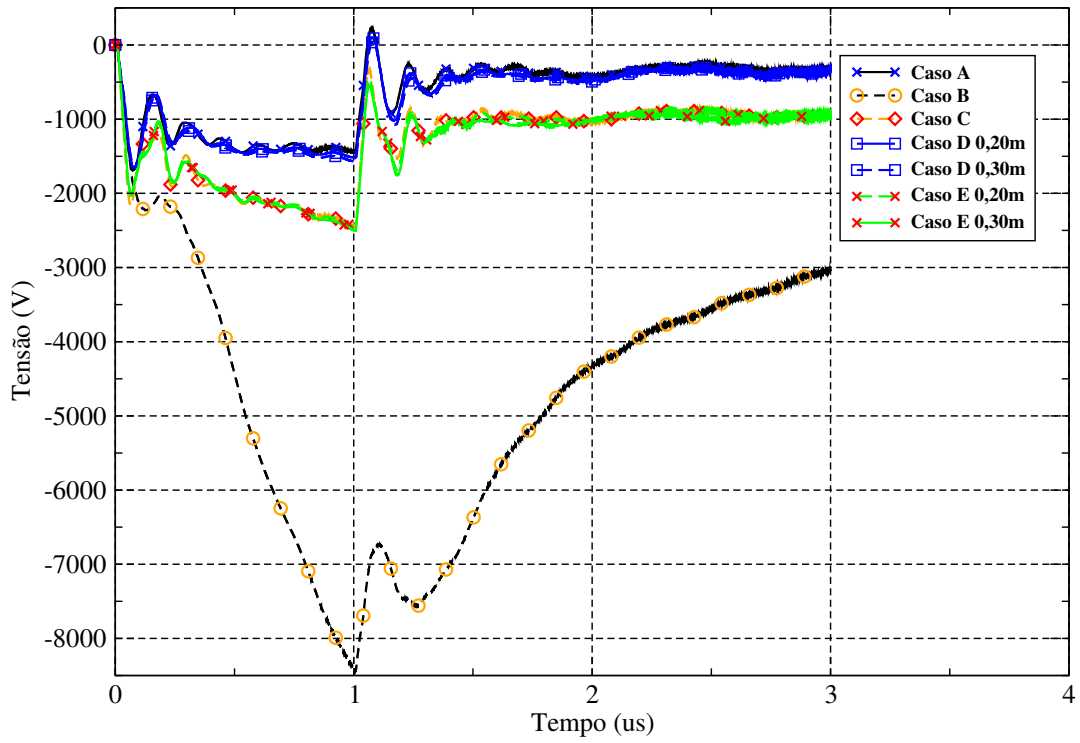


Figura 4.23: Distribuição da componente  $e_2$  do campo elétrico para  $y=1,32$  m,  $t = 0,02474526\mu s$  para (a)  $\Delta f = 0,20m$  e (b)  $\Delta f = 0,30m$ .

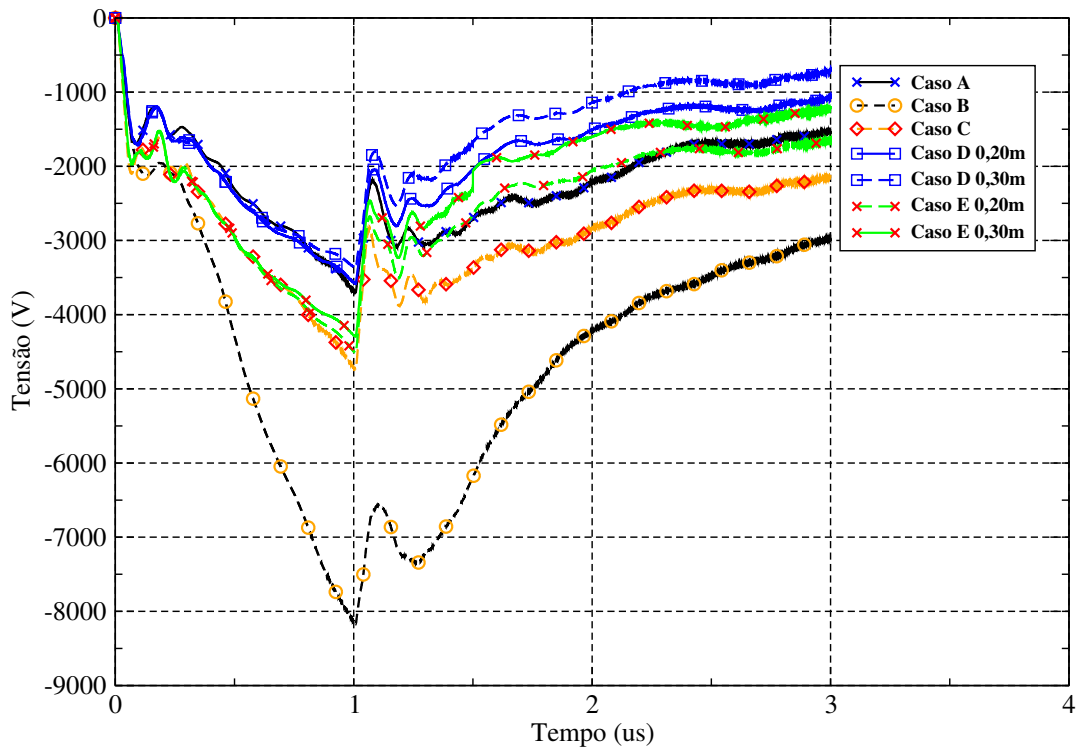
Na Figura 4.24, são apresentados os gráficos das tensões induzidas obtidas entre solo e neutro (Figura 4.24(a)), entre solo e fase (Figura 4.24(b)) e entre os terminais da tomada 1 (Figura 4.24(c)), para os Casos D e E ( $\Delta f = 0,20\text{m}$  e  $\Delta f = 0,30\text{m}$ ). Estes resultados são comparados com os resultados obtidos nas simulações anteriores.

Ao observar a Figura 4.24(a), nota-se que as tensões induzidas no neutro não são consideravelmente alteradas pela curvatura da linha, em todos os casos considerados. Isto ocorre devido ao aterramento deste condutor, cuja influência é dominante no processo eletrodinâmico em relação à catenária, conforme esperado. Porém, quando se analisa a Figura 4.24(b), é evidente a influência da curvatura do condutor fase na tensão induzida entre o solo e este condutor. Como estas tensões são avaliadas na posição de mínima altura da linha, observa-se que, de uma forma geral, ao longo do tempo, na medida em que  $\Delta f$  é incrementado, ou seja, a linha é rebaixada no ponto de avaliação da tensão induzida, há considerável redução do módulo desta grandeza elétrica. Isto é consistente com os problemas analisados em [32], trabalho no qual a altura das linhas é estudada, mas as catenárias não são modeladas. Tomando como referência os casos em que  $\Delta f = 0$ , observa-se uma diferença máxima de aproximadamente 600 V em relação aos casos em que  $\Delta f = 0,20\text{m}$  e da ordem de 1 kV em relação aos casos em que  $\Delta f = 0,30\text{m}$ . Por sua vez, este efeito se repercute na tomada 1, cujas tensões induzidas obtidas são apresentadas na Figura 4.24(c). A inspeção desta figura mostra que as tensões induzidas nas tomadas tendem a ser reduzidas. Isto está fisicamente consistente, pois a tensão no neutro não foi substancialmente alterada pela catenária e a tensão induzida entre solo e fase se aproximou dos valores obtidos para o condutor neutro. Dessa forma, ao se utilizar os casos em que  $\Delta f = 0$  como referência, a tensão induzida nos terminais da tomada 1 apresentou redução máxima de 700 V em relação aos casos em que  $\Delta f = 0,20\text{m}$  e de aproximadamente 1200 V em relação aos casos em que  $\Delta f = 0,30\text{m}$ . Por fim, vale ressaltar que a forma como a corrente chega ao solo (através da estrutura metálica ou através do condutor externo à residência) não afeta de forma significativa as tensões induzidas entre os terminais das tomadas (Figura 4.24(c)), o que reforça a ideia anteriormente discutida (as tensões induzidas nas tomadas dependem da posição relativa entre o canal da descarga e das linhas de distribuição).

## Tensão Induzida no Neutro Linha de Distribuição



## Tensão Induzida na Fase Linha de Distribuição



# Tensão Induzida na Tomada 1

Tomada da Residência

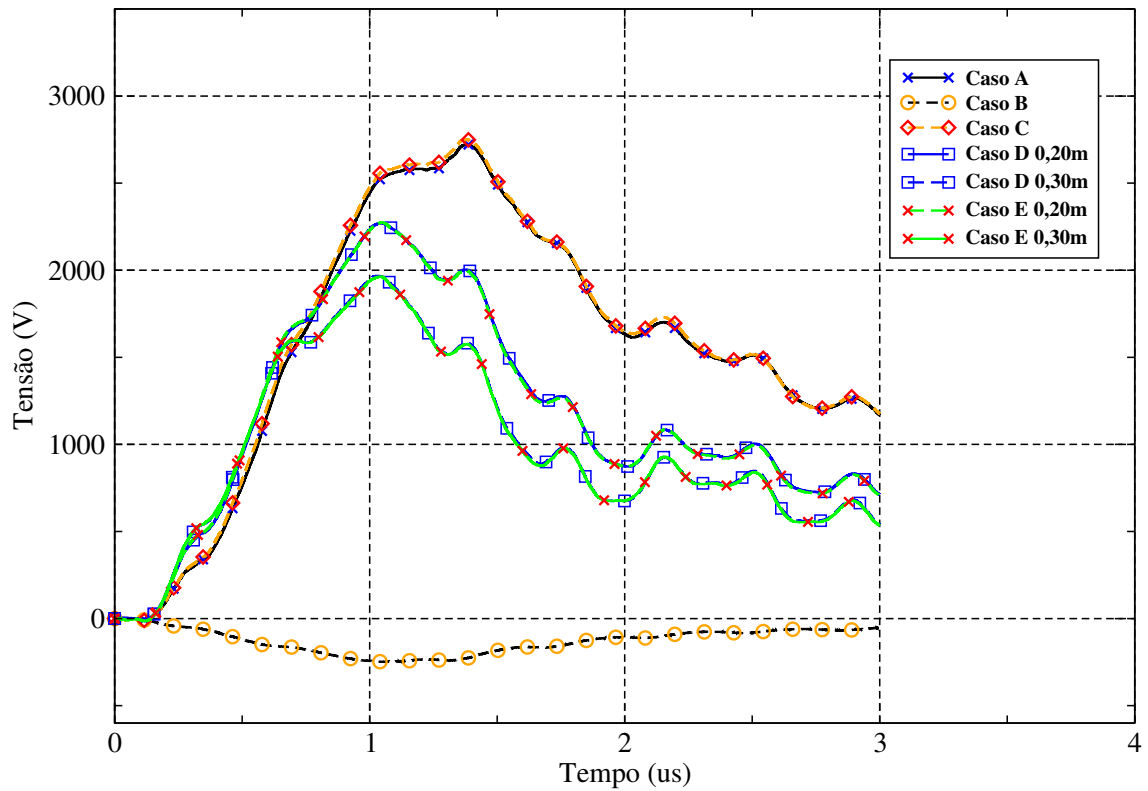


Figura 4.24: tensões induzidas para os casos D e E, e dos casos anteriores para (a) Fase; (b) Neutro; (c) tomada 1.

## 4.1.5 – Caso F

Este caso é similar ao Caso C, porém os condutores fase e neutro do sistema de distribuição estão posicionados à mesma altura ( $z = 6,48\text{m}$ ) e o condutor neutro não foi aterrado, conforme mostra a Figura 4.25. O afastamento entre as linhas é de  $0,20\text{m}$ .

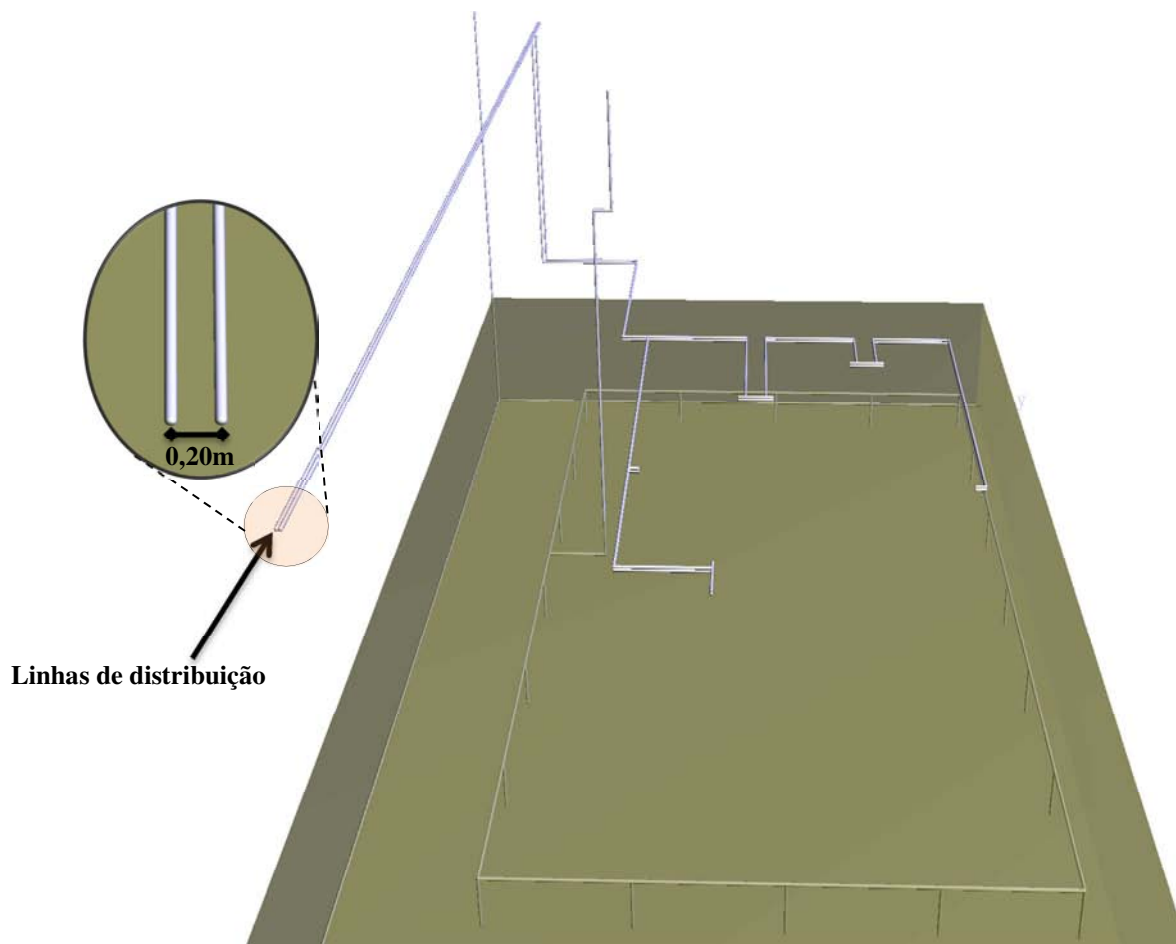


Figura 4.25: Definição do posicionamento das linhas para o Caso F.

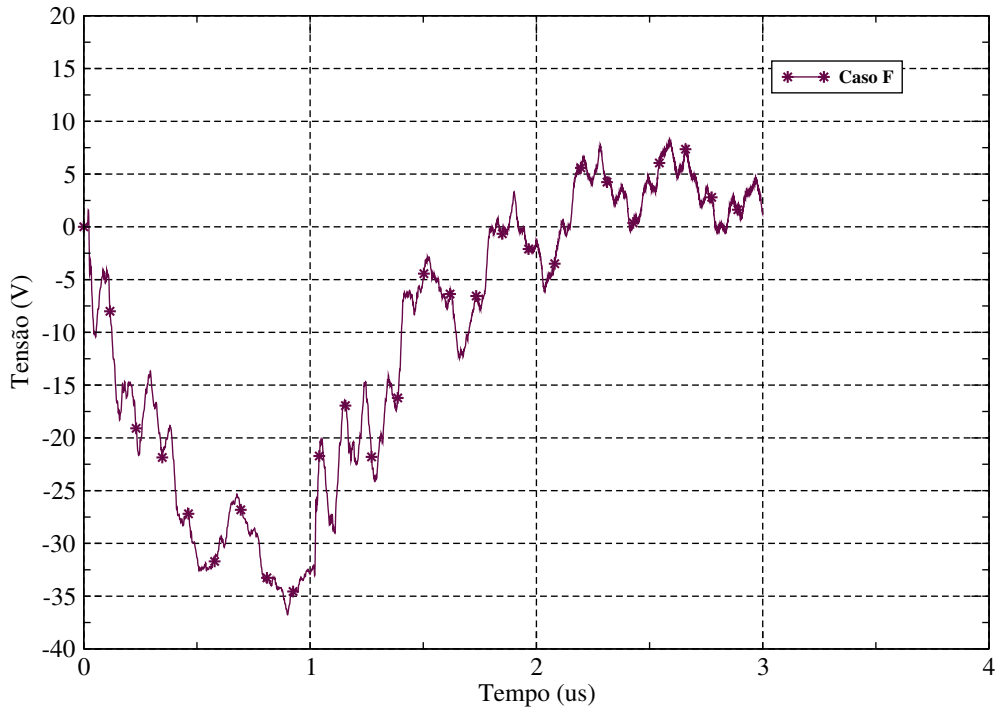
Na Figura 4.26, são apresentados os gráficos da tensão induzida obtida na tomada 1, fase e neutro para o Caso F, e, na Figura 4.27, são apresentados os resultados do Caso F juntamente com os resultados das simulações anteriores.

Quando os resultados apresentados pelas Figuras 4.26(a) e 4.16 são comparados, observa-se que o módulo da tensão induzida máxima na tomada 1 foi reduzida de 250 V para aproximadamente 35 V (redução de 86%). Isto é decorrente do fato de que, no presente caso, as linhas de distribuição estão à mesma altura diferente do caso da Figura 4.16, de tal forma que a diferença de tensão induzida entre os condutores fase e neutro é substancialmente reduzida (observe que as funções apresentadas pelas Figuras 4.26(b) e 4.26(c) são muito similares). Isto ocorre porque  $E_y$  (entre os fios deste caso) tende a ser menos intenso do que  $E_z$  (entre os fios do caso da Figura 4.16) devido à forma como o problema é excitado, ou seja, a corrente de excitação é paralela ao eixo  $z$  e é idêntica em ambos os casos. Além disso, como mostrado no Anexo E, a tensão induzida tende a ser reduzida devido à proximidade espacial

entre as linhas.

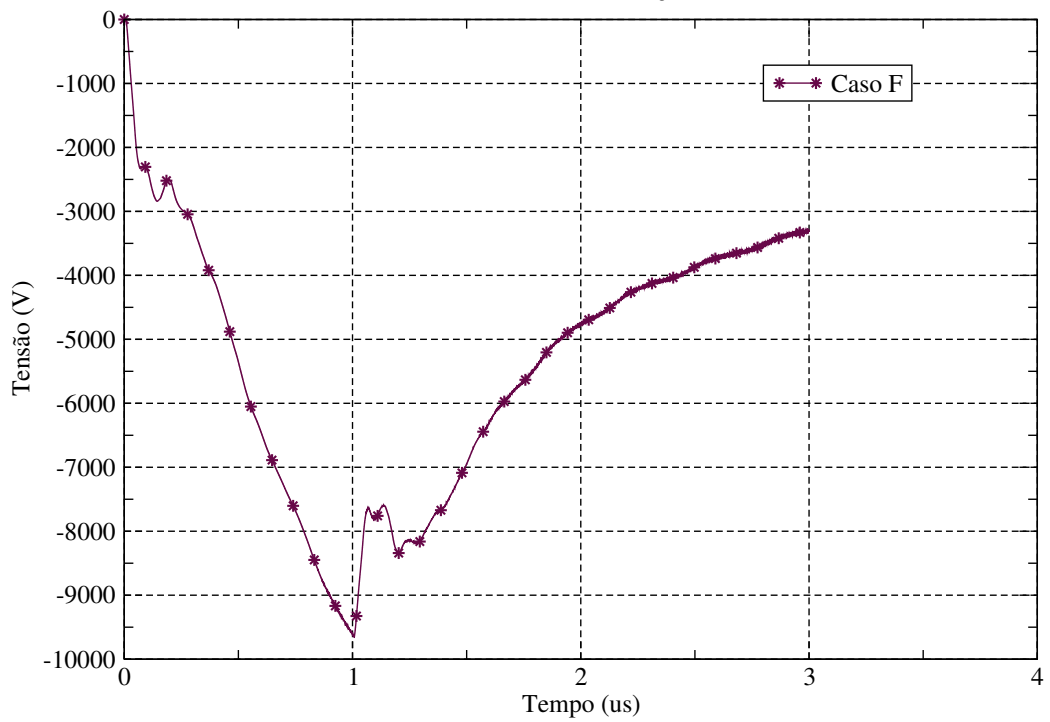
### Tensão Induzida na Tomada 1

Tomada da Residência



### Tensão Induzida na Fase

Linha de Distribuição



## Tensão Induzida no Neutro

Linha de Distribuição

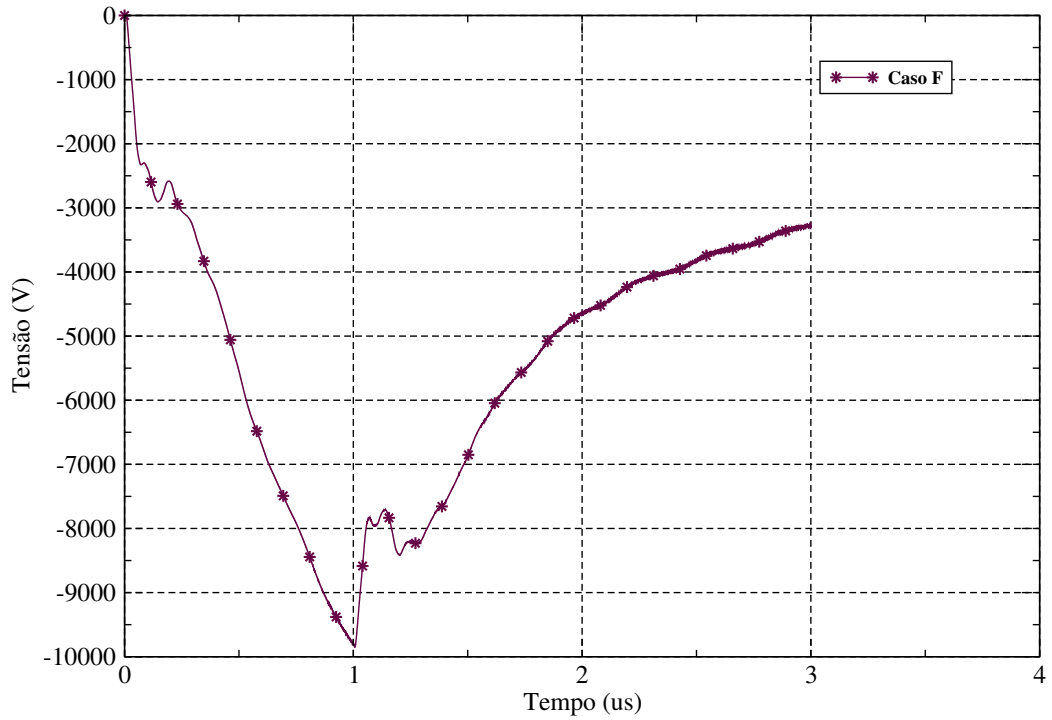
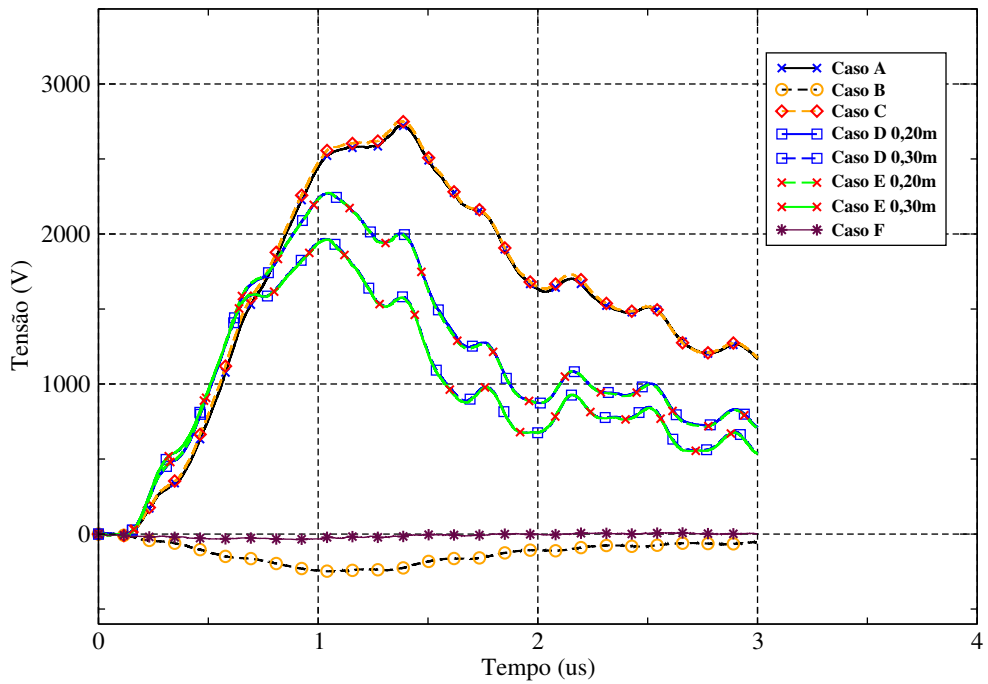


Figura 4.26: Tensões induzidas do Caso F para (a) Tomada 1; (b) Fase; (c) Neutro.

## Tensão Induzida na Tomada 1

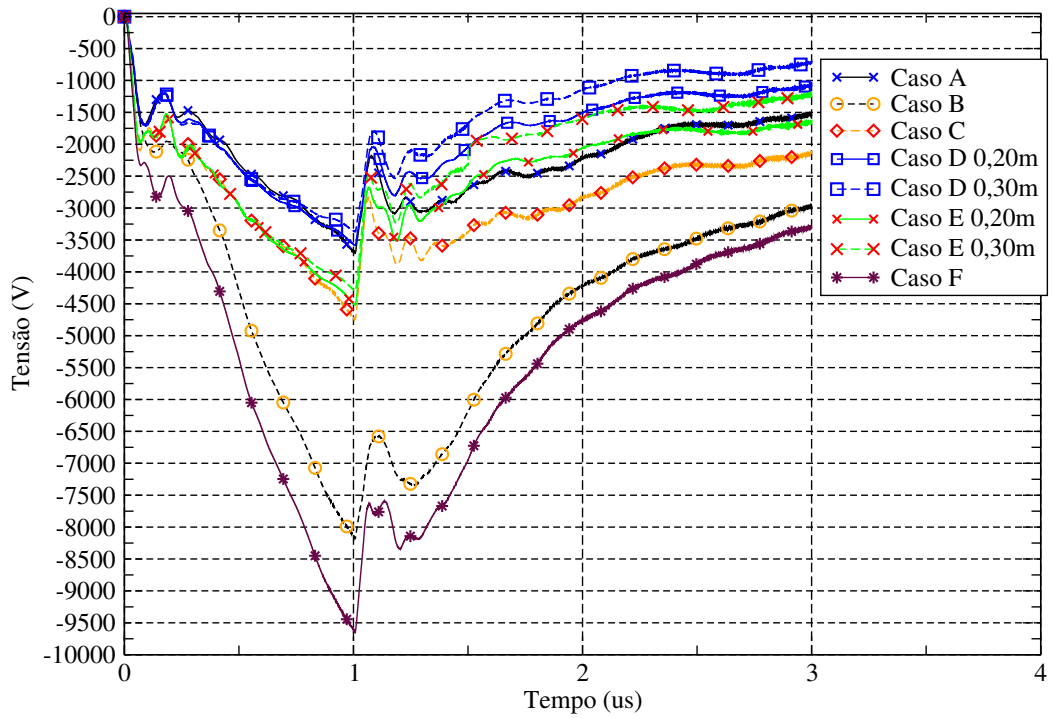
Tomada da Residência



(a)



### Tensão Induzida na Fase Linha de Distribuição



### Tensão Induzida no Neutro Linha de Distribuição

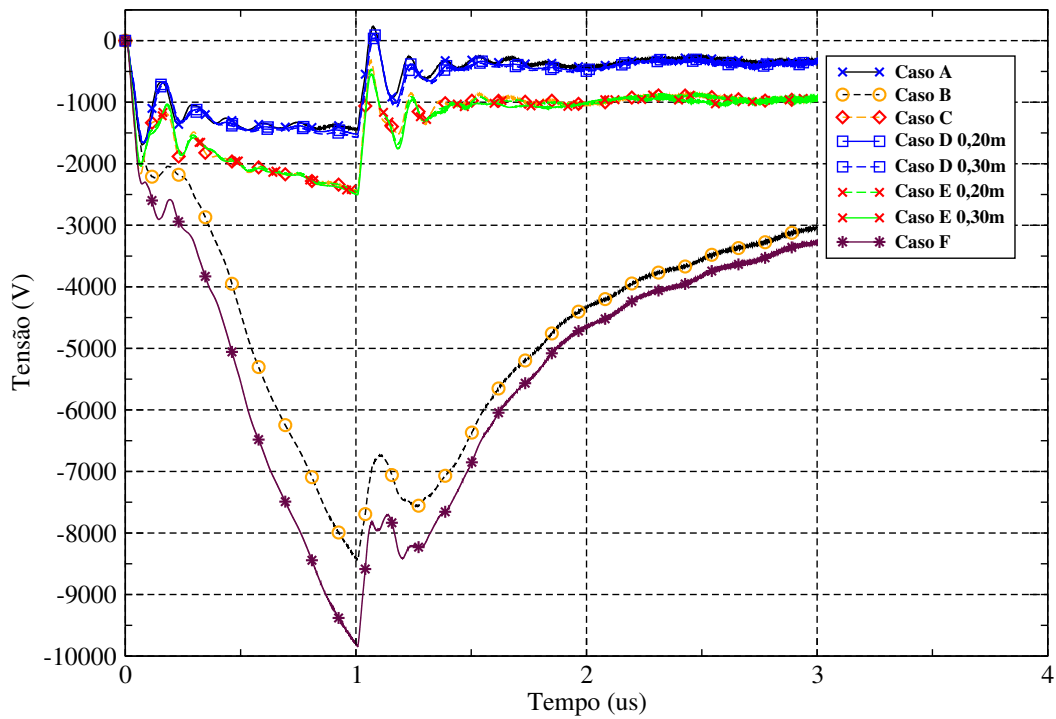


Figura 4.27: Tensões induzidas do Caso F e dos casos anteriores para (a) Tomada 1; (b) Fase; (c) Neutro.

No Apêndice E são apresentados casos adicionais que demonstram a flexibilidade e generalidade do *software* desenvolvido, além da consistência física dos resultados gerados.

## Capítulo 5 - Considerações Finais

Neste trabalho, é realizada a criação de uma interface gráfica de usuário (GUI), de um simulador baseado no método numérico LN-FDTD e de um visualizador de cenários e de malhas estruturadas 3D. O objetivo principal é simplificar a criação de cenários para a realização de simulações eletrodinâmicas paralelas ou sequenciais e, ao mesmo tempo, privar o usuário de interagir diretamente com o código-fonte do simulador.

A interface gráfica de usuário desenvolvida foi modelada através da ferramenta *Qt Designer* e o *design* da interface foi realizado cuidadosamente para que o usuário não tenha dificuldades em realizar as simulações. O simulador numérico baseado no método LN-FDTD foi desenvolvido utilizando a linguagem C e paralelizado com a utilização da biblioteca *pthread*. O visualizador de cenários 3D foi desenvolvido utilizando a linguagem C e a biblioteca gráfica *OpenGL*. Vale ressaltar, que o simulador numérico e o visualizador tridimensional estão em perfeita sincronia, sendo que o que é modelado no visualizador 3D é o que será simulado. Dessa forma, a representação tridimensional do cenário indica espacialmente a definição dos parâmetros dos meios. Esta definição é realizada através da GUI.

Para o desenvolvimento da ferramenta, foi utilizado o modelo de processo de *software* por prototipagem, com o objetivo de aumentar a qualidade do protótipo final, eficiência e, com isso, atender plenamente às definições do projeto do *software*.

Inicialmente foram realizadas simulações com a utilização de malhas retangulares para verificar a consistência dos resultados gerados pelo *software* simulador. Esses resultados foram comparados com aqueles gerados pela ferramenta LANE SAGS, mantendo sempre os mesmos parâmetros de simulação. Após a validação de diversos casos envolvendo coordenadas retangulares, simulações envolvendo o sistema de coordenadas gerais foram realizadas para validar plenamente a presente implementação. Para isto, foram considerados: um eletrodo de aterramento com geometria semi-esférica, um sistema de aterramento com eletrodos verticais conectados por um anel metálico e a análise de descargas parciais em um cabo coaxial. Dessa forma, as simulações realizadas neste trabalho complementam e asseguram o *software* desenvolvido como uma ferramenta que produz resultados confiáveis relativos à técnica numérica LN-FDTD.

O *software* aqui desenvolvido foi utilizado para analisar tensões induzidas em linhas de distribuição devido a descargas atmosféricas que atingem um pára-raios instalado em uma estrutura edificada (residência). Neste trabalho, foram considerados casos em que o captor é conectado à estrutura metálica da residência ou a um sistema de aterramento através de um condutor elétrico externo de descida. Tensões transitórias relacionadas ao fenômeno foram calculadas nas tomadas elétricas do prédio. Além disso, foram modeladas as curvaturas das linhas de distribuição que ocorrem devido ao efeito da gravidade (catenárias). Para tanto, malhas computacionais específicas foram desenvolvidas para representar essas geometrias.

De uma forma geral, nota-se que as tensões induzidas entre os terminais das tomadas dependem fortemente da posição relativa entre o ponto da descarga e das linhas de distribuição e independem da forma como a corrente elétrica da descarga é conduzida ao solo (através de um condutor externo ou através da estrutura metálica da residência), tal como mostrado pelos Casos A e C. Além disso, percebe-se que mudanças na disposição das linhas de distribuição (plano que contém as linhas) e catenárias afetam de forma significativa as tensões induzidas entre os terminais das tomadas. Em relação às catenárias, quanto maior o valor da flecha  $\Delta f$ , menor tende a ser a tensão induzida entre o solo e a fase de distribuição e nas tomadas, tal como mostrado nos Casos D e E. A tensão induzida no neutro é muito pouco afetada por  $\Delta f$  devido ao aterramento. Em relação à modificação do plano que contém as linhas de distribuição (Casos B e F), verificou-se que na situação considerada, as tensões induzidas obtidas no caso F, no qual o plano  $z = 6,28$  m contém as linhas, são aproximadamente 90% reduzidas em relação ao caso B, no qual o plano  $y=1,32$  m passa pelas linhas. Isso ocorre devido à polarização do campo elétrico relativa à fonte de corrente que modela a descarga atmosférica. Adicionalmente, verificou-se que o aterramento do condutor neutro tende a aumentar as tensões induzidas nas tomadas (entre fase e neutro), pois a conexão do neutro com a terra tende a reduzir a tensão induzida entre solo e o condutor neutro tal como bem evidenciado através da análise dos Casos A e B.

Como propostas de trabalhos futuros, sugerem-se: 1) utilização da biblioteca MPI combinada com *threads* para realizar o processamento paralelo do método LN-FDTD; 2) realizar processamento paralelo utilizando *Graphics Processing Unit* (GPUs); 3) aplicar métodos sem malhas para representar geometrias diversas; 4) modelagem de efeitos não lineares, tais como ionização do solo e do ar (devido às altas correntes da descarga atmosférica).

# APÊNDICE A

## Criando Interface Gráfica com *Qt Designer*

Inicie a ferramenta *Qt Designer*, selecione menu *File->New* e selecione a opção *C++ Project*. Clique em *OK* conforme mostra a Figura A.1.

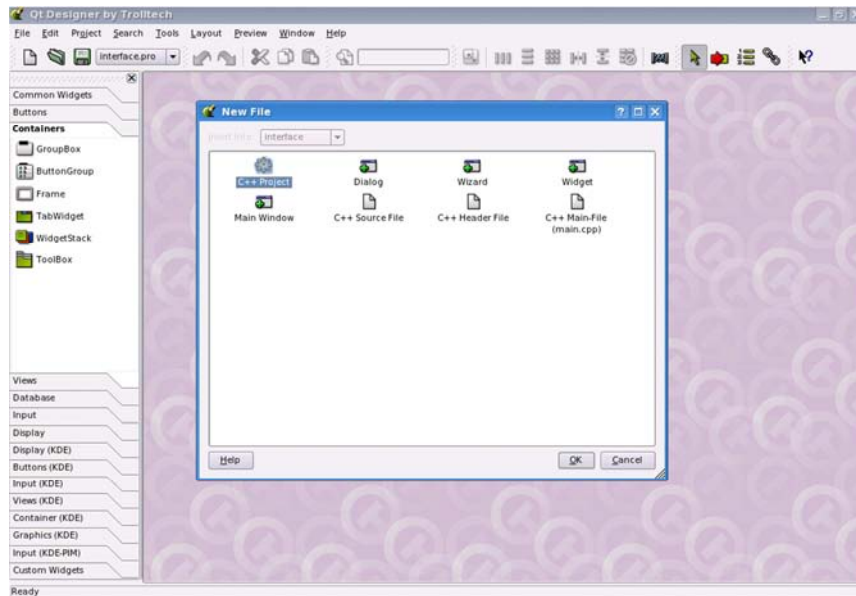


Figura A.1: *Qt Designer* - New File C++ Project.

Selecionada a tecla *OK*, a ferramenta irá pedir o nome do projeto “projeto.pro”. O usuário deverá fornecer este nome e clicar em *OK*.

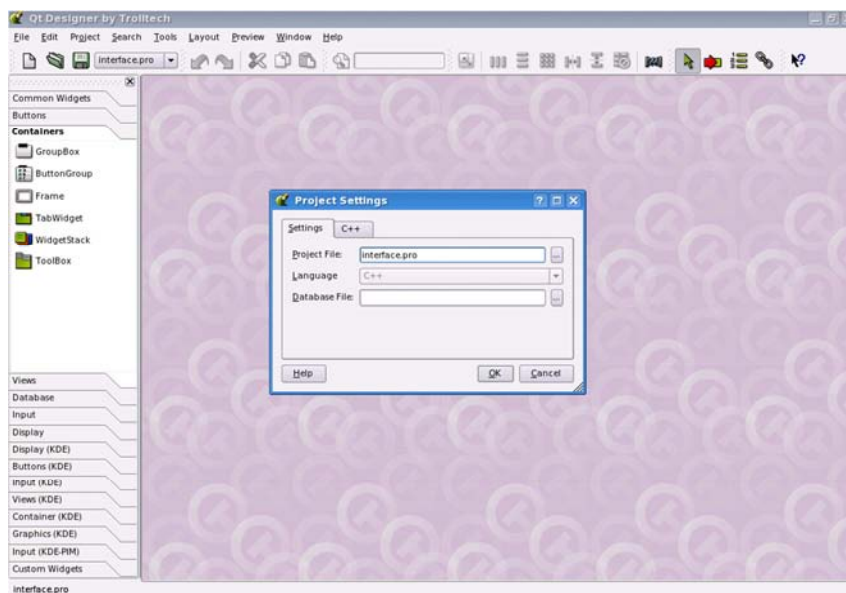


Figura A.2: *Qt Designer* - Project Settings.

Com o nome do projeto definido, será criada a janela principal da interface. Selecione o menu *File->New*, selecione a opção *Main Window* e clique em *OK*. Em seguida será apresentada uma tela com as opções de menu e barras de ferramentas que a interface irá dispor. Nesta tela de opções é possível adicionar ou remover menus e barras de ferramentas que a interface irá apresentar. Após definidas as opções, clique em *Next* para prosseguir.

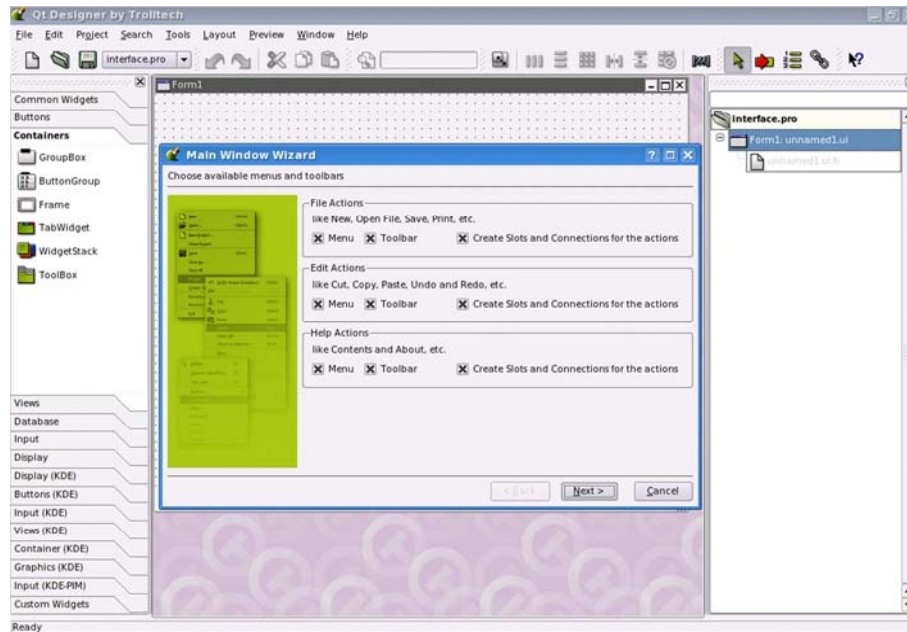


Figura A.3: *Qt Designer – Main Window Wizard*.

Neste passo serão definidos os itens dos menus principais. Na Figura abaixo, observe que o Menu *File*, que está selecionado em *Category*, irá conter como itens *New*, *Open*, *Save*, *Save As* e <Separator> (que indica que haverá uma linha separando os menus). Em *Category*, serão encontrados os outros menus como *Edit* e *Help*. Estes seguem os mesmos passos do menu *File*. Após o término das devidas configurações, clique em *Next* e em seguida em *Finish*.

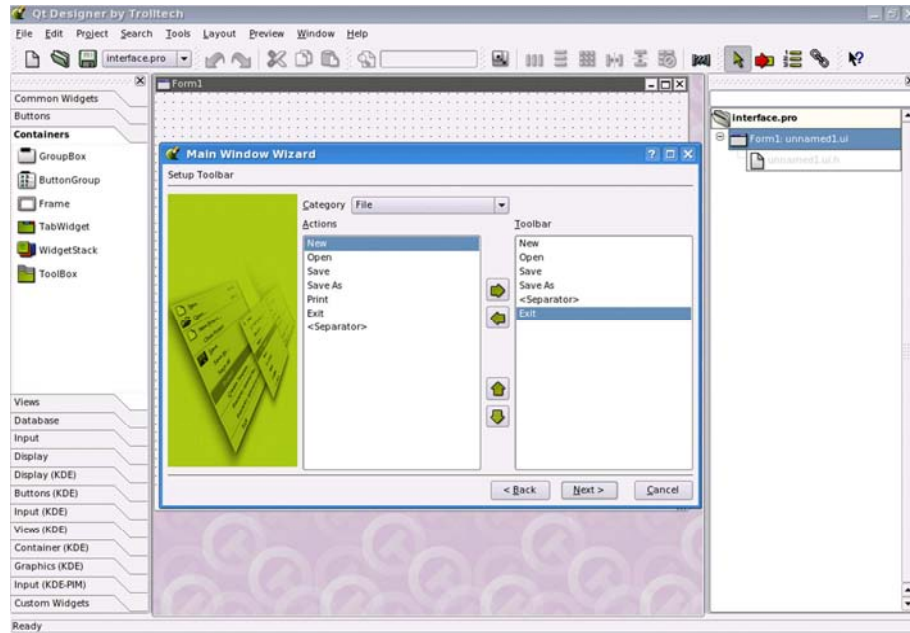


Figura A.4: Qt Designer – Main Window Wizard – Setup Toolbar.

Selecione o menu *File->New*, clique em *C++ Main-File* e clique em *OK*. Este é o arquivo *main.cpp* que contém as instruções para que a interface seja executada com sucesso. Sem este arquivo na hora de compilar, ocorrerá um erro indicando a falta do arquivo *main.cpp*. Depois de clicar em *OK*, a ferramenta irá perguntar qual a interface principal a que este arquivo *main.cpp* se refere, conforme mostra a Figura A.5, no caso de haver mais de uma interface.

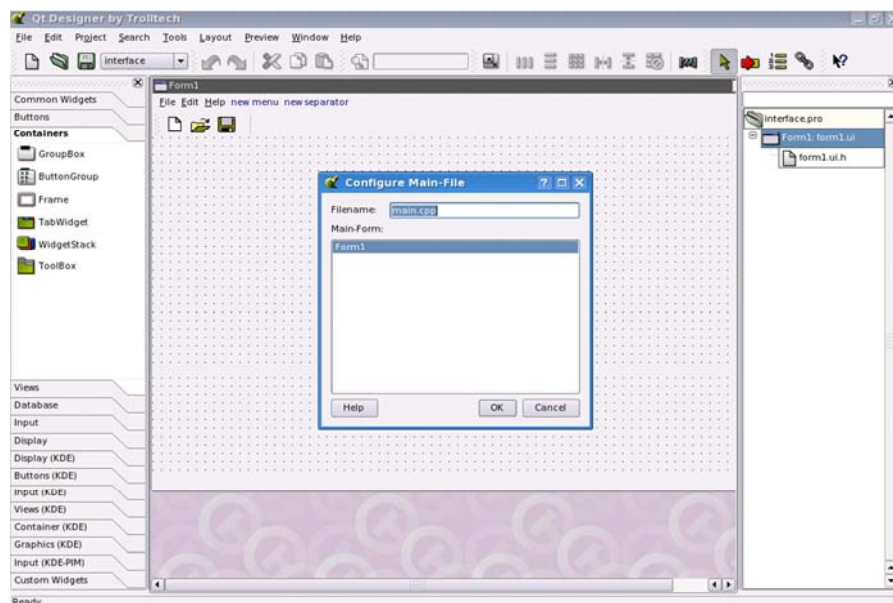


Figura A.5: Qt Designer – Configure Main-File.

Agora serão adicionados os widgets (componentes da interface gráfica, como janelas, botões, menus, ícones e etc.), a interface e também a implementação de suas ações e eventos originados a partir da interação do usuário com a interface. Como um exemplo simples, vamos adicionar dois botões (ver Figura A.6) e fazer estes executarem uma ação. Este exemplo, apesar de simples, é suficiente para o entendimento de como fazer a ação do botão. Por exemplo, ao ser clicado pelo mouse na interface, o botão gera uma ação e esta é definida pelo desenvolvedor. Tal como que é feito este procedimento para um botão, o mesmo se aplica a qualquer outro componente da interface que origine uma ação ao ser clicado pelo mouse.

Para adicionar um botão ou qualquer outro componente à interface, clique em uma das opções na barra de ferramenta do lado esquerdo do *Qt Designer* e arraste o objeto para dentro da janela principal da interface. Feito isso, será possível mover, alterar tamanho, mudar o texto que aparece no componente, entre outros, de acordo com a característica do objeto escolhido. Esses parâmetros são definidos baseados no *layout* que o desenvolvedor desejar.

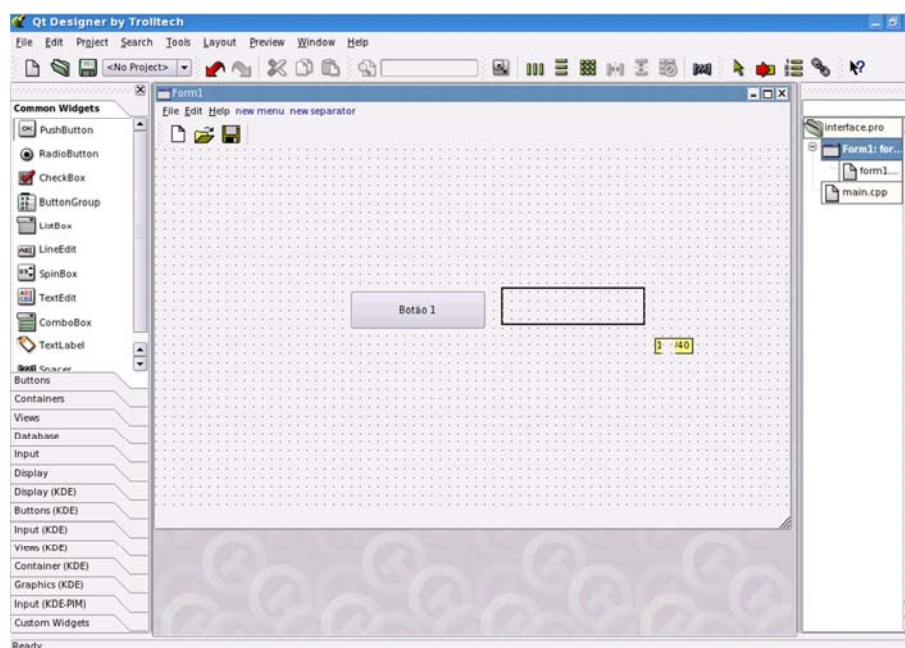


Figura A.6: *Qt Designer* – Adicionando *Widgets*.

Todos os parâmetros de configuração dos componentes adicionados à interface estão localizados na barra de ferramenta *Property Editor/Signal Handlers*, que fica ao lado direito do *Qt Designer*, conforme mostra a Figura A.7.



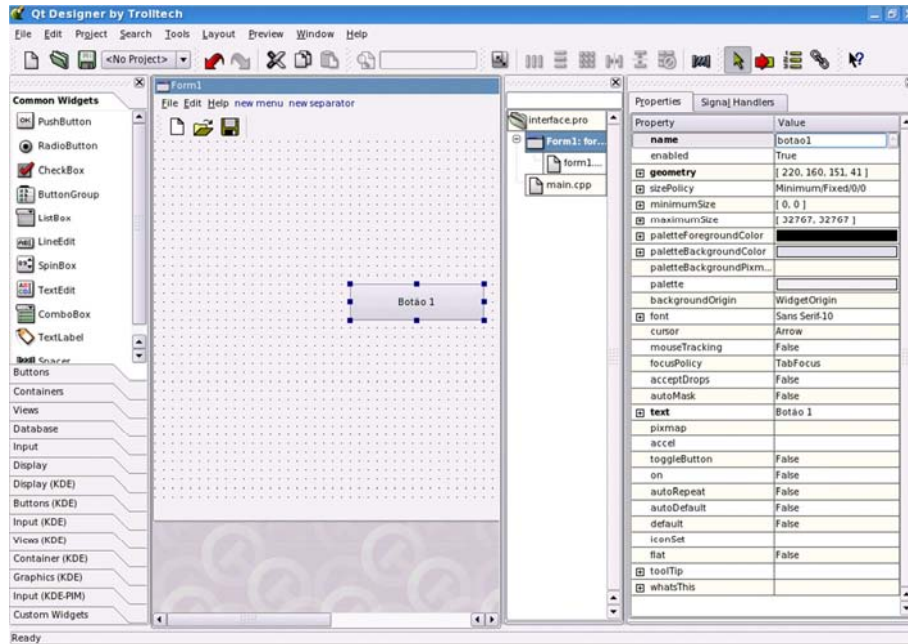


Figura A.7: Qt Designer - Property Editor/Signal Handlers.

Após a inserção de todos os componentes da interface, iremos criar a ação ao se clicar no botão criado na interface. Selecione menu *Edit->Slots*, clique em *New Function*, forneça um nome para a função e clique em *OK*.

Neste exemplo, serão criadas duas funções (uma para cada botão) conforme mostra a Figura A.8.

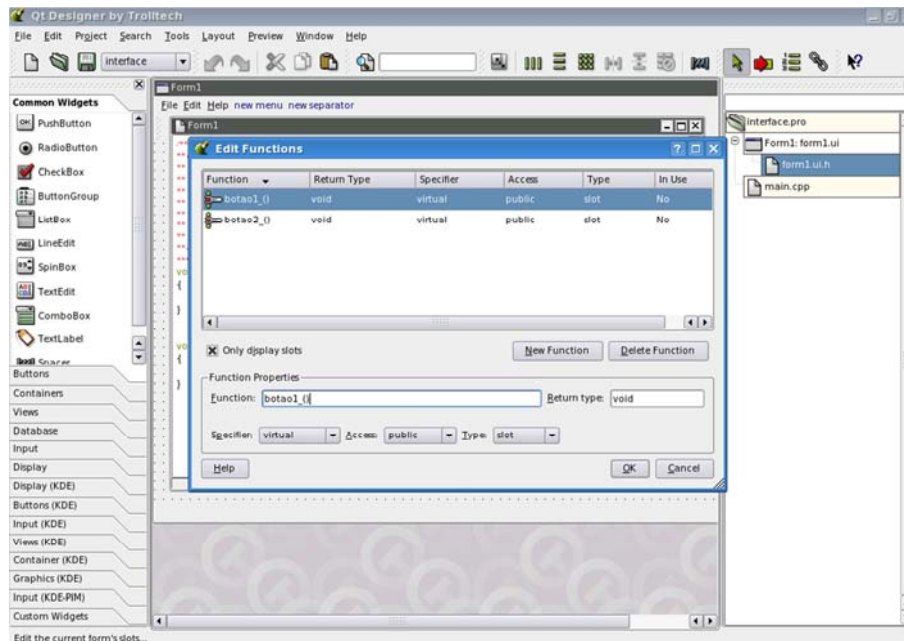


Figura A.8: Qt Designer – Edit Functions.

Depois de criadas as funções para cada componente de interface, será realizada a conexão do componente com a sua respectiva função (Figura A.9). Selecione menu *Edit->Connections* e clique em *New*. Na coluna *Sender*, será escolhido o componente da interface que executará a ação. Na coluna *Signal*, será escolhido a forma como deve ser detectada a maneira como o componente será ativado. Na coluna *Receiver*, será escolhido o componente que receberá esta ação. Neste exemplo, seria *Form1* a própria interface. E para finalizar, define-se o *Slot*, que é a função que será responsável por executar as instruções quando o componente escolhido for acionado através da interface.

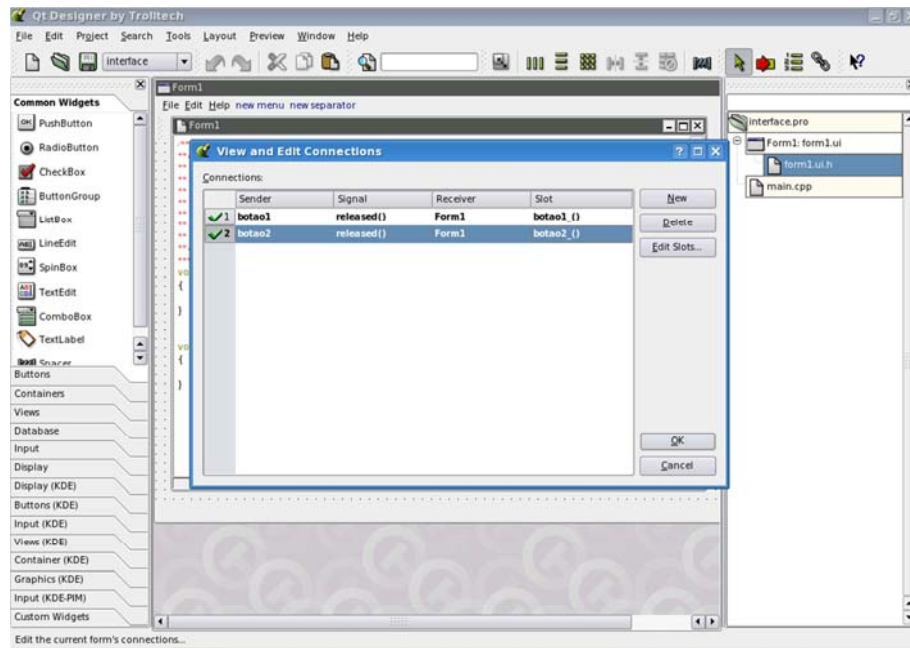


Figura A.9: *Qt Designer – View and Edit Connections.*

Depois de todas as configurações definidas, iremos implementar a ação do componente escolhido. Para isso, deve-se abrir o arquivo *form1.ui.h* (Figura A.10), que é onde se encontram as funções definidas no *Slot*. Dentro dessas funções, devem ser inseridas as instruções que o componente irá executar quando for acionado.

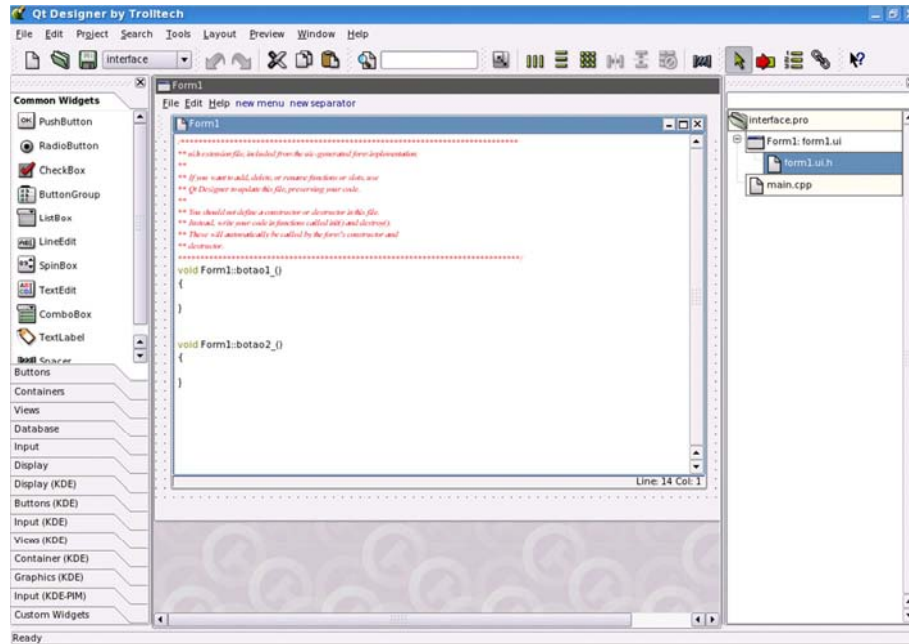


Figura A.10: *Qt Designer – form1.ui.h.*

Como exemplo simples para testar a ação que o componente botão irá executar, será implementada a seguinte ação: quando o botão 1 for pressionado, ele será desabilitado e habilitará o botão 2. Quando o botão 2 for pressionado, ele será desabilitado e habilitará o botão 1, conforme a Figura A.11.

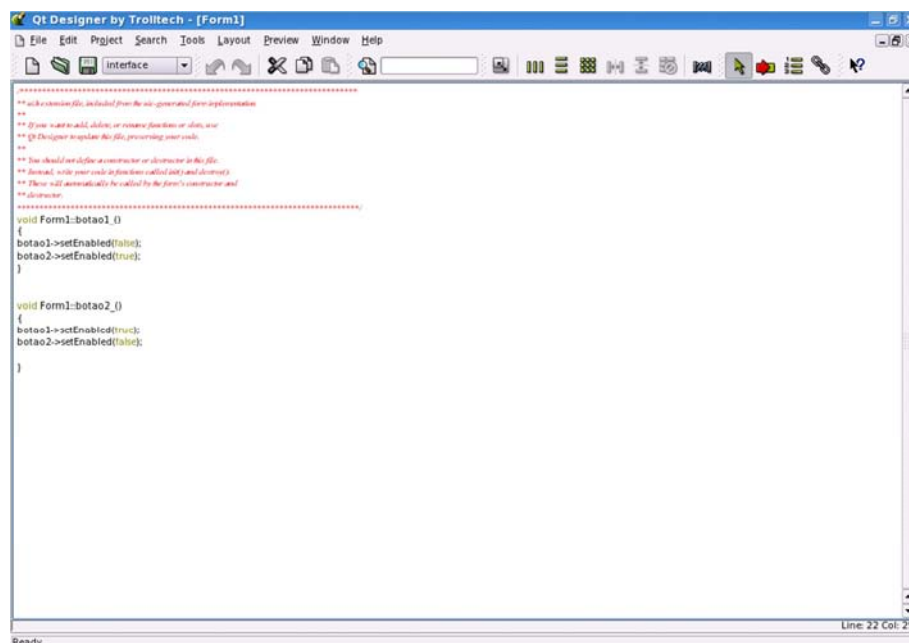


Figura A.11: *Qt Designer – form1.ui.h com implementação da ação dos componentes.*

Depois de todas as instruções implementadas em suas funções, o projeto deverá ser

salvo e em seguida compilado. Para compilar o projeto, abra o console e entre na pasta em que o mesmo está salvo e digite o comando: *qmake*; *make*.

O *qmake* é um utilitário que está incluído no pacote de ferramentas do Qt. Sua função é criar *makefiles* a partir de projetos (.pro) em diferentes plataformas [12], e o comando *make* é uma ferramenta que controla a geração de executáveis a partir de arquivos de origem de um programa fonte: os *makefiles*.

Após a compilação do programa será gerado o arquivo executável que tem o mesmo nome base do projeto. Como o projeto criado neste exemplo chama-se *interface.pro*, então para executá-lo digite no console o comando: *./interface*. Com isso, a interface será executada e visualizada no sistema operacional, conforme a Figura A.12.

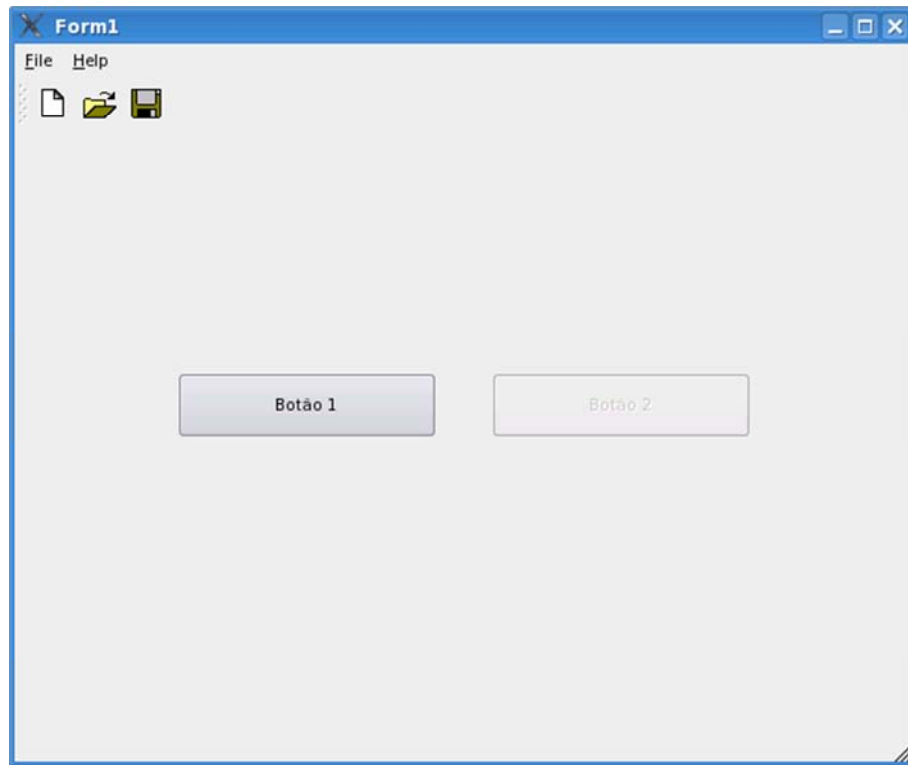


Figura A.12: Interface.

Este exemplo serve como base para a construção de qualquer interface em Qt, pois a adição de componentes em um projeto.pro em Qt sempre precisará estes passos.

# APÊNDICE B

## Desenvolvendo Aplicações Gráficas 3D com *Opengl*

Para desenvolver aplicações em *OpenGL* no Linux, é necessário verificar se as bibliotecas *OpenGL* estão incluídas no sistema. Normalmente, as principais distribuições Linux do mercado já possuem as bibliotecas *OpenGL* incluídas em suas instalações. Caso o sistema não possua, as mesmas podem ser obtidas no site <http://www.opengl.org>.

Após a verificação da existência e/ou instalação das bibliotecas *OpenGL*, a aplicação poderá ser desenvolvida.

Antes do início de qualquer implementação de código, deve ser garantido que as funções das bibliotecas *OpenGL* sejam reconhecidas pelo compilador. Com isso, basta inserir a seguinte diretiva de inclusão padrão da linguagem C:

```
#include<GL/glut.h>
```

A biblioteca *glut.h* gerencia eventos de teclado, mouse e *joystick* por meio de funções, que são chamadas de funções de *callback*. Isto é, quem chama uma função para tratamento de eventos de teclado, por exemplo, não é o programador e sim a *glut*. Com isso, o programador precisa especificar qual o nome da função de *callback* para cada evento a ser tratado [13].

A seguir serão descritas as funções básicas e essenciais para qualquer implementação gráfica tridimensional utilizando *OpenGL*.

### B.1 Função *Display*

Esta função é responsável por desenhar ou redesenhar a tela quando for necessário e contém a implementação das primitivas de desenho que será visualizado no ambiente tridimensional. Um exemplo de implementação é mostrado pela Figura B.1.

```

// Função callback de redesenho da janela de visualização
void Display(void)
{
    // Limpa a janela de visualização com a cor de fundo definida
    previamente
    glClear(GL_COLOR_BUFFER_BIT);

    // Altera a cor do desenho para preto
    glColor3f(0.0f, 0.0f, 0.0f);

    // Função da GLUT para fazer o desenho de um cubo
    // com a cor corrente
    glutWireCube(50);

    // Executa os comandos OpenGL
    glFlush();
}

```

Figura B.1: Função *Display* [13].

## B.2 Função *Keyboard*

Assim como a função *Display*, esta função é chamada quando ocorre algum evento. Por exemplo, quando o usuário pressionar uma tecla ASCII, esta se estiver sendo tratada na implementação, *Keyboard* deverá realizar alguma ação em resposta ao evento. Isto é ilustrado pela Figura B.2 para a tecla Esc.

```

void Keyboard (unsigned char key, int x, int y)
{
    if (key == 27)
        exit(0);
}

```

Figura B.2: Função *Keyboard* [13].

Note que nesta função é verificado se o usuário pressionou a tecla ESC, e caso seja verdadeiro a execução do programa é encerrada.

## B.3 Função *Init*

Esta função é responsável por inicializar os parâmetros e variáveis do programa, e é chamada apenas durante a inicialização do programa. Sua implementação é ilustrada pela Figura B.3.

```

void Init (void)
{
    // Define a cor de fundo da janela de visualização como branca
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glLineWidth(2.0);
}

```

Figura B.3: Função *Init* [13].

## B.4 Programa Principal

As chamadas das funções descritas acima são realizadas no programa principal (*void main*), descrito pela Figura B.4.

```

// Programa Principal
int main(void)
{
    // Define do modo de operação da GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // Especifica a posição inicial da janela GLUT
    glutInitWindowPosition(5,5);
    // Especifica o tamanho inicial em pixels da janela GLUT
    glutInitWindowSize(450,450);
    // Cria a janela passando como argumento o título da mesma
    glutCreateWindow("Desenho de um cubo");
    // Registra a função callback de redesenho da janela de visualização
    glutDisplayFunc (Display);
    // Registra a função callback para tratamento das teclas ASCII
    glutKeyboardFunc (Keyboard);
    // Chama a função responsável por fazer as inicializações
    Init();
    // Inicia o processamento e aguarda interações do usuário
    glutMainLoop();
    return 0;
}

```

Figura B.4: Programa Principal *main* [13].

É importante tentar compilar e executar corretamente esse programa, para saber se as bibliotecas da *OpenGL* estão instaladas e funcionando corretamente no sistema. Para compilar e executar o programa corretamente utilize os comandos a seguir.

Para realizar esses comandos utilize o console na pasta onde se encontra o arquivo a ser compilado.

Para compilar: gcc nome\_do\_programa.c -o nome\_do\_binário -lGL -lGLU -lglut

Para executar: ./nome\_do\_binário

O resultado da execução do programa pode ser visto na Figura B.5.

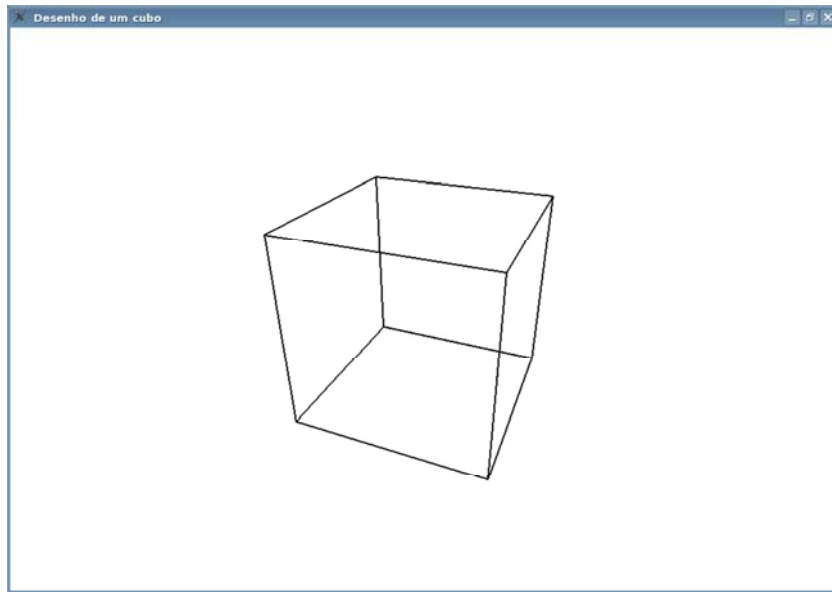


Figura B.5: Desenho de um cubo.



# APÊNDICE C

## Desenvolvimento de Interface Gráfica como Suporte para Soluções Numéricas das Equações de Maxwell

Adolfo F. de O. Colares, Rodrigo Lisboa Pereira, Rodrigo M. S. de Oliveira, Carlos Leonidas de S.S. Sobrinho  
Programa de Pós-Graduação em Engenharia Elétrica - PPGEE  
Universidade Federal do Pará - UFPA  
Caixa Postal 8619 – 66.075-900 Belém – Pará – Brasil  
{adolfo, rlp, rmso, leonidas}@ufpa.br

**Abstract** – This article proposes the creation of a Graphical User Interface (GUI) by using the Qt toolkit from Nokia, to simplify the scenery creation for performing Parallel simulations by using the numerical technique Local Nonorthogonal Finite-Difference Time-Domain (LN-FDTD). The simulator was written by using the C programming language and the visualization of the 3D scenario to be simulated (and of the associated mesh) is performed by a viewer specially developed by employing the OpenGL library. By depriving the user to interact directly with the source code of the simulation, the probability of occurrence of human errors during the building process of the scenarios is reduced. In order to demonstrate the developed tool operation, non-orthogonal grounding systems were simulated.

**Keywords**–Graphical User Interface; scenery creation automation; LN-FDTD Method; Parallel Processing;

### I. INTRODUÇÃO

Problemas relacionados com eletromagnetismo podem ser solucionados utilizando diversos métodos, classificados como: analíticos, experimentais e numéricos [1]. Dentre estes grupos, destacam-se os métodos numéricos por, em geral, apresentarem soluções mais flexíveis em termos de geometria e parâmetros para solucionar as equações de Maxwell [2,3].

Dentre os métodos numéricos existentes para a solução de problemas em eletromagnetismo, podemos mencionar o FDTD (*Finite-Difference Time-Domain*) [3], MoM (*Method of Moments*) [2], FEM (*Finite Element Method*) [4] e finalmente o método LN-FDTD (*Local Nonorthogonal Finite-Difference Time-Domain*) [5], no qual este trabalho se baseia.

O método FDTD foi publicado por Yee em 1966 e é fundamentado na transformação das equações diferenciais escalares das componentes dos campos elétrico e magnético, no sistema Cartesiano, em um grupo de equações de diferenças finitas centradas [5]. O método LN-FDTD é uma adaptação do FDTD, no qual malhas estruturadas com células ortogonais são utilizadas de forma a modelar geometrias não compatíveis com o sistema Cartesiano de coordenadas, evitando aproximações do tipo *staircase* [3], muitas vezes presentes no método FDTD. Assim como o FDTD clássico, o método LN-FDTD gera soluções de onda completa, sendo que a sua robustez e estabilidade estão diretamente ligadas à malha utilizada para representar o problema geometricamente [5].

Existem técnicas de GUI e de visualização tridimensional para a maioria dos métodos numéricos mencionados anteriormente, tais como a ferramenta FEMLAB [6], que utiliza um simulador baseado na técnica FEM. Em [7] é apresentada a ferramenta SAGS baseada na técnica FDTD. Em

[8] encontra-se a ferramenta HARP baseada na técnica MoM e, até então não foi encontrada qualquer técnica de GUI ou visualização tridimensional desenvolvida para utilizar um simulador baseado na técnica LN-FDTD.

Neste trabalho, foi desenvolvida uma interface gráfica para usuários usando a ferramenta Qt 3 [9]. Esta interface tem como objetivo dar suporte (entrada e saída de informação) para simulações paralelizadas e automatizadas de problemas de eletromagnetismo aplicado. Na formulação do problema, as equações de Maxwell são resolvidas numericamente através do método LN-FDTD, em Coordenadas Gerais. Neste trabalho, também foi implementado um visualizador de cenários e de malhas de discretização tridimensionais com os recursos da biblioteca OpenGL [10], o que certamente dará aos usuários uma maior visibilidade do problema em questão. A ferramenta desenvolvida e suas funções seguem o padrão *Unix* e foi implementada e testada em ambiente Linux com KDE 3.5 ou superior e Qt 3.3 ou superior. O objetivo principal da ferramenta é proporcionar ao usuário transparência no que está sendo modelado no simulador e, conseqüentemente, afastar o engenheiro do desenvolvimento de códigos fontes complexos, reduzindo as possibilidades de erro humano durante a fase de concepção das estruturas. Vale ressaltar que o ambiente computacional desenvolvido pode ser aplicado nas soluções de problemas 3D independentemente de haver ou não simetria, para meios homogêneos ou estratificados, oferecendo possibilidades para saídas numéricas, gráficas ou através de filme (evolução temporal do processo eletromagnético).

O trabalho está organizado da seguinte maneira: é apresentada a teoria sobre o método numérico LN-FDTD em II, em III são mostrados a interface e o visualizador 3D desenvolvidos, em IV são mostrados e discutidos os resultados obtidos e em V são feitas as considerações finais.

### II. FUNDAMENTAÇÃO TEÓRICA

#### A. O Sistema de Coordenadas Gerais

Considerando um sistema de coordenadas gerais ou curvilíneas, o vetor posição  $\vec{r}$ , relativo a um ponto P (Figura 1), pode ser obtido em um sistema de coordenadas gerais, tal que o vetor de comprimento diferencial  $d\vec{r}$  é dado por (1) [3].

$$d\vec{r} = \sum_{l=1}^3 \frac{\partial \vec{r}}{\partial u^l} du^l = \sum_{l=1}^3 \vec{a}_l du^l, \quad (1)$$

Nesta equação, os vetores  $\vec{a}_l$  são chamados de vetores unitários e formam uma base unitária, que define os eixos curvilíneos gerais que passam por um ponto P no espaço (cada ponto tem seu próprio sistema de coordenadas). Na Figura 1,

estão representados os vetores  $\vec{a}^l$  ( $l = 1, 2, 3$ ) que são tangentes aos eixos  $u^l$  e podem ser escritos como funções das coordenadas cartesianas  $x, y$  e  $z$ . Um conjunto alternativo de três vetores complementares  $\vec{a}^l$ , denominados de vetores recíprocos, pode ser definido de modo que cada um é normal a dois vetores unitários com diferentes índices [5], formando assim uma base recíproca. Esse conjunto pode ser matematicamente calculado pela expressão:

$$\vec{a}^1 = \frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}}, \vec{a}^2 = \frac{\vec{a}_1 \times \vec{a}_3}{\sqrt{g}}, \vec{a}^3 = \frac{\vec{a}_1 \times \vec{a}_2}{\sqrt{g}} \quad (2)$$

em que  $\sqrt{g}$  é o volume do hexaedro  $\vec{a}_1, \vec{a}_2$  e  $\vec{a}_3$  formado pelos vetores  $\vec{a}_1, \vec{a}_2$  e  $\vec{a}_3$  (Figura 1). Em (2),  $g$  é o determinante da matriz métrica ou tensor covariante [5].

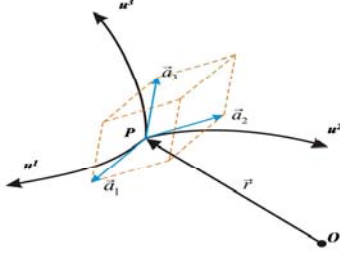


Figura 1. O sistema de coordenadas curvilíneas no ponto P e unidades de vetores em uma célula não-ortogonal.

Com base na idéia apresentada anteriormente, um vetor  $\vec{F}$  pode ser representado pelo sistema unitário ou pelo sistema recíproco, como mostra a expressão (3).

$$\vec{F} = \sum_{l=1}^3 f^l \cdot \vec{a}_l = \sum_{l=1}^3 f_l \cdot \vec{a}^l. \quad (3)$$

Em (3),  $f_l$  e  $f^l$  são chamadas de componentes covariante e contravariantes do vetor  $\vec{F}$ , respectivamente. As componentes  $f^l$  e  $f_l$  podem ser calculadas por meio do produto escalar de  $\vec{F}$ , na equação anterior, por  $\vec{a}^l$  e  $\vec{a}_l$  respectivamente, como é mostrado pela expressão (4).

$$f^l = \vec{F} \cdot \vec{a}^l, f_l = \vec{F} \cdot \vec{a}_l. \quad (4)$$

Estas componentes (covariante e contravariante) podem ser relacionadas por meio das seguintes expressões:

$$f_m = \sum_{l=1}^3 g_{lm} \cdot f^l, \quad f^m = \sum_{l=1}^3 g^{lm} \cdot f_l, \quad (5)$$

em que  $g_{lm} = \vec{a}_l \cdot \vec{a}_m$  e  $g^{lm} = \vec{a}^l \cdot \vec{a}^m$ . É importante ressaltar que os vetores unitários e os recíprocos não têm necessariamente amplitudes unitárias porque eles dependem da natureza do sistema de coordenadas curvilínea (comprimento das arestas das células). Hexaedros similares aos da Figura 1 são usados como células de Yee. Dessa forma, um conjunto apropriado de vetores unitários e seus respectivos comprimentos unitários são definidos pelas equações elementares

$$\vec{i}_1 = \frac{\vec{a}_1}{\sqrt{\vec{a}_1 \cdot \vec{a}_1}} = \frac{\vec{a}_1}{\sqrt{g_{11}}}, \vec{i}_2 = \frac{\vec{a}_2}{\sqrt{g_{22}}}, \vec{i}_3 = \frac{\vec{a}_3}{\sqrt{g_{33}}}. \quad (6)$$

A partir de (3), pode-se escrever

$$\vec{F} = F^1 \vec{i}_1 + F^2 \vec{i}_2 + F^3 \vec{i}_3, \quad (7)$$

em que  $F^l$  representa o valor físico da componente no sistema de base e seus valores são calculados por (8).

$$F^l = f^l \cdot \sqrt{g_{11}}, F_l = \sqrt{g^{ll}}. \quad (8)$$

Essa representação dos vetores pode ser utilizada para descrever as componentes dos campos elétrico e magnético, como visto a seguir.

#### A. O método LN-FDTD aplicado para solucionar as Equações de Maxwell

As equações de Maxwell na forma diferencial podem ser escritas para meios com perdas, isotrópicos e não dispersivos da seguinte maneira, no domínio do tempo:

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} \quad e \quad \nabla \times \vec{H} = \sigma \vec{E} + \epsilon \frac{\partial \vec{E}}{\partial t}, \quad (9)$$

em que  $\vec{E}$  é o vetor intensidade de campo elétrico,  $\vec{H}$  é o vetor intensidade de campo magnético,  $\epsilon$  é a permissividade elétrica,  $\mu$  é a permeabilidade magnética e  $\sigma$  é a condutividade elétrica, do meio. Calculando a componente contravariante dos campos  $\vec{E}$  e  $\vec{H}$ , utilizando diferenças centradas para termos derivativos e observando as células primárias e secundárias na Figura 2, podem ser obtidas as equações para atualização das componentes de  $\vec{E}$  e  $\vec{H}$ , tal como no algoritmo de Yee original.

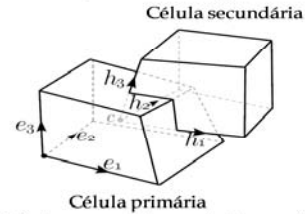


Figura 2. Distribuição das componentes covariantes nas células não ortogonais Yee, célula primária para as componentes do campo elétrico ( $\vec{e}_1, \vec{e}_2, \vec{e}_3$ ) e célula secundária para as componentes do campo magnético ( $\vec{h}_1, \vec{h}_2, \vec{h}_3$ ).

Para obtenção de uma componente do campo (por exemplo, a primeira componente contravariante de  $\vec{H}$ , ( $h^1$ )), (4) e (2) podem ser usadas com  $l = 1$ . Nesse caso, a operação do produto escalar é realizada na lei de Faraday. Isto é:

$$(\nabla \times \vec{E}) \cdot \frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}} = -\mu \left( \frac{\partial \vec{H}}{\partial t} \right) \cdot \vec{a}^1. \quad (10)$$

A partir da qual é fácil ver que

$$\frac{\partial h^1}{\partial t} = -\frac{1}{\mu \sqrt{g}} \left( \frac{\partial e_3}{\partial u_2} - \frac{\partial e_2}{\partial u_3} \right). \quad (11)$$

Observado isto, a partir de (2) é possível dizer que  $\vec{a}^m \cdot \vec{a}_n = \delta_{m,n}$ , em que  $\delta_{m,n}$  é o delta de Kronecker. A Equação (11) e as equações para as outras cinco componentes de campo podem ser representadas por diferenças finitas utilizando o algoritmo de Yee [5] (com derivadas centradas).

#### B. A Estabilidade e Precisão do Método LN-FDTD

A estabilidade numérica do método está relacionada ao incremento  $\Delta t$ , que segue as condições dadas pela expressão (12) [3], em que  $V_m$  é a máxima velocidade de propagação.



$$\Delta t \leq \frac{1}{V_m \cdot \max \left( \sqrt{\sum_{l=1}^3 \sum_{m=1}^3 g^{lm}} \right)} \quad (12)$$

Para reduzir os efeitos da dispersão numérica para níveis adequados e para garantir a precisão dos cálculos, as dimensões de cada célula não-ortogonal em todas as direções devem ser inferiores a  $\lambda/10$  [3], onde  $\lambda$  é o menor comprimento de onda envolvido no problema.

A UPML (*Uniaxial Perfectly Matched Layers*) para meios condutivos é usada para o truncamento do método LN-FDTD, que consiste em um conjunto de camadas perfeitamente casadas e com anisotropia uniaxial [5].

#### A. Processamento Paralelo para o método LN-FDTD

O ambiente para que o simulador desenvolvido funcione corretamente consiste de: 1) um *cluster* do tipo Beowulf; 2) configuração do tipo *Master-Slaves*; 3) suporte para o padrão de interface de comunicação LAM/MPI (*Message Passing Interface*); 4) serviço RSH (*Remote Shell*) configurado para operar sem senha; 5) NSF (*Network File System*) para compartilhamento dos dados.

A principal idéia para utilizar a computação paralela para resolver problemas eletromagnéticos pelo LN-FDTD é baseada na divisão espacial do domínio de análise em subdomínios (Figura 3). Cada subdomínio é uma fração de todo o volume numérico que será tratado por um único núcleo de processamento. Cada núcleo executa essencialmente o mesmo código, mas com condições particulares de contorno. Informações de campos devem ser trocadas nas interfaces dos subdomínios, tal como ilustrado pela Figura 4.

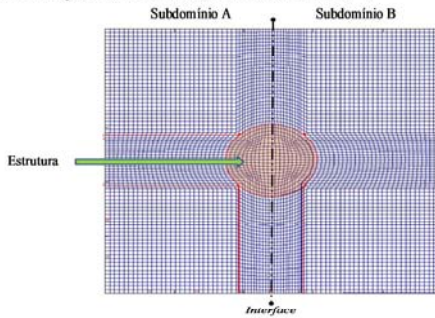


Figura 3. Decomposição do domínio em dois subdomínios e seção transversal (superfície 1-2) de uma grade 3D não-ortogonal.

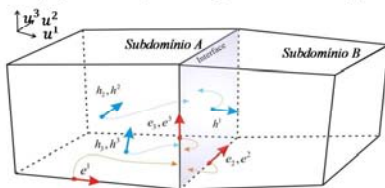


Figura 4. Componentes dos campos, trocadas entre dois subdomínios vizinhos, com interface em uma superfície 2-3.

Além do ilustrado pela Figura 3, algumas estruturas podem ser divididas entre dois ou possivelmente mais subdomínios. Esta questão foi tratada através da implementação de uma sub-rotina, que distribui automaticamente os parâmetros do meio

entre as unidades de processamento de forma que o comportamento eletromagnético do cenário dividido é idêntico ao comportamento de todo domínio numérico tratado por um único núcleo de processamento.

Dessa forma, o *software* pode gerar cenários eletromagnéticos complexos definindo automaticamente os parâmetros para cada CPU, utilizando os dados definidos pelo usuário em uma interface gráfica de usuário (GUI). O cenário é representado graficamente por uma ferramenta de visualização específica, desenvolvida utilizando a biblioteca OpenGL. Essa ferramenta, associada à divisão automática de domínio de sub-rotinas reduz, significativamente, a probabilidade de erros humanos ao construir e simular os cenários. Muitas figuras neste trabalho são resultados diretos do visualizador 3D implementado neste trabalho. Para a geração dos cenários, é utilizada a mesma malha que será fornecida para a realização da simulação numérica. Esta malha deve ser estruturada (Figura 5) para que o processo de simulação e criação do cenário tridimensional seja realizado com sucesso, o que está relacionado à natureza do método LN-FDTD. A malha será desenvolvida de acordo com a natureza do problema a ser simulado e será construída pelo usuário e não pelo *software* deste trabalho, em que o mesmo requer os dados da malha construída como parâmetro de entrada para concepção da estrutura (definição das condições de contorno) e para os cálculos realizados pelo simulador.

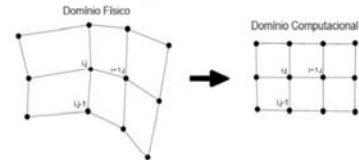


Figura 5. Exemplo de uma malha estruturada na visão do domínio físico e na visão do domínio computacional (como o algoritmo visualiza as malhas).

## II. A INTERFACE GRÁFICA: CENÁRIOS TRIDIMENSIONAIS

A interface mostrada pela Figura 6 foi desenvolvida para facilitar a realização de simulações baseadas no método LN-FDTD em coordenadas gerais e construída utilizando a ferramenta Qt 3 [9] e a linguagem de programação C++. Suas funções foram projetadas com base nas necessidades do usuário da área de engenharia, o que deixa o simulador numérico intuitivo e de fácil uso.

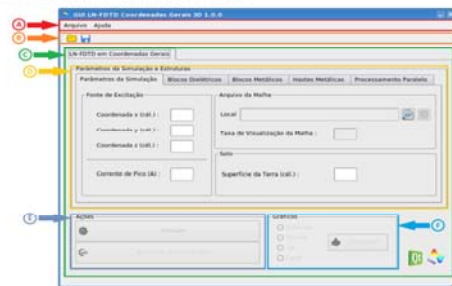


Figura 6. GUI. (A) Menu Principal, (B) Barra de Atalhos, (C) Guia do Método Numérico, (D) Parâmetros da Simulação e Estruturas (Blocos e Funcionalidades da Interface), (E) Ações e (F) Gráficos.

A estrutura da interface foi dividida em seis áreas: *Menu Principal*, *Barra de Atalhos*, *Guia do Método Numérico*, *Parâmetros da Simulação e Estruturas*, *Ações e Gráficos*, como pode ser observado na Figura 6. A organização das áreas permite um acesso simples e direto a todas as funções que contém a interface, colocando a ferramenta acessível aos usuários que possuam dificuldades em lidar com desenvolvimento de *softwares*, evitando o contato do usuário com códigos-fonte.

O *Menu Principal* possui o sub-menu *Arquivo*, que contém as opções *abrir*, *salvar*, *salvar como* e *sair*. Essas opções estão relacionadas aos arquivos de entrada dos parâmetros da simulação e estruturas. O sub-menu *Ajuda* possui as opções *Manual* e *Sobre*. O manual contém todas as informações sobre como utilizar o *software* corretamente e informações sobre os desenvolvedores da aplicação.

A *Barra de Atalho* contém os atalhos para abrir e salvar os arquivos de entrada. A *Guia Método Numérico* foi colocada a fim de que, no futuro, agreguem-se novos métodos numéricos enriquecendo as funcionalidades da interface.

*Parâmetros da Simulação e Estruturas* são responsáveis pela inserção dos *blocos dielétricos*, *blocos metálicos*, *hastes metálicas* e as máquinas do processamento paralelo na simulação. Esses blocos e hastes serão representados no ambiente 3D para que se possa visualizar o que será simulado. *Ações* contém o botão para iniciar a simulação e visualizar a estrutura tridimensional. Finalmente, a seção *Gráficos* tem a finalidade de mostrar, ao final de cada simulação, os gráficos referentes à Corrente, Tensão, TGR (*Total Grounding Resistance*) [11], para sistemas de aterramento e a fonte da simulação (representando uma descarga atmosférica).

#### A. Cenário Gráfico Tridimensional

O cenário gráfico tridimensional foi desenvolvido utilizando as bibliotecas OpenGL [10] e a linguagem de programação C++. Para entender como foi desenvolvido o cenário no ambiente 3D, precisamos conhecer os principais conceitos envolvidos, que são: Visualização e Criação Tridimensional, Sólidos, Câmera e Modelagem de Superfícies Visualização da Malha que serão descritos nos tópicos seguintes.

#### B. Visualização e Criação Tridimensional

Quando se trabalha com três dimensões em OpenGL, o Sistema de Referência do Universo (SRU) passa a possuir três eixos ortogonais entre si ( $x$ ,  $y$ ,  $z$ ) e a origem (0,0, 0,0, 0,0), conforme ilustra a Figura 7.

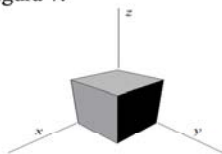


Figura 7. O SRU em três dimensões (OpenGL).

A partir disso, um ponto é definido pelos valores de  $x$ ,  $y$  e  $z$  que correspondem às suas coordenadas nos respectivos eixos [10]. Com isso, compreende-se facilmente como o sistema de

coordenadas funciona em 3D (idêntico ao sistema Cartesiano), e, a partir dele, poderão ser formadas as superfícies do cenário, especificando e posicionando os pontos na malha associada, usando as coordenadas de cada eixo.

#### C. Sólidos

Para este trabalho, um sólido é considerado tudo que tem forma própria (Figura 7). Como não iremos representar o cenário com objetos compostos por materiais líquidos, gasosos, flexíveis ou que não possuem formas totalmente definidas, este conceito se emprega perfeitamente.

#### D. Câmera

Para criar um cenário no ambiente 3D, deve-se, primeiramente, definir os objetos que o compõe. Tais elementos serão incluídos e posicionados no SRU através de suas coordenadas tal como anteriormente comentado [10].

O próximo passo consiste em definir as especificações do observador no ambiente 3D, em que é estabelecido de que ponto no espaço deseja-se que a cena 3D seja exibida. Deste modo, a especificação do observador inclui a sua posição e orientação, ou seja, onde o observador está e para onde ele está olhando no ambiente 3D [10]. A imagem gerada da posição e orientação do observador é estática. Assim, faz-se a relação com uma foto (Figura 8). Porém, modificando-se a posição do observador, cria-se a idéia de movimento e de interatividade. Neste trabalho, isto é controlado via *mouse*.

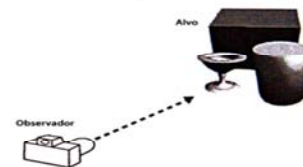


Figura 8. Modelo de câmera utilizado em OpenGL.

#### E. Modelagem de Superfícies Visualização da Malha

A modelagem tem a finalidade de criar objetos a partir de coordenadas  $x$ ,  $y$  e  $z$  no ambiente tridimensional, o que é feito usando-se a malha de discretização que será fornecida como parâmetro de entrada. A partir deste ponto, usando as coordenadas especificadas na interface, os objetos serão criados no cenário 3D (Figura 9).

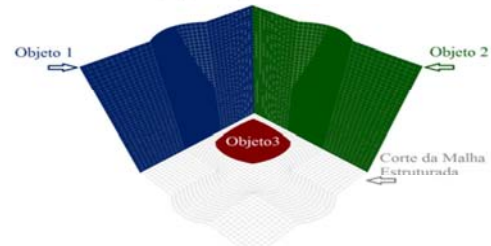


Figura 9. Representação de um cenário 3D (em OpenGL) com seus Objetos e de um corte da malha computacional.

Na Figura 9, têm-se o cenário criado a partir da malha dada como arquivo de entrada, o qual contém as coordenadas e seus pontos (relativas ao sistema cartesiano / SRU). Mediante tais pontos serão construídos os objetos, respeitando-se os limites



do domínio de análise, os quais são especificados no arquivo da malha. Como a malha é estruturada, a visualização da mesma é gerada simplesmente ligando-se os pontos vizinhos do domínio computacional (Figura 5), o que é realizado através de linhas nas direções de  $\vec{a}_1$ ,  $\vec{a}_2$  e  $\vec{a}_3$ , tal como ilustrado na Figura 1. Para representar um objeto como os da Figura 9, o volume de cada célula computacional é preenchido por uma cor, dando a idéia de que o contorno dos objetos acompanha as curvas da malha. Dessa forma, o engenheiro tem a perfeita noção de onde o objeto será posicionado na malha e se a forma real condiz com a representação computacional.

Vale ressaltar que o simulador LN-FDTD está em perfeito sincronismo com o visualizador ilustrado pela Figura 9, de forma que o que é visto é exatamente o que é simulado. Os parâmetros eletromagnéticos  $\epsilon_r$ ,  $\sigma$  e  $\mu_r$  e as coordenadas das estruturas são definidos através de tabelas tal como ilustrado pela Figura 10. Nesta figura, cada linha da tabela corresponde a um objeto, as tabelas (Figura 10, tela B) descrevem as coordenadas e os parâmetros eletromagnéticos e de visualização usados para criar o cenário da Figura 9. A linha 1 da tela B representa o Objeto 1 da Figura 9, a linha 2 o Objeto 2 e a tela C referencia o Objeto 3 da Figura 9. Este cenário não possui hastes metálicas. Portanto, a tabela estará vazia e não será mostrada na Figura 10.

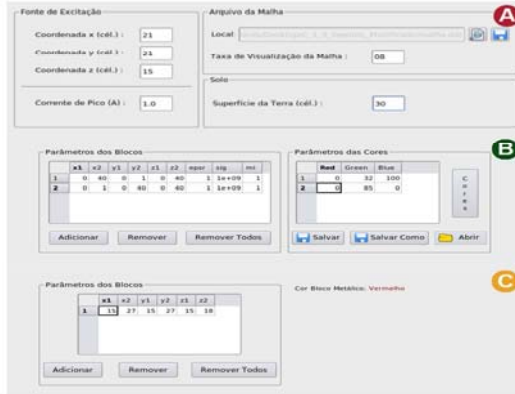


Figura 10. Parâmetros de simulação e criação de objetos (coordenadas em células): (A) Tela de Parâmetros Básicos da Simulação, (B) Blocos Dielétricos e (C) Blocos Metálicos.

A Figura 10 mostra as telas de parâmetros para a realização da simulação. A tela de *Parâmetros da Simulação* (Figura 10, tela A) requer informações sobre a fonte de excitação, a localização do arquivo da malha e da superfície da terra (solo).

Na tela *Blocos Dielétricos* (Figura 10, tela B), são inseridas as informações, tais como:  $\mu$ ,  $\epsilon$ ,  $\sigma$ , dos objetos que compõem o cenário de simulação. Novamente, cada linha corresponde a um objeto inserido e sua respectiva cor deverá ser fornecida. Esses parâmetros serão utilizados tanto pelo simulador como pelo visualizador tridimensional, com exceção dos parâmetros de cores, usados somente para identificação dos objetos no cenário. A tela *Blocos Metálicos* (Figura 10, tela C) segue as mesmas características dos *Blocos*

*Dielétricos*, porém, diferem na cor, cujo padrão é vermelho para todos os blocos definidos e, também diferem na ausência dos parâmetros eletromagnéticos (condutores perfeitos).

## I. RESULTADOS

Para validar os resultados obtidos pelo simulador, dois casos são considerados: A) no primeiro caso, é simulado um eletrodo de aterramento semi-esférico e; B) no segundo caso, faz-se uso de um sistema de aterramento com quatro hastes verticais posicionadas em círculo. Os resultados aqui obtidos, com o uso da interface gráfica, são idênticos aos resultados obtidos em [12] sem o uso de interfaces.

### A. Eletrodo Semi-Esférico

O Eletrodo Semi-Esférico da Figura 11 foi modelado utilizando o visualizador tridimensional desenvolvido.

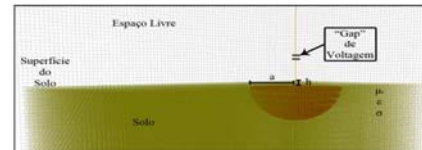


Figura 11. Representação 3D (OpenGL) do Eletrodo Semi-Esférico simulado.

Segundo [12], o valor estacionário para a resistência de terra, para  $h = 0$  (face superior do eletrodo coincidindo com a superfície da terra) é dado por:

$$R = \frac{1}{2\pi a \sigma} \quad (13)$$

A Figura 12 descreve a relação  $V(t)/I(t)$  obtidas para  $\sigma = 2,28 \text{ mS/m}$ ,  $\epsilon_r = 10$  e raio da semi-esfera  $a = 6 \text{ m}$ . Para esta situação, a equação (13) fornece  $R = 11,634 \Omega$ . Na simulação em [12], obteve-se  $R = 12,55 \Omega$ , e no simulador desenvolvido neste trabalho encontrou-se  $R \cong 12,70 \Omega$  (Figura 12) para  $t = 0,5 \mu\text{s}$ .

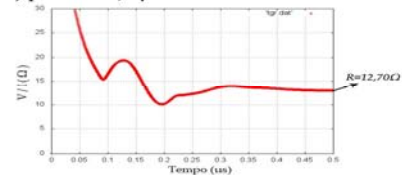


Figura 12. Relação  $V(t)/I(t)$  obtida para o eletrodo semi-esférico.

A diferença entre os valores encontrados pode ser justificada por problemas de arredondamento dos simuladores distintos.

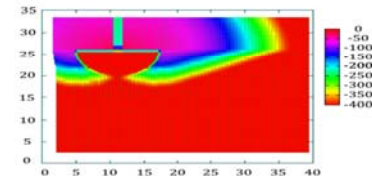


Figura 13. Componente  $e_3$  do campo  $\vec{E}$  (dB) no início do processo de propagação.

A distribuição espacial da componente  $e_3$  do campo  $\vec{E}$  (Figura 13), em uma superfície vertical que passa pelo centro do eletrodo semi-esférico, é gerada ao final da simulação.

### A. Sistema de Aterramento Com Quatro Hastes Verticais Posicionadas em Circulo

A simulação consiste em quatro eletrodos verticais, de quatro metros de comprimento, conectados por um condutor circular de raio também de quatro metros, posicionado 0,5 metro abaixo da superfície do solo.

A Figura 14 mostra a concepção da estrutura no simulador 3D desenvolvido. Esta estrutura foi construída utilizando somente hastes metálicas. A conexão elétrica entre as hastes é feita através de condutores elétricos que seguem a malha inicialmente estabelecida. Neste caso, a malha da Figura 3 foi utilizada.



Figura 14. Representação dos Eletrodos em Disposição Circular 3D (em OpenGL).

De acordo com [12], o valor estacionário da resistência de terra para a estrutura acima pode ser calculada por

$$R = \frac{1/(4L) + 1/(8\pi r)}{\sigma}, \quad (14)$$

em que  $L$  é o comprimento das hastes e  $r$  é o raio do círculo. Para este caso, (14) fornece  $R = 36,224 \Omega$  ( $\sigma = 0,002 \text{ S/m}$ ) e o simulador deste trabalho obteve  $R \cong 37,90 \Omega$ . A Figura 15 mostra os resultados obtidos com os métodos FDTD, considerando-se discretizações diferentes, e LN-FDTD.

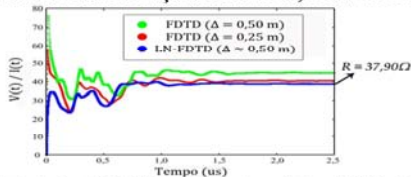


Figura 15. Relações  $V(t)/I(t)$  obtidas pelos métodos FDTD e LN-FDTD para os eletrodos dispostos circularmente.

Na Figura 15, observa-se que na medida em que se aumenta a resolução do método FDTD, os resultados tendem para o resultado calculado pelo simulador desenvolvido. Observa-se que a simulação em coordenadas gerais concorda com (14) em termos de resposta estacionária, e que as soluções obtidas pelo método FDTD geram oscilações adicionais devido a aproximações por *staircase*. Além disso, estas aproximações aumentam a relação  $V(t)/I(t)$  para a estrutura, já que este tipo de geometria (conexão circular, Fig. 14) causa mudanças bruscas no percurso da corrente, dificultando a sua circulação.

## II. CONCLUSÕES

O principal objetivo deste trabalho foi desenvolver um ambiente computacional de fácil interação com o usuário, em que o mesmo possa realizar simulações utilizando o método numérico LN-FDTD, com processamento paralelo. O uso do *software* não requer que o usuário tenha algum tipo de especialização sobre a técnica numérica. Isto possibilita que

estudantes iniciantes da área de engenharia possam utilizá-lo sem dificuldades.

Os resultados obtidos pelo simulador desenvolvido foram fieis aos resultados disponíveis na literatura, validando a presente implementação computacional. A interface gráfica, junto com o visualizador 3D, apresenta vantagens importantes que vão da melhoria da eficiência, pois simplifica o uso do *software* desenvolvido, até a redução dos índices de erro humano no momento da concepção dos cenários. Os resultados obtidos mostram que a interface gráfica está em perfeito sincronismo com o simulador, garantindo que o que é mostrado no ambiente virtual é o que será simulado. Com isso, o presente trabalho se apresenta como uma ferramenta capaz de realizar outras simulações envolvendo o método LN-FDTD, tais como em antenas e em compatibilidade eletromagnética, por exemplo, sem que seja necessária qualquer modificação no código-fonte do programa simulador.

Para trabalhos futuros, serão desenvolvidas simulações experimentais para realizar a validação completa da ferramenta desenvolvida.

## REFERÊNCIAS

- [1] M. N. O. Sadiku, "Numerical Techniques in Electromagnetics". 2nd ed., R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev, New York: CRC Press, pp. 590 - 649, 2001.
- [2] D. R. de Melo, M. N. Kawakatsu, V. Dmitriev, K. Q. da Costa, "Aplicação do Método dos Momentos na Análise de Antenas," XXXI CNMAC. Belém-Pará-Brasil, 2008.
- [3] A. Taflove and S. C. Hagness, "Computational Electrodynamics, The Finite-Difference Time-Domain Method", 3rd ed. Artech House Inc, 2005.
- [4] C. E. R. Mercedes, V. F. R. Esquerre, A.M.F. Frasson, and H.E.H.Figueroa, "Novel FEM Approach for the Analysis of Cylindrically Symmetric Photonic Devices", Journal of Lightwave Technology, vol. 27, n. 21, pp. 4717-4721, November 2009.
- [5] R. M. S. de Oliveira and C. L. S. S. Sobrinho, "UPML Formulation for Truncating Conductive Media in Curvilinear Coordinates", Numerical Algorithms, vol. 46, pp. 295-319, Springer Netherlands, December 2007.
- [6] <http://www.comsol.com/products/multiphysics/research/tutorials/> accessed on April 15, 2010.
- [7] R. M. S. de Oliveira e C. L. S. S. Sobrinho, "Simulações Através do Método FDTD do Espalhamento Eletromagnético em uma Subestação de Potência Devido a Descargas Atmosféricas", MOMAG, 2008.
- [8] K. S. Sampath, R. G. Rojas "Application of the Helicopter Antenna Radiation Prediction Code (HARP) to Modeling Fixed Wing Aircraft", The Ohio State University ElectroScience Laboratory, Columbus, Ohio, 1993.
- [9] <http://qt.nokia.com/> accessed on April 15, 2010.
- [10] M. Cohen, I. H. Manssour. OpenGL: uma abordagem prática e objetiva. São Paulo: Novatec, 2006.
- [11] K. Tanabe, "Novel method for analyzing the transient behavior of grounding systems based on the finite-difference time-domain method", vol. 3, pp.1128-132 28, in Power Engineering Society Winter Meeting, IEEE, Columbus (OH USA), 2001.
- [12] R. M. S. de Oliveira, "Nova Metodologia para Análise e Síntese de Sistemas de Aterramento Complexos Utilizando o Método LN-FDTD, Computação Paralela Automática e Redes Neurais Artificiais", Tese (Doutorado em Engenharia Elétrica), ITEC - Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2008, unpublished.

# APÊNDICE D

## Redução do Tempo de Transferência de Dados Para o Método LN-FDTD em um *Cluster* Beowulf Utilizando *Sockets*

Rodrigo Lisbôa Pereira, Adolfo F. de O. Colares, Rodrigo M. S. de Oliveira, Carlos Leonidas de S.S. Sobrinho.  
Programa de Pós-Graduação em Engenharia Elétrica  
Universidade Federal do Pará - UFPA  
Caixa Postal 8619 – 66.075-900 Belém – Pará – Brasil  
{rlp, adolfo, rmso, leonidas}@ufpa.br

**Resumo** – Neste artigo, é investigado o uso de *sockets*, através do desenvolvimento de uma biblioteca para a troca de mensagens em sistemas distribuídos, objetivando a redução do tempo de execução de algoritmos paralelos, simulados em um *cluster* Beowulf. Com este objetivo, é feito um estudo comparativo para avaliar o desempenho da especificação MPI e da API *socket* para a troca de mensagens entre diferentes máquinas do *cluster*, que são interconectados via rede Ethernet. Para o estudo, uma versão paralela do algoritmo, de implementação do método LN-FDTD, é usada como base para a comparação.

**Palavras-chave** – *Socket*; processamento paralelo; sistemas distribuídos; MPI; avaliação de desempenho; Método LN-FDTD; Método LN-UPML; *cluster*.

### I. INTRODUÇÃO

Problemas práticos de engenharia, relacionados ao eletromagnetismo, são comumente resolvidos através de métodos numéricos [1]. Dentre os diversos métodos numéricos existentes, que podem ser usados na solução das equações de Maxwell, destaca-se o método LN-FDTD (*Local Nonorthogonal Finite-Difference Time-Domain*) [2]. Este método, que será usado neste trabalho, representa uma adaptação do método FDTD (*Finite-Difference Time-Domain*), criado em 1966 por Yee [3], para a solução das equações de Maxwell em um sistema de coordenadas gerais. Desta forma, através do método LN-FDTD é possível solucionar problemas envolvendo estruturas com diversas geometrias, o que representa uma vantagem deste método [4].

Com o desenvolvimento de computadores cada vez mais velozes e de *softwares* eficientes, a solução numérica torna-se a cada dia mais interessante em relação às outras opções de solução, como por exemplo, a experimental e a analítica. Isto se deve aos custos mais baixos e a grande flexibilidade, inerentes a este tipo de solução. Assim, os COWs (*Cluster Of Workstations*) ou *clusters* de computadores passaram a ter grande importância para os pesquisadores, que trabalham com problemas reais de engenharia, pois representam ferramentas de relativamente fácil manutenção, baixo custo e com grande poder computacional. Os COWs podem ser considerados como uma classe econômica de supercomputadores, dada a capacidade de processamento e de memória RAM (*Random Access Memory*).

Estudos na área de processamento paralelo são cada vez

mais importantes, uma vez que os algoritmos seqüenciais complexos, via de regras, envolvem tarefas que podem ser resolvidas simultaneamente. Isto é feito com a finalidade de obter maior desempenho e economia de tempo de execução dos programas, de tal forma que, os resultados gerados pelos algoritmos, seqüencial e paralelo, sejam idênticos. Além disso, o uso de processamento paralelo viabiliza a geração de resultados que não poderiam ser gerados seqüencialmente, em um único PC (*Personal Computer*), face ao grande tempo de processamento e à necessidade de memória RAM requeridos por aplicações específicas [5].

Em [6], é feito um estudo comparativo entre os desempenhos de diversas linguagens e compiladores, os quais são aplicados na solução paralela do método FDTD. O paralelismo em [6] utiliza em todos os casos a especificação MPI (*Message Passing Interface*) e, a linguagem C se mostrou a mais eficiente dentre as aferidas. Em [7], descreve-se um trabalho que utiliza a computação paralela em *cluster*, evidenciando o ganho de desempenho obtido.

A MPI se apresenta como uma especificação de transferência de mensagens, com o objetivo maior de simplificar o trabalho dos programadores. Para isto, traz uma quantidade considerável de ferramentas que permitem o controle da transferência de dados entre as máquinas. Por exemplo, sistema para fácil identificação dos processos, diversos procedimentos para comunicação coletiva, vários tipos de dados para troca de mensagens, entre muitos outros recursos [8]. Nesta especificação, são suportados diversos paradigmas de paralelismo.

O *socket* também é utilizado para a troca de mensagens entre máquinas. Em [9], encontra-se um trabalho que aborda, em linhas gerais, a implementação e avaliação do *socket* de Berkeley em uma arquitetura paralela, o *cluster* Maestro2. Os resultados experimentais observados em [9], mostram que a MMP (*Massive Parallel Processing*) *Sockets* oferece uma latência mínima de 25  $\mu$ s (mili-segundos) e um *throughput* máximo de 1250 Mbps (*Megabit* por segundo), correspondendo a um aumento relativo de 80% para a latência e uma diminuição de cerca de 30% para o *throughput*, em relação a uma comunicação baseada apenas em MMP.

Neste contexto, este trabalho demonstra a eficiência do uso do *socket*, em relação ao tempo total de processamento. Com



este objetivo, desenvolveu-se uma biblioteca baseada em *sockets* para realizar a troca de mensagens em sistemas distribuídos. Tal biblioteca inclui apenas o que é necessário para se executar em paralelo uma simulação LN-FDTD, de tal forma que o objetivo principal do trabalho é verificar e contabilizar ganhos em relação à MPI.

Este artigo está organizado da seguinte maneira: em II, descreve-se a teoria referente ao método numérico LN-FDTD, processamento paralelo e ambiente para a troca de mensagens em rede; em III, apresenta-se a biblioteca baseada em *socket* desenvolvida neste trabalho; em IV, demonstra-se e discute-se o estudo de caso e os resultados obtidos em termos de tempo total de simulação; finalmente, as considerações finais deste trabalho são feitas em V.

## I. EMBASAMENTO TEÓRICO

### A. O sistema de Coordenadas Gerais

Seja um ponto  $P$  no espaço curvilíneo tridimensional por onde passam as funções  $u^1$ ,  $u^2$  e  $u^3$ , denominadas coordenadas curvilíneas, em uma dada região de espaço (Fig. 1). Considere também um vetor  $\vec{r}$  que vai da origem  $O$  a  $P$  e três vetores  $\vec{a}_1$ ,  $\vec{a}_2$  e  $\vec{a}_3$  tangentes às curvas  $u^1$ ,  $u^2$  e  $u^3$ , respectivamente, em  $P$ .

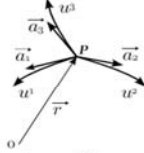


Fig. 1. Sistema de coordenadas curvilíneas e os vetores unitários em  $P$ .

Nestas condições, o diferencial de  $\vec{r}$ , denotado por  $d\vec{r}$ , é dado por

$$d\vec{r} = \vec{a}_1 du^1 + \vec{a}_2 du^2 + \vec{a}_3 du^3, \quad (1)$$

na qual os vetores  $\vec{a}_l = \partial\vec{r}/\partial u^l$  (com  $l = 1, 2$  ou  $3$ ) são chamados de vetores unitários, que formam uma base unitária em  $P$ . Os vetores unitários podem, obviamente, ser escritos em termos do sistema Cartesiano de coordenadas.

A partir dos vetores  $\vec{a}_l$ , pode-se definir um conjunto auxiliar de três vetores  $\vec{a}^l$  (chamados de vetores recíprocos), de tal forma que cada novo vetor é ortogonal a dois vetores unitários simultaneamente. Matematicamente, tem-se

$$\vec{a}^1 = \frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}}, \quad \vec{a}^2 = \frac{\vec{a}_3 \times \vec{a}_1}{\sqrt{g}}, \quad \vec{a}^3 = \frac{\vec{a}_1 \times \vec{a}_2}{\sqrt{g}}, \quad (2)$$

na qual  $g$  representa o determinante do tensor métrico [4] dado por [4]

$$[g] = \begin{pmatrix} \vec{a}_1 \cdot \vec{a}_1 & \vec{a}_1 \cdot \vec{a}_2 & \vec{a}_1 \cdot \vec{a}_3 \\ \vec{a}_2 \cdot \vec{a}_1 & \vec{a}_2 \cdot \vec{a}_2 & \vec{a}_2 \cdot \vec{a}_3 \\ \vec{a}_3 \cdot \vec{a}_1 & \vec{a}_3 \cdot \vec{a}_2 & \vec{a}_3 \cdot \vec{a}_3 \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix}. \quad (3)$$

Desenvolvendo o determinante de  $[g]$ , observa-se que  $\sqrt{g} = \vec{a}_1 \cdot (\vec{a}_2 \times \vec{a}_3)$ , que é o volume do hexaedro formado pelos vetores  $\vec{a}_1$ ,  $\vec{a}_2$  e  $\vec{a}_3$ .

Usando a definição matemática (2), pode ser facilmente observado que  $\vec{a}^l \cdot \vec{a}_m = \delta_{lm}$ , na qual  $\delta_{lm}$  é o Delta de Kronecker. Além disso, uma definição métrica similar à

introduzida por (3) pode ser feita para os vetores recíprocos da seguinte maneira:  $g^{lm} = \vec{a}^l \cdot \vec{a}^m$ .

Assim, com esses conceitos em mente, é possível expandir um vetor  $\vec{V}$  usando vetores unitários e recíprocos, a partir de (1)-(3), da seguinte forma

$$\vec{V} = \sum_{l=1}^3 v^l \vec{a}_l = \sum_{l=1}^3 v_l \vec{a}^l, \quad (4)$$

na qual  $v^l$  e  $v_l$  são chamadas de componentes contravariantes e covariantes do vetor  $\vec{V}$ , respectivamente. Para calcular a  $l$ 'ésima componente contravariante do vetor  $\vec{V}$ , o produto escalar  $\vec{V} \cdot \vec{a}^l$  deve ser calculado. Dessa forma, utilizando a relação de Kronecker, verifica-se que

$$\vec{V} \cdot \vec{a}^l = (\sum_{i=1}^3 v^i \vec{a}_i) \cdot \vec{a}^l = v^l. \quad (5)$$

De forma similar, a  $l$ 'ésima componente covariante de  $\vec{V}$  é obtida por

$$\vec{V} \cdot \vec{a}_l = (\sum_{i=1}^3 v^i \vec{a}_i) \cdot \vec{a}_l = v_l. \quad (6)$$

Pode-se ainda obter uma relação para calcular as componentes covariantes de  $\vec{V}$  a partir de suas componentes contravariantes (e vice-versa). Se for calculado o produto escalar  $\vec{V} \cdot \vec{a}_m = (\sum_{l=1}^3 v^l \vec{a}_l) \cdot \vec{a}_m$ , obtém-se  $v_m = \sum_{l=1}^3 g_{lm} v^l$ . Seguindo procedimento similar, chega-se a  $v^m = \sum_{l=1}^3 g^{lm} v_l$ .

### B. O Método LN-FDTD

As equações de Maxwell, em sua forma diferencial no domínio do tempo, são dadas por :

$$\frac{\partial \mu \vec{H}}{\partial t} = -\vec{\nabla} \times \vec{E} \quad (7)$$

e

$$\frac{\partial \epsilon \vec{E}}{\partial t} = \vec{\nabla} \times \vec{H} - \vec{J}, \quad (8)$$

nas quais  $\vec{E}$  é o vetor intensidade de campo elétrico (volt por metro),  $\vec{H}$  é o vetor intensidade de campo magnético (ampère por metro),  $\epsilon$  e  $\mu$  são, respectivamente, permissividade elétrica (farad por metro) e permeabilidade magnética (henry por metro). O termo  $\vec{J} = \sigma \vec{E}$  é o vetor densidade de corrente elétrica de condução (ampère por metro quadrado), na qual  $\sigma$  é a condutividade elétrica do meio em consideração, dada em siemens por metro.

A discretização do espaço de análise é feita de forma similar ao que é feito no algoritmo de Yee original [3]. Porém, as células de Yee possuem agora arestas não ortogonais entre si, tal como ilustrado pela Fig. 2. As células primárias (Fig. 2b), onde são posicionadas as componentes covariantes do campo elétrico, são definidas pelos vetores unitários (Fig. 2a). Os vetores recíprocos definem células secundárias, onde as componentes contravariantes do campo magnético são posicionadas.

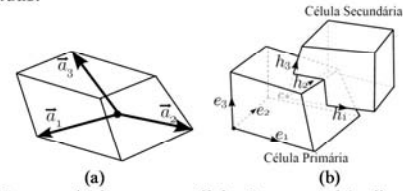


Fig. 2. (a) Vetores unitários em uma célula não ortogonal (malha estruturada) e (b) componentes covariantes em células não ortogonais de Yee.



Para obterem-se as componentes contravariantes do campo  $\vec{H}$ , representadas por  $h^l$  ( $l = 1,2,3$ ), a equação vetorial (7) pode ser decomposta em vetores paralelos aos vetores  $\vec{a}^l$  usando a operação produto escalar. Dessa forma, sem perda de generalidade, para  $l = 1$  tem-se

$$\left(-\mu \frac{\partial \vec{h}}{\partial t}\right) \cdot \vec{a}^1 = (\vec{\nabla} \times \vec{E}) \cdot \vec{a}^1. \quad (9)$$

Utilizando (2) e (5), obtém-se

$$\left(-\mu \frac{\partial h^1}{\partial t}\right) = (\vec{\nabla} \times \vec{E}) \cdot \left(\frac{\vec{a}_2 \times \vec{a}_3}{\sqrt{g}}\right). \quad (10)$$

Aplicando-se a identidade vetorial  $(\vec{A} \times \vec{B}) \cdot (\vec{C} \times \vec{D}) = (\vec{A} \cdot \vec{C})(\vec{B} \cdot \vec{D}) - (\vec{A} \cdot \vec{D})(\vec{B} \cdot \vec{C})$  no lado direito da Equação acima, chega-se a

$$\left(-\mu \frac{\partial h^1}{\partial t}\right) = \frac{(\vec{\nabla} \cdot \vec{a}_2)(\vec{E} \cdot \vec{a}_3) - (\vec{\nabla} \cdot \vec{a}_3)(\vec{E} \cdot \vec{a}_2)}{\sqrt{g}}. \quad (11)$$

De (1), nota-se que  $\vec{\nabla} = \sum_{l=1}^3 (\partial / \partial u^l) \vec{a}^l$ . Assim, empregando a propriedade de Kronecker ainda para o lado direito, verifica-se que a primeira equação, para a primeira componente contravariante de  $\vec{H}$ , é dada por

$$\frac{\partial h^1}{\partial t} = -\frac{\left(\frac{\partial e_3}{\partial u^2} - \frac{\partial e_2}{\partial u^3}\right)}{\mu \sqrt{g}}, \quad (12)$$

onde  $e_i$  ( $i = 1,2,3$ ) são as componentes covariantes de  $\vec{E}$ . A equação acima pode ser discretizada por diferenças finitas, obtendo-se

$$-\frac{\Delta t}{\mu \sqrt{g}} \left[ \frac{e_{3(i,j,k+1)}^{(n)} - e_{3(i,j,k)}^{(n)}}{\Delta u^2} - \frac{e_{2(i,j,k+1)}^{(n)} - e_{2(i,j,k)}^{(n)}}{\Delta u^3} \right]. \quad (13)$$

As demais componentes dos campos  $\vec{H}$  e  $\vec{E}$  são obtidas e discretizadas do mesmo modo. Vale ressaltar [4] que  $\Delta u^l = 1$ .

#### A. UPML (Uniaxial Perfectly Matched Layers) para meios condutivos – Truncamento do método LN-FDTD

As equações de Maxwell (7) e (8), no domínio da frequência para um meio condutivo, anisotrópico uniaxial, são dadas por

$$\nabla \times \vec{E}^* = -j\omega \vec{S} \vec{H}^* ; \quad \nabla \times \vec{H}^* = (j\omega \epsilon + \sigma) \vec{S} \vec{E}^*, \quad (14)$$

na qual  $\omega$  define a frequência angular da onda eletromagnética,  $\vec{E}^*$  e  $\vec{H}^*$  são transformadas de Fourier dos campos elétrico e magnético, respectivamente,  $\mu = \mu_0 \mu_r$ ,  $\epsilon = \epsilon_0 \epsilon_r$ ,  $\vec{S}$  é a matriz tensorial que gera a atenuação, ao longo dos eixos coordenados, para as camadas das regiões absorventes e  $\sigma$  representa a condutividade do meio. A matriz  $\vec{S}$  possui o formato:

$$\vec{S} = \begin{bmatrix} \frac{S_2 \cdot S_3}{S_1} & 0 & 0 \\ 0 & \frac{S_1 \cdot S_3}{S_2} & 0 \\ 0 & 0 & \frac{S_1 \cdot S_2}{S_3} \end{bmatrix}, \quad (15)$$

onde  $S_\alpha$  ( $\alpha = 1,2,3$ ) são os parâmetros responsáveis pela atenuação na UPML ao longo dos eixos de coordenadas gerais referidos. Tais parâmetros têm a forma geral dada por (16).

$$S_\alpha = K_\alpha + \frac{\sigma_\alpha}{j\omega \epsilon_0}. \quad (16)$$

Tal como descrito em [4], é possível transformar as equações (14) para o domínio do tempo, considerando-se (15) e (16), através da introdução de variáveis auxiliares, as quais eliminam a necessidade do uso de convoluções. O resultado é um eficiente algoritmo explícito.

#### B. Processamento Paralelo para o método LN-FDTD

O uso da computação paralela para resolver problemas eletromagnéticos pelo método LN-FDTD é baseado na divisão do domínio de análise em subdomínios, conforme é mostrado na Fig. 3 (malha de discretização para a secção transversal de um cabo de alta tensão). Cada subdomínio consiste em uma parte de todo o volume numérico a ser tratado por um único núcleo de processamento. Um núcleo realiza, basicamente, a execução do mesmo código LN-FDTD, todavia com condições de contorno particulares.

No processamento paralelo, existem alguns paradigmas de programação, dentre os quais podemos citar a passagem de mensagens (*message passing*) entre as máquinas presentes em uma arquitetura de *clusters*, a memória compartilhada distribuída (*Distributed Shared Memory - DSM*), *threads* e o paradigma da POO (Programação Orientada a Objetos). Os *clusters* consistem em agrupar computadores de maneira a, de uma forma geral, aumentar o desempenho das aplicações que serão executadas [10].

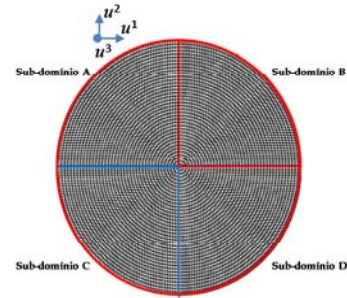


Fig. 3. Domínio decomposto em quatro subdomínios e divisão transversal de uma malha 3D não ortogonal.

A Fig. 4 exibe a passagem de informações entre duas células adjacentes e não ortogonais pertencentes aos subdomínios A e B da Fig. 3. A interface entre tais subdomínios é uma superfície  $u^2 - u^3$  que define as componentes de campo que devem ser trocadas para garantir a continuidade espacial do algoritmo (setas tracejadas). Além disso, para completar este modelo, faz-se necessária a passagem das informações métricas  $g$  na interface entre os subdomínios.

A Fig. 5 descreve o algoritmo paralelo do método FDTD. Tal algoritmo é similar àquele do LN-FDTD, com exceção da leitura da malha e do cálculo das métricas, o que é feito antes do *loop* do tempo. Nesta figura, é importante observar que a troca de mensagens é feita para todos os instantes de tempo. Desta forma, para este algoritmo, fica evidente que um dado ganho no tempo de execução da rotina, que realiza a troca de informações entre máquinas na rede, é multiplicado pelo número de vezes que a troca é realizada ( $n_{\max}$ ).

### A. MPI (Message Passing Interface)

A MPI consiste em um padrão de interface para programação de aplicações, desenvolvida por um grupo especialista representante da indústria de HPC (*High Performance Computing*) e de centros de pesquisa, entre os anos de 1993 e 1994 [11]. A MPI fornece tipos especiais que facilitam bastante, para o programador, a troca de dados entre processadores, utilizando o conceito de comunicador, que permite organizar os processadores por grupos [12].

O modelo empregado neste trabalho pela MPI é o *master-slave*, porém a MPI pode ser usada com muitos outros paradigmas de programação [11]. Este modelo funciona da seguinte maneira: o nó *master* controla o *cluster* inteiro e distribui os processos aos nós *slaves*. Dentre as rotinas presentes na MPI, existem as que são executadas tanto pelo *slave* quanto pelo *master* (*MPI\_Init*, *MPI\_Comm\_rank*, *MPI\_Comm\_size*, *MPI\_Send*, *MPI\_Recv*, *MPI\_Finalize*) [11], de acordo com a necessidade da aplicação.

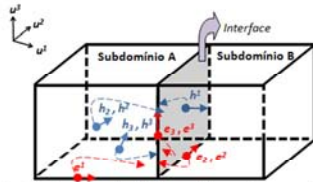


Fig. 4. Representação da troca de informações entre os subdomínios A e B.

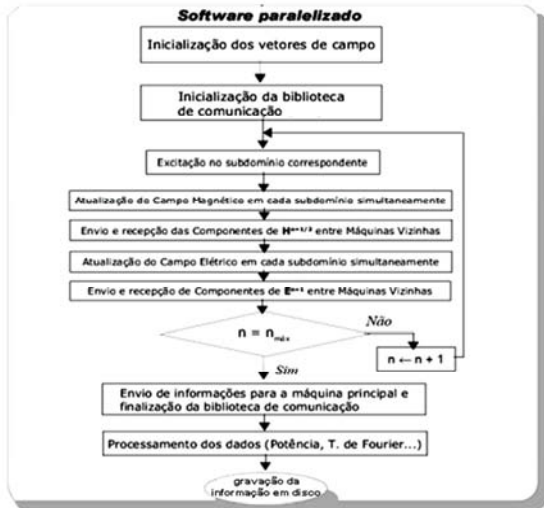


Fig. 5. Lógica do algoritmo paralelizado baseado no método FDTD.

### B. BSD (Berkeley Sockets)

O Soquete de Berkeley (BSD), popularmente conhecido como *socket*, consiste no conjunto de primitivas de transporte aplicadas ao UNIX de Berkeley de acordo com os protocolos TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*) [13]. O *socket* é uma API (*Application Programming Interface*), a qual pode ser utilizada também em programação paralela, tal como feito neste trabalho.

Um modelo utilizado em *socket* é o cliente-servidor [13]. O servidor faz uma abertura passiva, esperando conexões de um cliente, e o cliente realiza a abertura ativa da conexão. Dentre as primitivas utilizadas no *socket* executadas pelo servidor, podemos citar *SOCKET*, *BIND*, *LISTEN*, *ACCEPT*, *SEND*, *CLOSE* e as executadas pelo cliente (*SOCKET*, *CONNECT*, *RECEIVE*, *CLOSE*) [14].

Vale ressaltar que o *socket* não possui qualquer facilidade para uso em *clusters*, tais como as ferramentas disponibilizadas pela MPI (nem mesmo a identificação dos processadores).

## II. A BIBLIOTECA MYSOCKET.H

A biblioteca desenvolvida contém um conjunto de macros que utilizam transferência de dados bidirecional. Para isso, foram desenvolvidos macros [15] para envio e recepção de variáveis do tipo *float*, além de se criar um sistema de identificação de máquinas similar ao da MPI.

As macros do servidor são as macros definidas inicialmente na biblioteca. São elas *SERV\_VARS*: variáveis do servidor; *SERV\_CONNECT(PORTA)* inicializam o servidor e *SERV\_END* finalizam o servidor. Tais macros estão definidas na Tabela I. Na Tabela II são apresentadas as macros criadas para o cliente: *CLT\_VARS*; *CLT\_CONNECT(IP,PORTA)* e *CLT\_END*. Ressalta-se que as funcionalidades são similares às do servidor.

Para proporcionar um melhor entendimento da implementação das macros, o código encontra-se endentado. É importante frisar que para o correto funcionamento da biblioteca *mysock*, as macros devem ser definidas em uma única linha de código.

Por fim, são apresentadas na Tabela III as macros para envio e recepção (*SEND* e *RECV*). Em ambas, têm-se as primitivas que realizam o envio e a recepção de uma mensagem do tipo (*float*). Nelas, a variável *A* é tratada de forma binária para acelerar o envio da mensagem pela rede (o *socket* espera receber informações em forma de texto/*char*).

Vale ressaltar que as macros desenvolvidas atuam similarmente às rotinas da MPI, facilitando o trabalho do programador. A execução dos processos é feita utilizando um *script* concebido em *bash*, emulando o comando *mpirun* [11].

## III. ESTUDO DE CASO E RESULTADOS OBTIDOS

O ambiente em que foi testada a biblioteca desenvolvida consiste em um *cluster* do tipo Beowulf [16]. O *cluster* denominado Amazônia é formado por quatro computadores, sendo um servidor *master* e três escravos (*slaves*), trabalhando com o sistema operacional GNU/Linux Slackware (versão Slam64 12.2). Todas as quatro máquinas do *cluster* possuem dois processadores de dois núcleos Intel Xeon 64 bits *QuadCore* com 2,0 GHz (*gigahertz*) de *clock*, com memória de 4 GB (*Gigabytes*) DDR2 com *clock* de 800 MHz (*megahertz*) por máquina, com placa de rede 10/100/1000 Mbps e são interligadas por um *switch ethernet* de 10/100/1000 Mbps. As máquinas possuem instalada a



TABELA I. MACROS PARA INICIALIZAÇÃO DO SERVIDOR.

Macro	Composição (primitivas socket)
<code>#define SERV_VARS</code>	<code>char SERV_BUFF[FLOAT_SIZE]; int SERV_i; char* SERV_TEMP; int SOCKFD_S, SOCKFD_STD; struct sockaddr_in SERV_MY_ADDR; struct sockaddr_in SERV_THEIR_ADDR; socklen_t SERV_SIN_SIZE; struct sigaction SERV_SA; int SERV_YES = 1;</code>
	<code>if( (SOCKFD_S = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("socket"); exit(errno); } if(setsockopt(SOCKFD_S, SOL_SOCKET, SO_REUSEADDR, &amp;SERV_YES, sizeof(int)) == -1) { perror("setsockopt"); return errno; }</code>
<code>#define SERV_CONNECT (PORTA)</code>	<code>SERV_MY_ADDR.sin_family = AF_INET; SERV_MY_ADDR.sin_port = htons(PORTA); SERV_MY_ADDR.sin_addr.s_addr = INADDR_ANY; memset(SERV_MY_ADDR.sin_zero, '\0', sizeof(SERV_MY_ADDR.sin_zero)); if(bind(SOCKFD_S, (struct sockaddr*)&amp;SERV_MY_ADDR, sizeof(SERV_MY_ADDR)) == -1) { perror("bind"); return errno; } if(listen(SOCKFD_S, BACKLOG) == -1) { perror("listen"); return errno; }</code>
	<code>SERV_SA.sa_handler = sigchld_handler; sigemptyset(&amp;SERV_SA.sa_mask); SERV_SA.sa_flags = SA_RESTART; if(sigaction(SIGCHLD, &amp;SERV_SA, NULL) == -1) { perror("sigaction"); return errno; }</code>
	<code>SERV_SIN_SIZE = sizeof(SERV_THEIR_ADDR); if(SOCKFD_STD = accept(SOCKFD_S, (struct sockaddr*)&amp;SERV_THEIR_ADDR, &amp;SERV_SIN_SIZE) == -1) { perror("accept"); }</code>
<code>#define SERV_END</code>	<code>shutdown(SOCKFD_STD, SHUT_RDWR); close(SOCKFD_STD);</code>

TABELA II. MACROS PARA INICIALIZAÇÃO DO CLIENTE.

Macro	Composição (primitivas socket)
<code>#define CLT_VARS</code>	<code>int CLT_i; char* CLT_TEMP; int CLT_N; struct sockaddr_in CLT_ENDV; char CLT_BUFF[FLOAT_SIZE]; int CLT_YES = 1;</code>
	<code>if( (SOCKFD_STD = socket(AF_INET, SOCK_STREAM, 0)) &lt; 0) { perror("socket"); return errno; } if(setsockopt(SOCKFD_STD, SOL_SOCKET, SO_REUSEADDR, &amp;CLT_YES, sizeof(int)) == -1) { perror("setsockopt"); return errno; }</code>
<code>#define CLT_CONNECT (IP, PORTA)</code>	<code>CLT_ENDV.sin_family = AF_INET; CLT_ENDV.sin_port = htons(PORTA); if( inet_pton(AF_INET, IP, &amp;CLT_ENDV.sin_addr) &lt; 0) { perror("inet_pton"); return errno; }</code>
	<code>memset(CL_T_ENDV.sin_zero, '\0', 8); while(connect(SOCKFD_STD, (struct sockaddr*)&amp;CLT_ENDV, sizeof(struct sockaddr)) &lt; 0) { usleep(1000); }</code>
<code>#define CLT_END</code>	<code>shutdown(SOCKFD_STD, SHUT_RDWR); close(SOCKFD_STD);</code>

TABELA III. MACROS USADAS PARA ENVIO E RECEPÇÃO DE DADOS.

Macro	Composição (primitivas socket)
<code>#define SEND(A)</code>	<code>SERV_TEMP = (char*)&amp;A; CLT_N = 1; for(SERV_i = 0; SERV_i &lt; sizeof(float); SERV_i++) SERV_BUFF[SERV_i] = SERV_TEMP[SERV_i]; while(CL_T_N == -1) { CLT_N = send(SOCKFD_STD, SERV_BUFF, FLOAT_SIZE, 0); } CLT_TEMP = (char*)&amp;A; CLT_N = 1;</code>
<code>#define RECVA</code>	<code>while(CL_T_N != FLOAT_SIZE) { CLT_N = read(SOCKFD_STD, CLT_BUFF, FLOAT_SIZE); for (CLT_i = 0; CLT_i &lt; FLOAT_SIZE; CLT_i++) CLT_TEMP[CLT_i] = CLT_BUFF[CLT_i]; }</code>

### A. Estrutura do Cabo Coaxial

A Fig. 6 ilustra um cabo de alta tensão com um núcleo metálico usado como referência para a aplicação relativa à paralelização da técnica numérica LN-FDTD. O condutor interno é envolvido pelas camadas indicadas pela Fig. 6.

O modelo numérico do cabo possui suas extremidades truncadas por regiões absorventes do tipo UPML para meios condutivos [4]. Este procedimento é usado para absorver ondas eletromagnéticas que chegam a essas regiões, impedindo reflexões não físicas nos limites do domínio de análise. Esta técnica permite solucionar numericamente o problema para um cabo de comprimento infinito [17]. A malha da Fig. 3 foi usada para discretizar o cabo de seção reta constante.



Fig. 6. Cabo coaxial de alta tensão.

### B. Troca de Mensagens: Cabo Coaxial de Alta Tensão

A troca de mensagens é feita tanto através da MPI quanto por *socket*, para comparação dos desempenhos. O cabo foi dividido com um corte paralelo ao plano *yz* (para duas máquinas do *cluster*). Neste algoritmo, as mensagens são trocadas tal como indicado pela Fig. 4 e a divisão do domínio de análise em subdomínios foi feita conforme indicado na Fig. 7. A malha computacional possui 62 células na direção *z*.

Nove simulações foram realizadas, com os seguintes parâmetros: o diâmetro do cabo em células ( $nx_1 = ny_1$ ) foi simulado com os valores 79, 179 e 279 células. A Fig. 8 exibe o gráfico do tempo de simulação médio considerando quatro mil passos de tempo, obtido para um total de três execuções para cada valor de  $ny_1$ . Na Fig. 8, não se considerou troca de mensagens pela rede e os tempos obtidos são idênticos para *socket* e *MPI*.

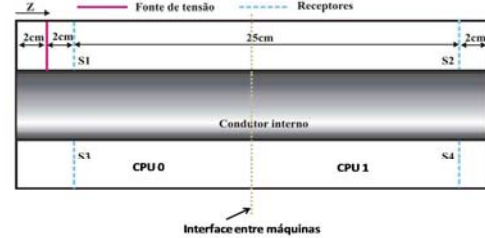


Fig. 7. Representação da estrutura simulada – corte longitudinal.

Na Fig. 9 os parâmetros de execução não foram alterados, porém as trocas de mensagens pela rede *gigabit* do *cluster* são consideradas. Foi observada uma redução máxima de aproximadamente 60% no tempo total de execução quando o *socket* foi usado.

A Fig. 10 apresenta a diferença entre os dados das Figs. 10 e 9, referente ao tempo entre a execução da aplicação com envios e sem envios de mensagens. Nota-se que a utilização do *socket* para o envio e recepção de mensagens pela rede *gigabit*, aplicado ao caso, proporciona uma redução do tempo máximo de troca de mensagens, pela rede, em aproximadamente 84%.

A Fig. 11 mostra saídas dos programas baseados em MPI e *socket*, além dos resultados obtidos seqüencialmente, para o campo elétrico transitório registrado no ponto S2 da Fig. 7. Foram considerados os três diferentes diâmetros de cabo indicados na Fig. 11. Esta Figura mostra a concordância de resultados quando utilizadas as bibliotecas LAM/MPI e *mysock* com os resultados obtidos seqüencialmente, para todos os casos testados. Os resultados obtidos são idênticos aos calculados em [17].

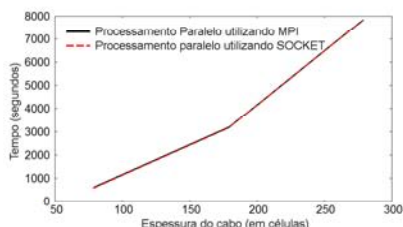


Fig. 8. Tempos de processamento referentes à aplicação do cabo (sem trocas de mensagem pela rede).

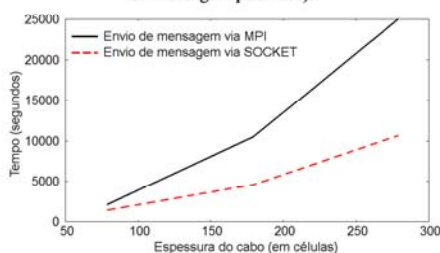


Fig. 9. Tempo total de execução (processamento e transferência pela rede).

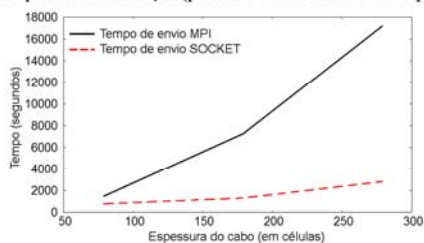


Fig. 10. Tempo de transferência de mensagens via LAM/MPI e *mysock*.

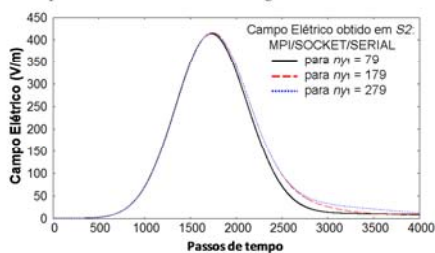


Fig. 11. Representação da saída Campo Elétrico (V/m).

## I. CONCLUSÃO

Tomando-se como referência o tempo necessário para realizar transferência de dados em ambiente *Beowulf* através do *software* LAM/MPI, verificou-se que o uso de *sockets* pode reduzir consideravelmente este tempo. A avaliação da biblioteca desenvolvida (*mysock*) foi realizada por meio de um modelo eletromagnético de onda completa de um cabo coaxial de alta tensão, solucionando-se

numericamente as equações de Maxwell no domínio do tempo, através do método LN-FDTD.

Observou-se uma redução máxima próxima de 84% no tempo de transferência das informações (Fig. 10) referentes à paralelização do método LN-FDTD. Vale ressaltar que o uso de *sockets* para realizar a transferência de mensagens em um *cluster* *Beowulf* é uma técnica de baixo nível de acesso ao sistema e, portanto, praticamente todos os problemas de transferência de dados e controle de processos são de responsabilidade do programador, algo que torna a MPI mais atraente. Porém, os resultados obtidos neste trabalho mostram que o uso de técnicas primárias de transferência pode produzir ganhos consideráveis em termos do tempo total de execução.

Como trabalho futuro, é proposto o desenvolvimento de uma biblioteca baseada em *sockets* capaz de trabalhar com mais de dois computadores.

## REFERÊNCIAS

- [1] S. C. Chapra, R. P. Canale, *Numerical Methods for Engineers*, 2<sup>nd</sup> ed. New York: McGraw-Hill, 1990.
- [2] W. H. Barros Jr., R. C. F. Rocha, R. M. S. de Oliveira e C.L. S. Souza Sobrinho, "Localização de Defeitos por Simulação LN-FDTD em Cabos HV Usando Algoritmos Genéticos e PSO," in *MOMAG*, 2008.
- [3] K. Yee, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302-307, 1966.
- [4] R. M. S. Oliveira and C. L. S. Sousa Sobrinho, "UPML formulation for truncating conductive media in curvilinear coordinates," *Springer Netherlands*, vol. 46, N<sup>o</sup>. 4, pp. 1572-9265, December 2007.
- [5] R. M. S. Oliveira and C. L. S. Sousa Sobrinho, "Computational Environment for Simulating Lightning Strokes in a Power Substation by Finite-Difference Time-Domain Method," *IEEE Transactions on Electromagnetic Compatibility*, vol. 51, Issue 4, pp. 995-1000, November 2009.
- [6] J. S. Araújo, C. L. S. Sousa Sobrinho, J. M. Rocha, L. A. Guedes, R. Y. Kawasaki, R.O. dos Santos, "Analysis of Antennas by FDTD Method Using Parallel Processing with MPI," in *International Microwave and Optoelectronics Conference*, Foz do Iguaçu-PR, Brazil, September 2003.
- [7] E. Ogasawara, D. Oliveira, E. Seabra, C. E. Barbosa, V. Braganholo, R. Elias, A. Coutinho and M. Mattoso, "Exploring Many Task Computing in Scientific Workflows," in *2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Portland, Oregon, November 2009.
- [8] P. S. Pacheco, *Parallel programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.
- [9] R. Guapo, L. Sousa, S. Yamagiwa, "On the Implementation and Evaluation of Berkeley Sockets on Maestro2 cluster computing environment," in *The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, pp. 317-324, 2005.
- [10] A. Vrenios, *Linux Cluster Architecture*, Indianapolis: SAMS, 2002.
- [11] "The lam-mpi parallel computing at the web" [Online]. Available: <http://www.lam-mpi.org>. [Accessed: February 24, 2010].
- [12] I. T. Foster, *Designing and Building a Parallel Programs: Concepts and Tools for Parallel Software Engineering*, New York: Addison-Wesley, 1995.
- [13] A. S. Tanenbaum, *Computer Networks*, 4<sup>th</sup> ed. Prentice Hall, 2003.
- [14] "Linux Socket-API" [Online]. Available: <http://linux.die.net/man/7/socket>. [Accessed: February 24, 2010].
- [15] P. Van der Linden, *Expert C Programming: Deep C Secrets*, California: Prentice Hall, 1994.
- [16] "Beowulf.org" [Online]. Available: <http://www.beowulf.org>. [Accessed: February 24, 2010].
- [17] W. H. Barros Júnior, R. M. S. Oliveira and C. L. S. Sousa Sobrinho, "Analysis and Synthesis of PD Sources in HV Cables by the LN-FDTD Method and Neural Networks," in *International Conference on Grounding and Earthing & 3rd International Conference on Lightning Physics and Effects*, Florianópolis, Brazil, November 2008.

# APÊNDICE E

## Casos E1 e E2

Os Casos E1 e E2 são similares aos Casos A e C, respectivamente. Nestas novas simulações, a posição do canal de descarga atmosférica foi modificada tal como indicado na Figura E.1.

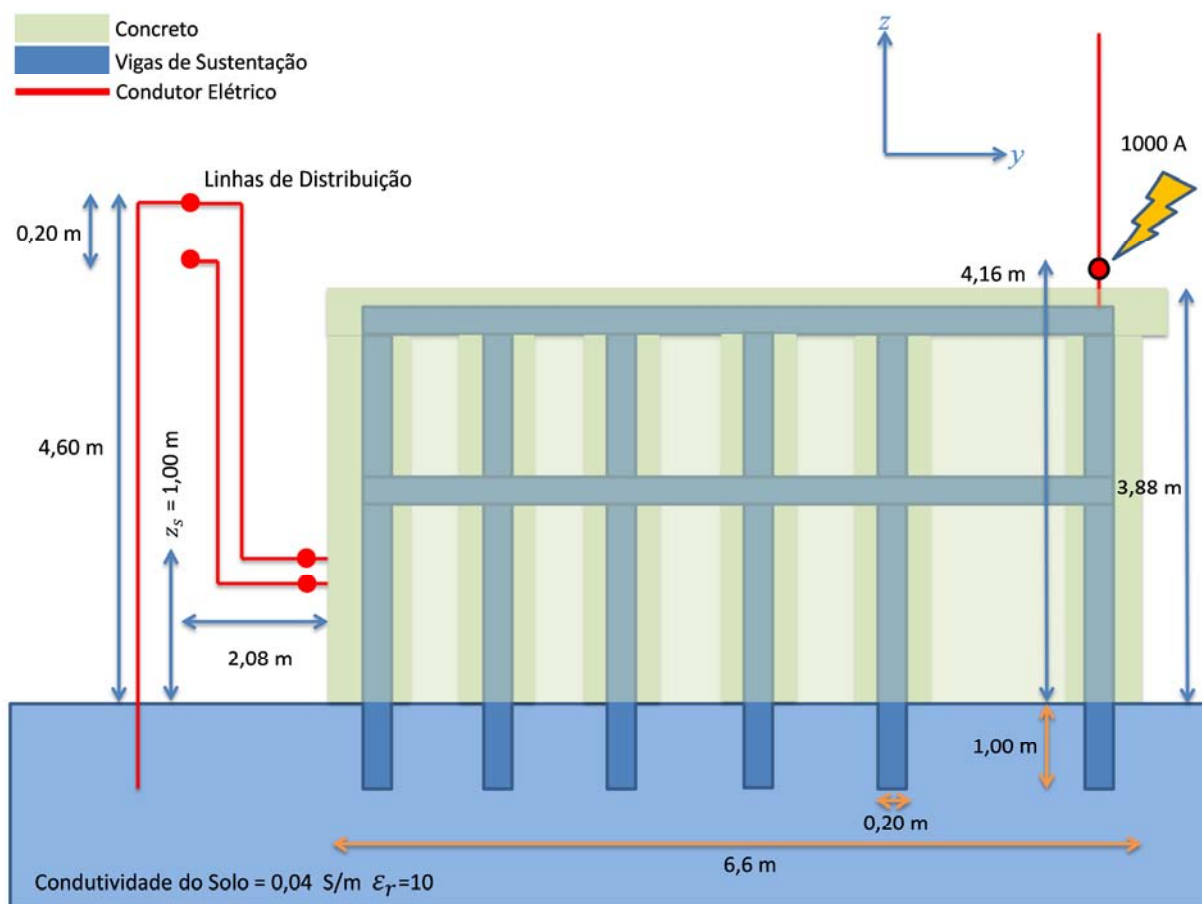


Figura E.1: Caso E1 – localização modificada da fonte de descarga atmosférica.

Nas Figuras E.2 – E.6 são apresentados os gráficos dos Casos E1 e E2 para as oito tomadas, fase e neutro.



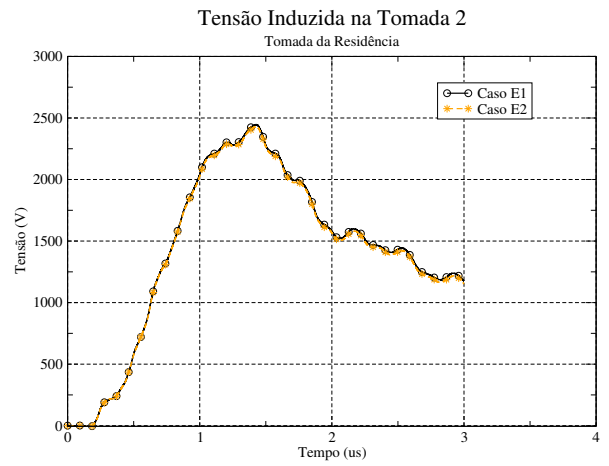
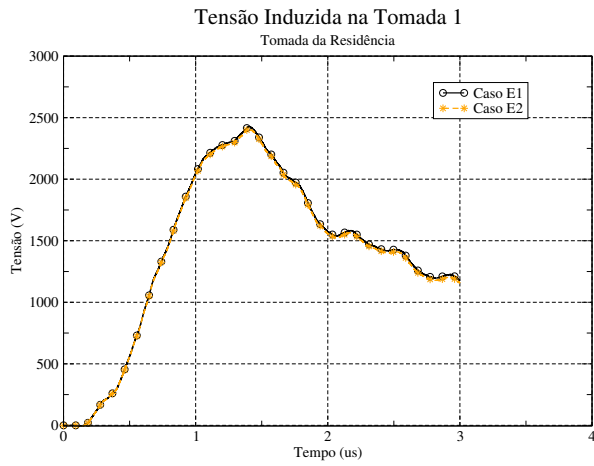


Figura E.3: Tensão induzida para (a) Tomada 1; (b) Tomada 2

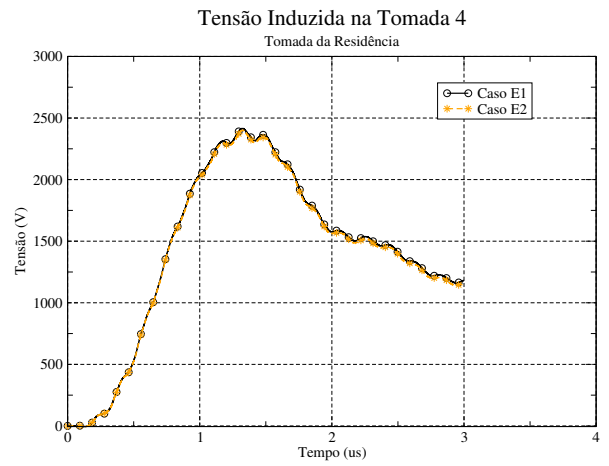
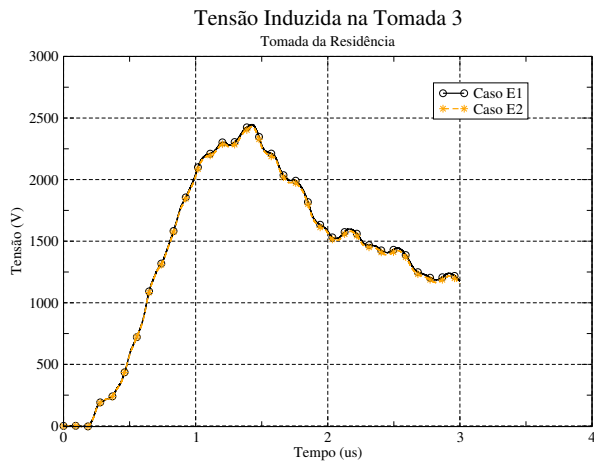


Figura E.3: Tensão induzida para (a) Tomada 3; (b) Tomada 4

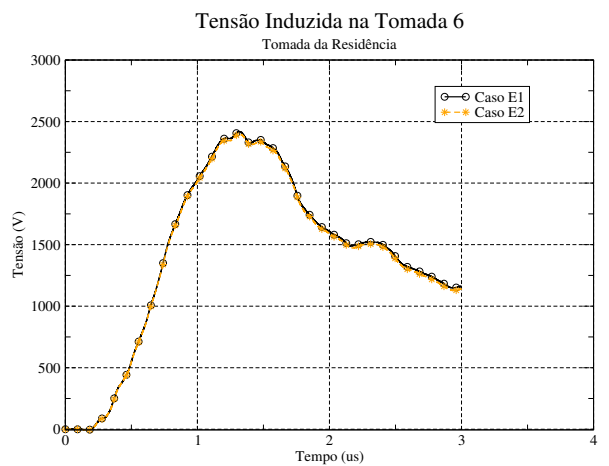
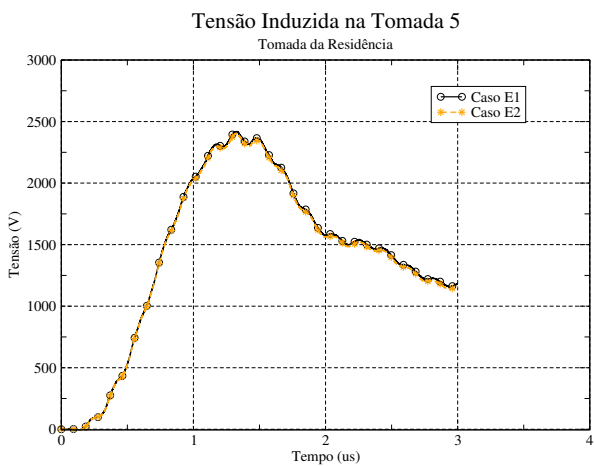


Figura E.4: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

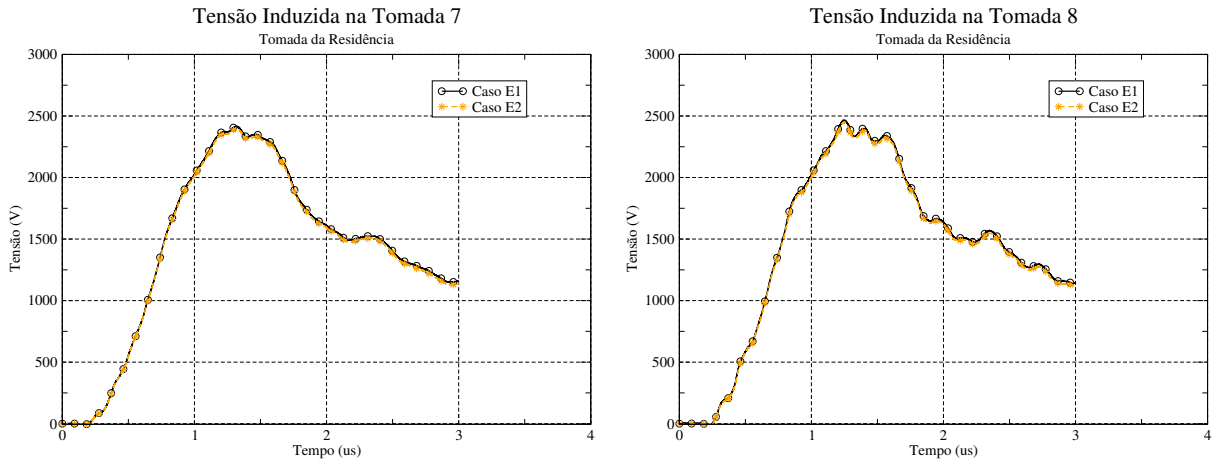


Figura E.5: Tensão induzida caso (a) Tomada 7; (b) Tomada 8

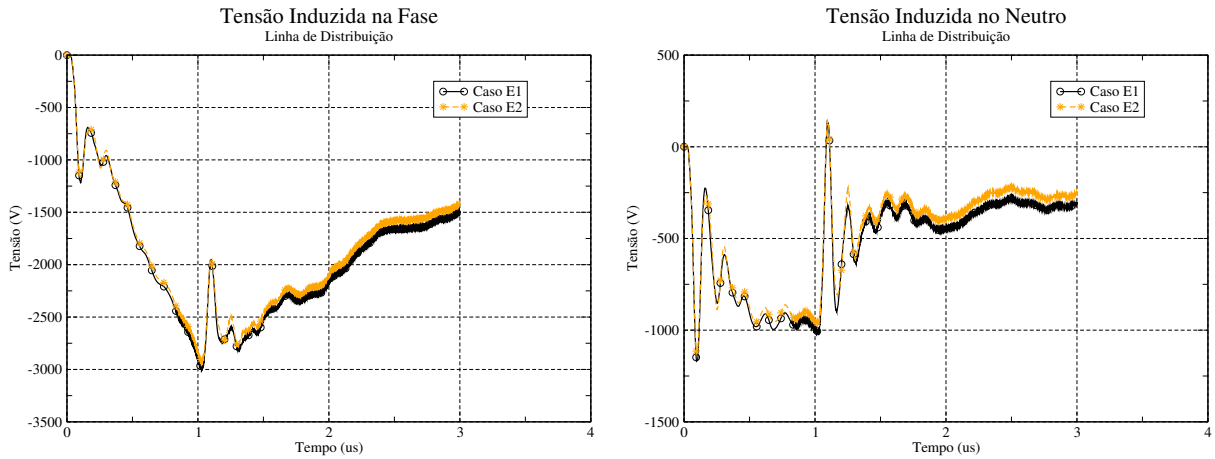


Figura E.6: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

Conforme esperado, as tensões induzidas nas linhas de distribuição foram reduzidas em relação ao Caso A e C, dada a maior distância entre as linhas de distribuição e o canal de descarga. Além disso, *todas* as tensões induzidas nas tomadas foram reduzidas neste caso, confirmando a importância da posição relativa entre canal de descarga e linhas de distribuição.

### Casos E3 e E4

As simulações dos Casos E3 e E4 são similares aos Casos A e C respectivamente, entretanto as tomadas internas foram aterradas a viga de sustentação da residência (Figura E.7). Foi construída uma malha de aterramento interligando o aterramento do neutro da linha

de distribuição e as vigas de sustentação da residência, juntamente com o condutor elétrico do pára-raios do Caso C, conforme mostra a Figura E.8.

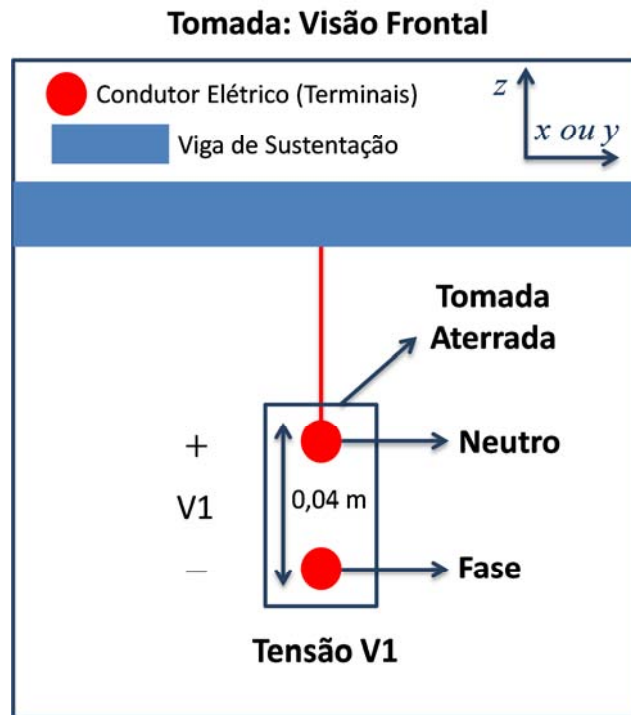


Figura E.7: Visão frontal da tomada aterrada a viga de sustentação da residência.



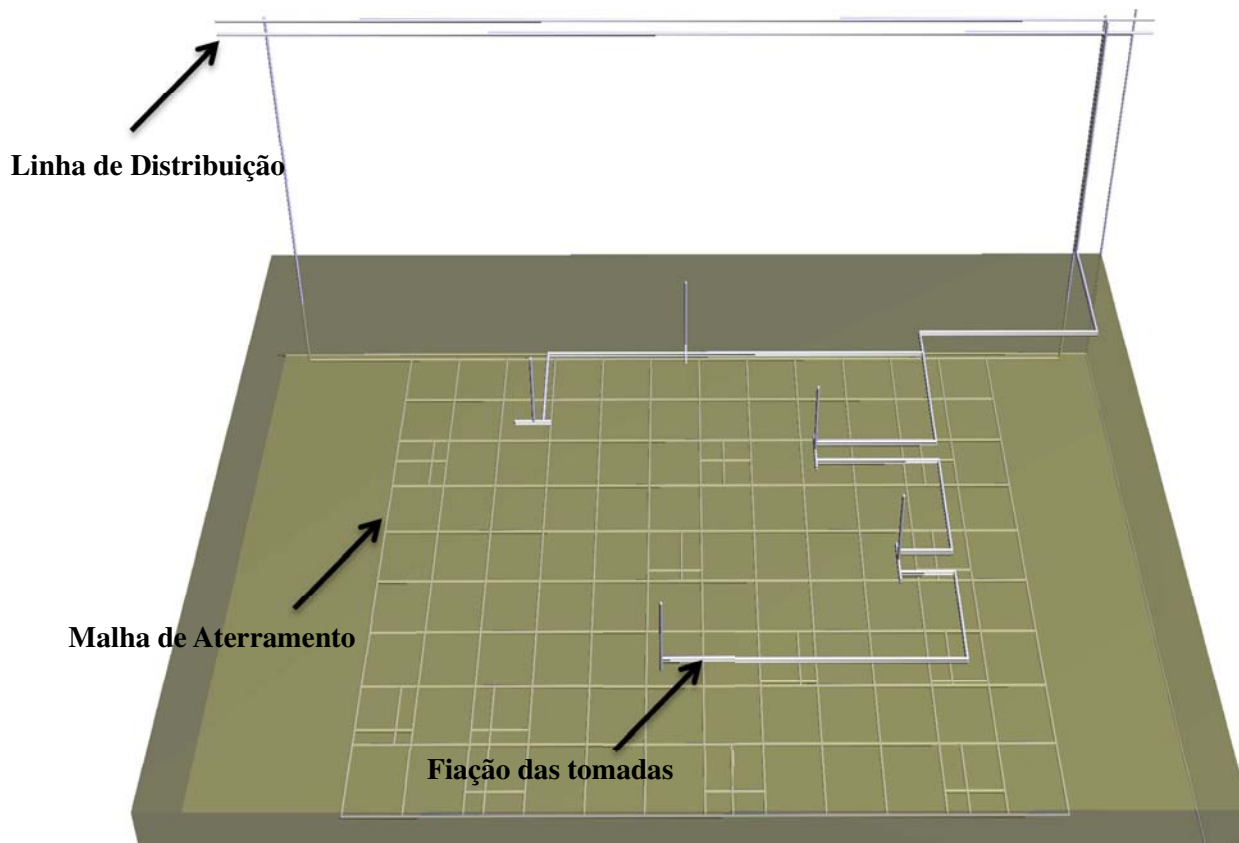


Figura E.8: Estrutura da malha de aterramento da residência.

A Figura E.9 apresenta o gráfico da distribuição espacial de  $|\vec{E}|$  para  $z = 0,84\text{m}$  ( $k=21$ ) após  $1\mu\text{s}$ , no plano da malha de aterramento apresentada pela Figura E.4.

$$E [ z = 8.400000\text{E-}01 \text{ m } ( k = 21 ) ; t = 1 \text{ us } ] = \{ 0 - 2.4\text{e}+03 \} \text{ V/m}$$

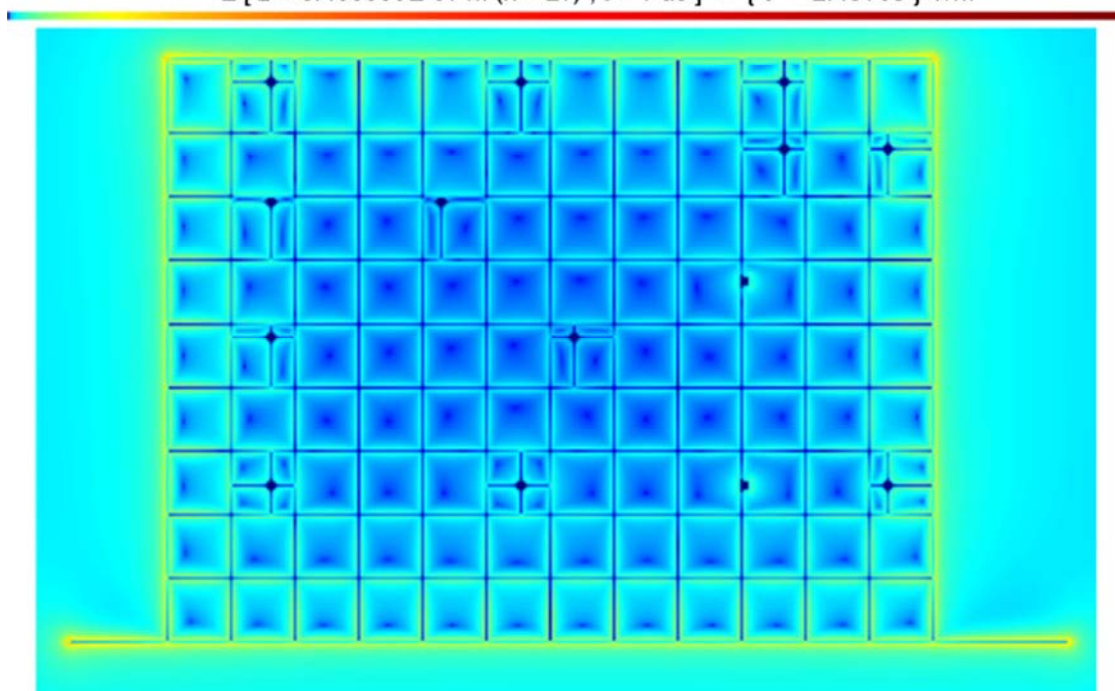


Figura E.9: Distribuição espacial de  $|\vec{E}|$  para  $z = 0,84\text{m}$  ( $k=21$ ).

Nas Figuras E.10 – E.14 são apresentados os gráficos das tensões induzidas para os Casos E3 e E4 para as oito tomadas fase e neutro

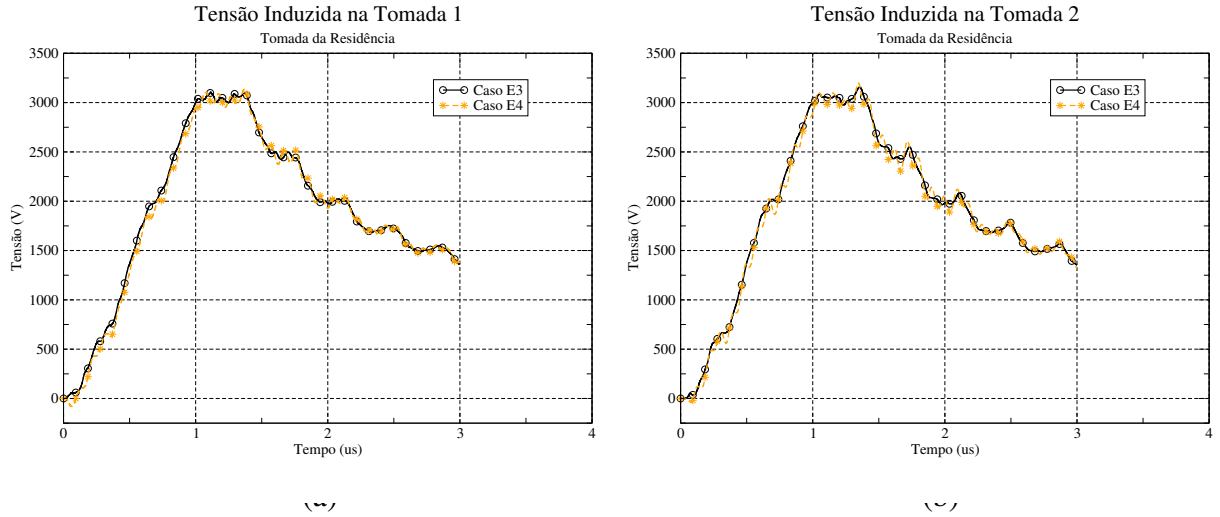


Figura E.10: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

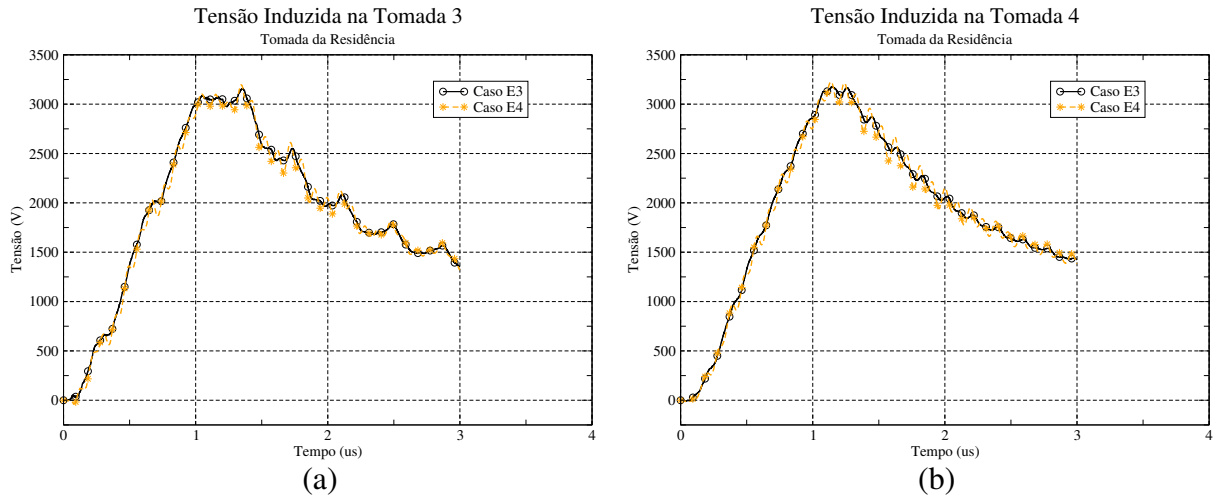
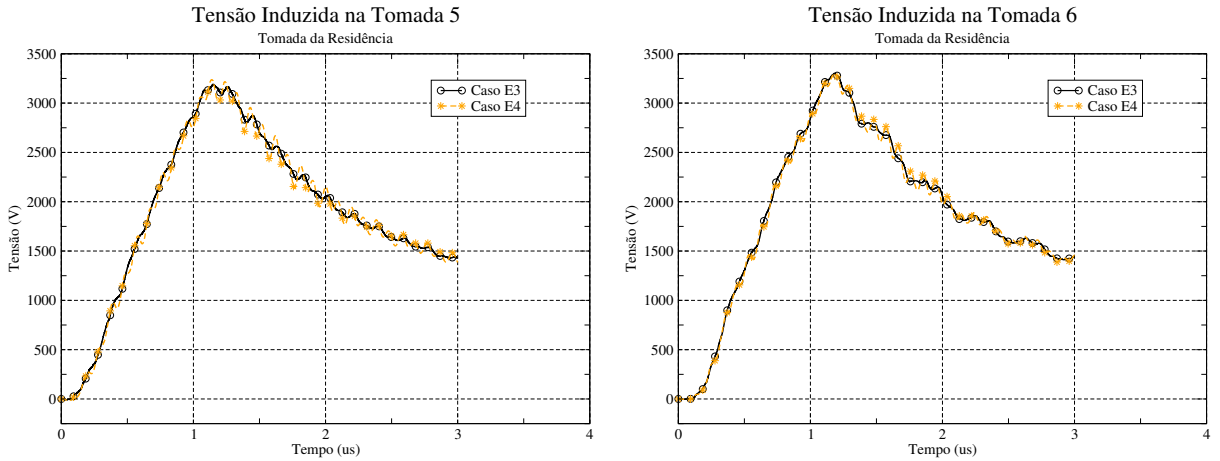


Figura E.11: Tensão induzida para (a) Tomada 3; (b) Tomada 4.



(a) (b)  
 Figura E.12: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

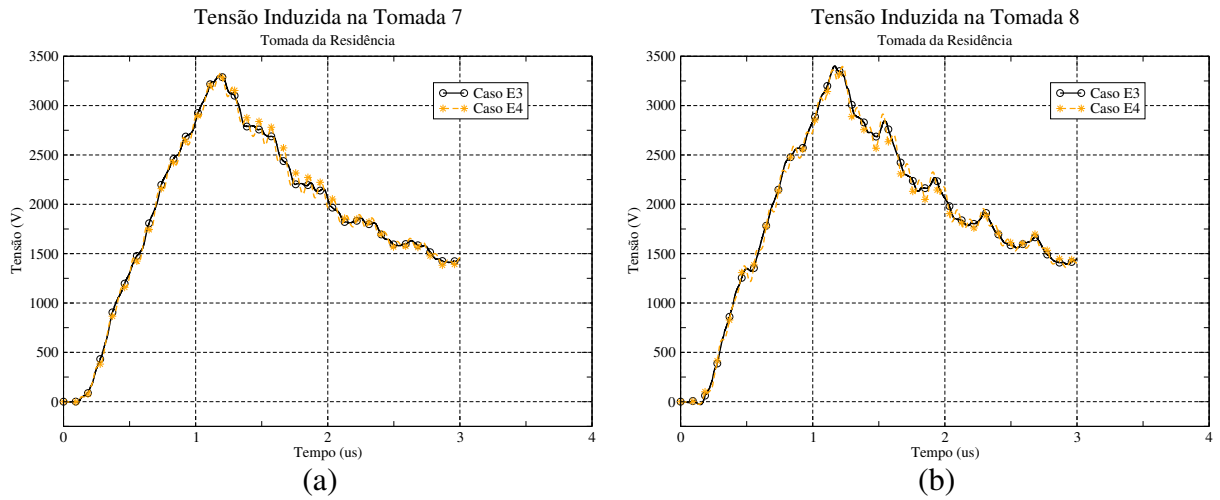


Figura E.13: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

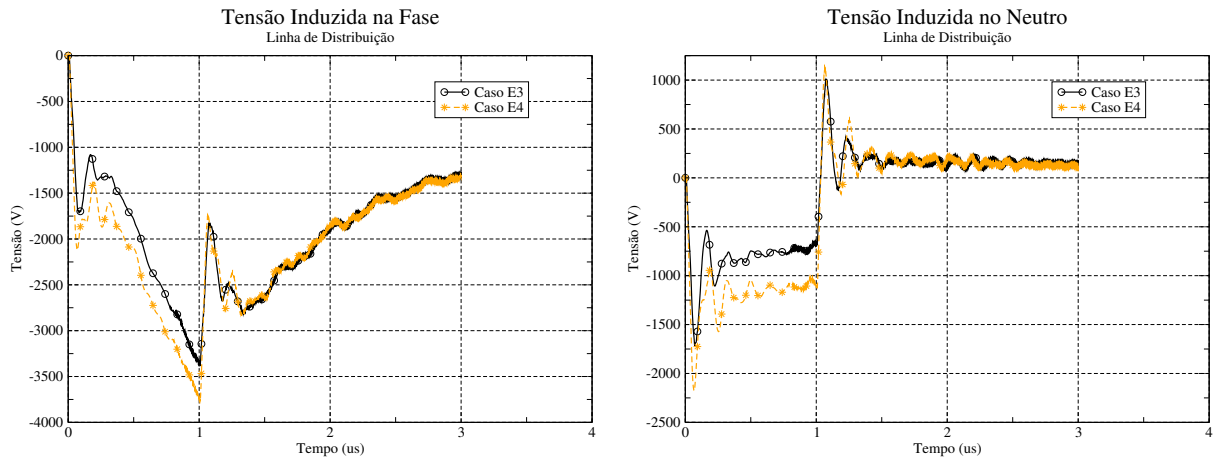


Figura E.14: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

## Casos E5 e E6

As simulações dos Casos E5 e E6 são similares aos Casos E3 e E4 respectivamente, porém, não há aterramento do neutro das tomadas (Figura E.3), e a malha de aterramento (Figura E.4) foi mantida.

Nas Figuras E.15 – E.19 podem ser observados os gráficos da tensão induzida para as oito tomadas, fase e neutro da simulação.

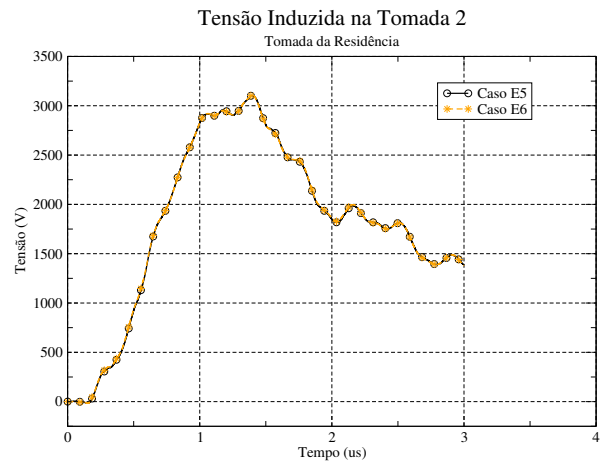
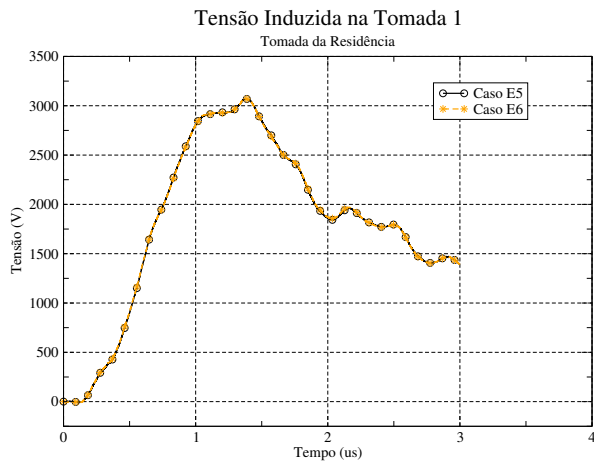


Figura E.15: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

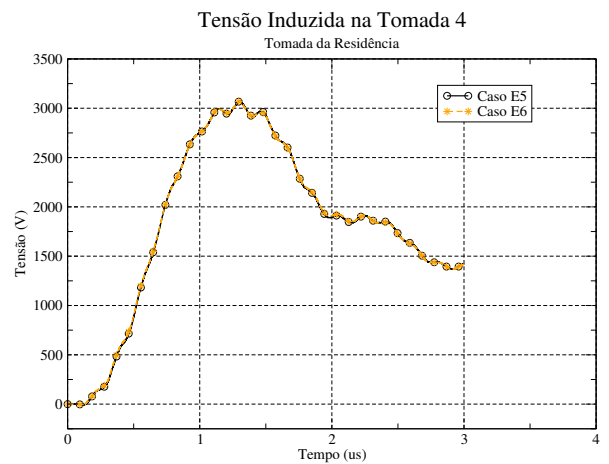
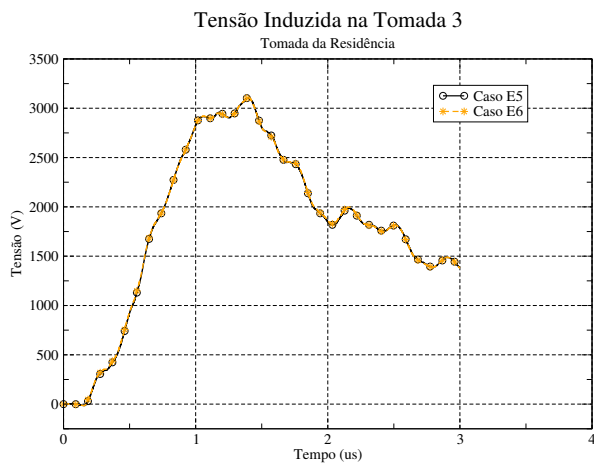


Figura E.16: Tensão induzida para (a) Tomada 3; (b) Tomada 4.

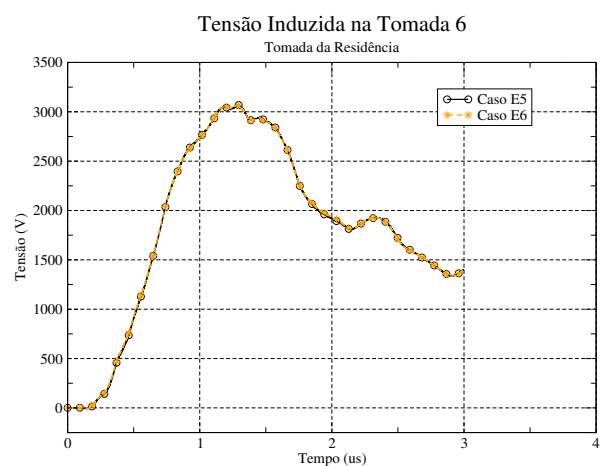
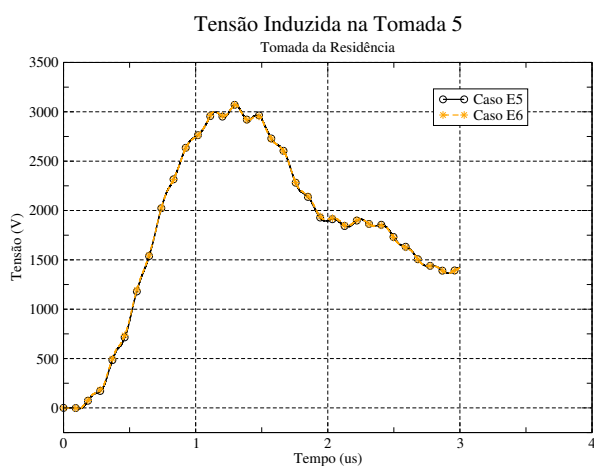


Figura E.17: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

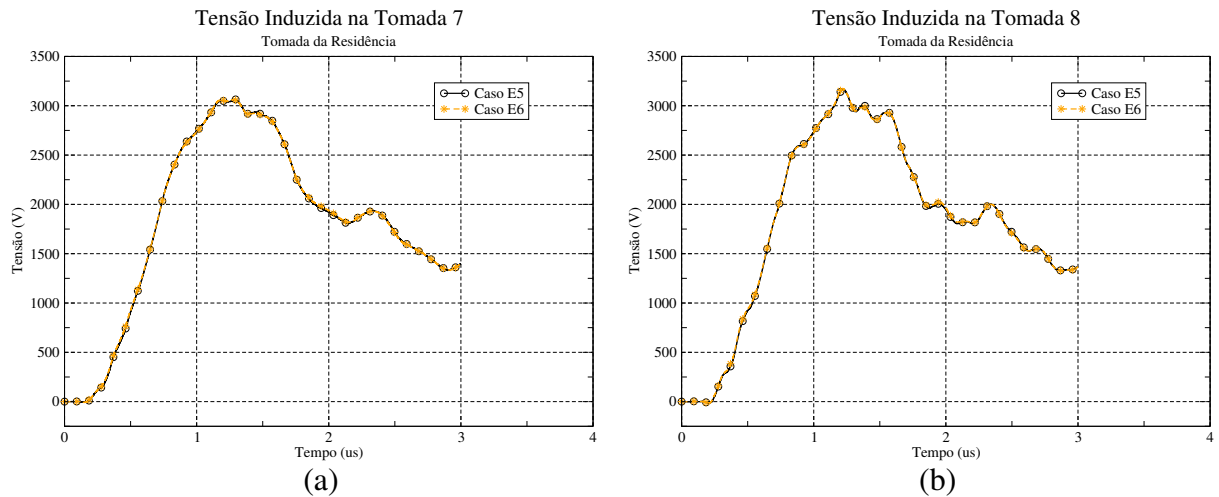


Figura E.18: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

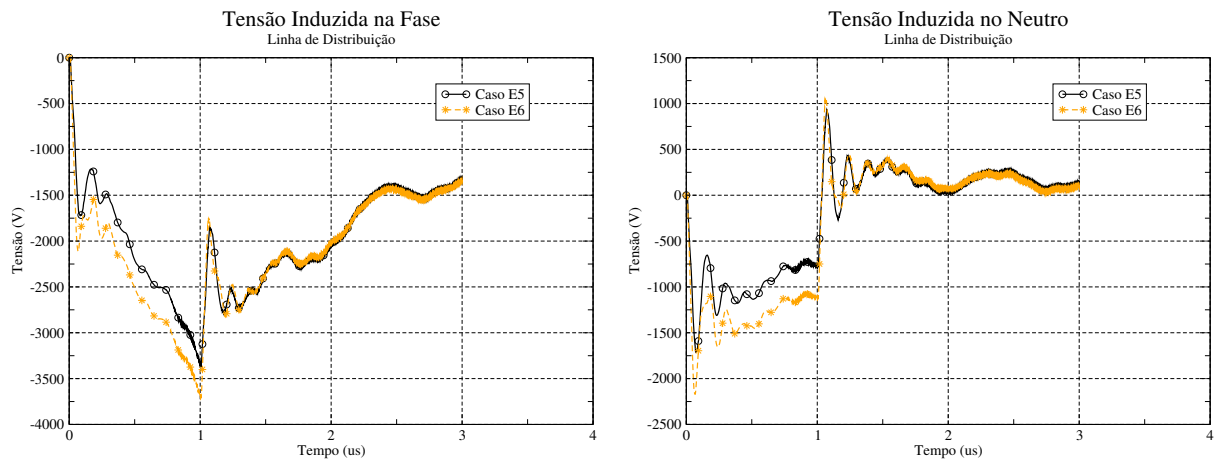


Figura E.19: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

## Casos E7 e E8

As simulações dos Casos E7 e E8 diferem das simulações dos Casos E5 e E6 respectivamente, em que há o aterramento do neutro das tomadas (Figura E.3), porém não há a malha de aterramento da Figura E.4.

Nas Figuras E.20 – E.24 são observados os gráficos da tensão induzida para as oito tomadas, fase e neutro da simulação.

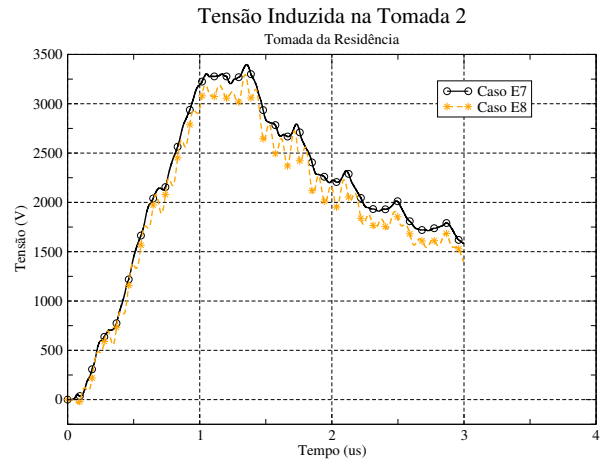
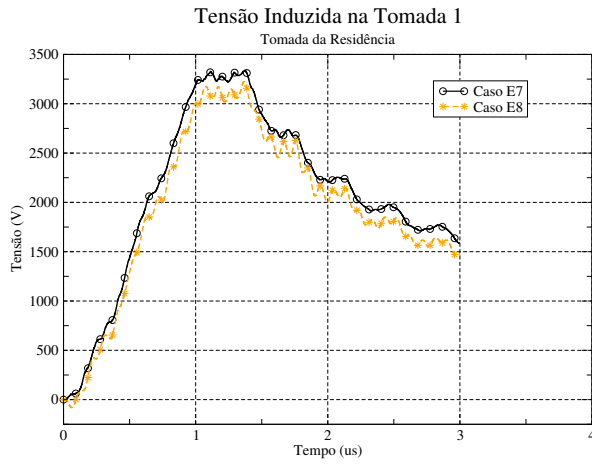


Figura E.20: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

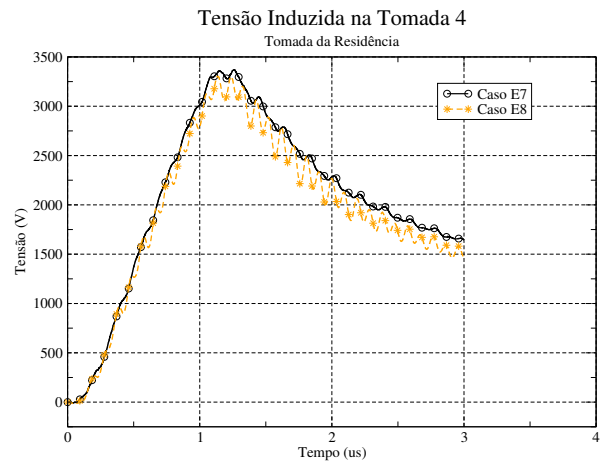
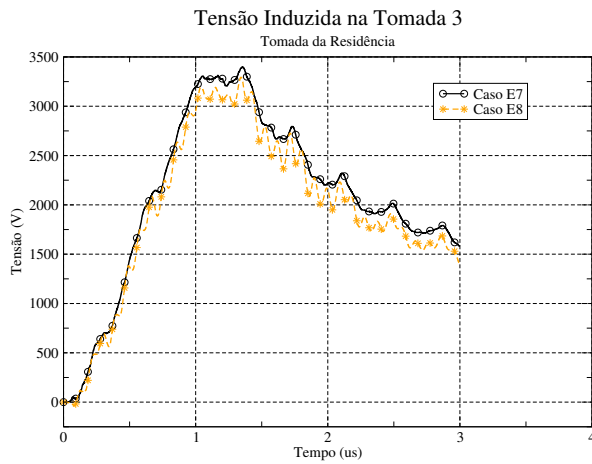


Figura E.21: Tensão induzida para (a) Tomada 3; (b) Tomada 4.

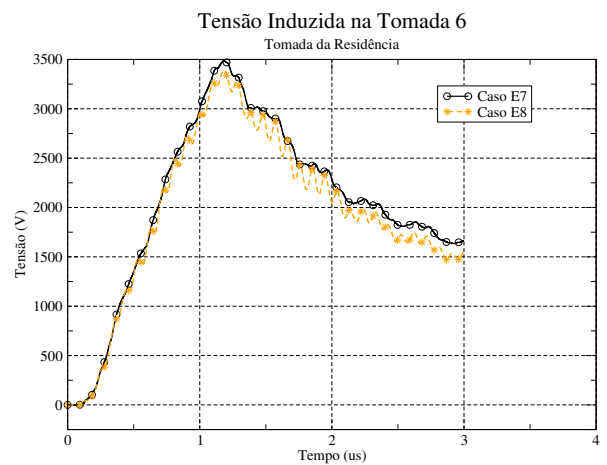
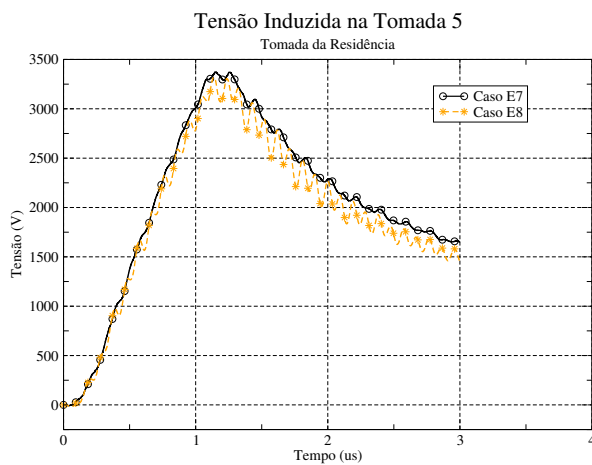


Figura E.22: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

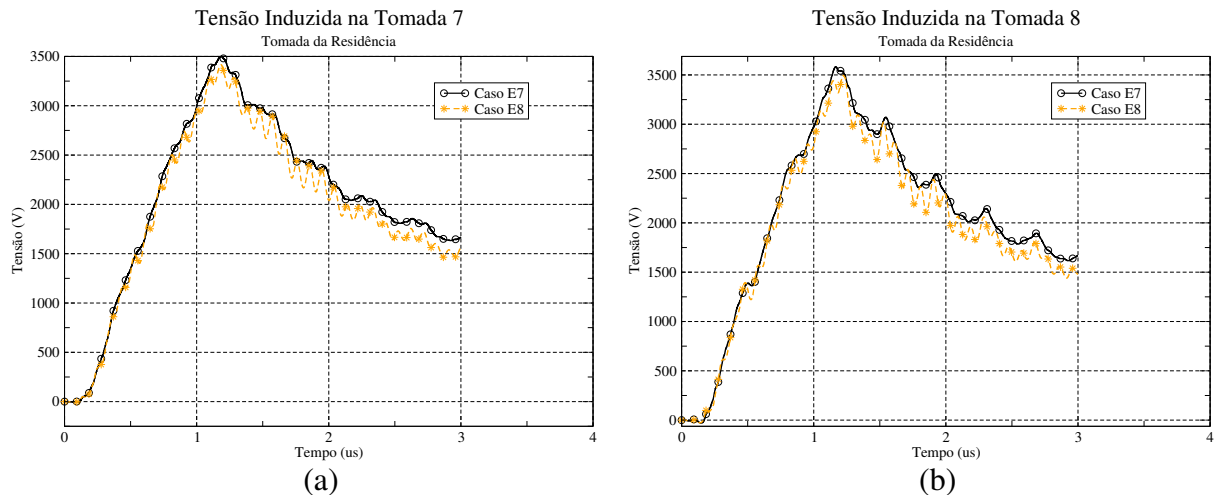


Figura E.23: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

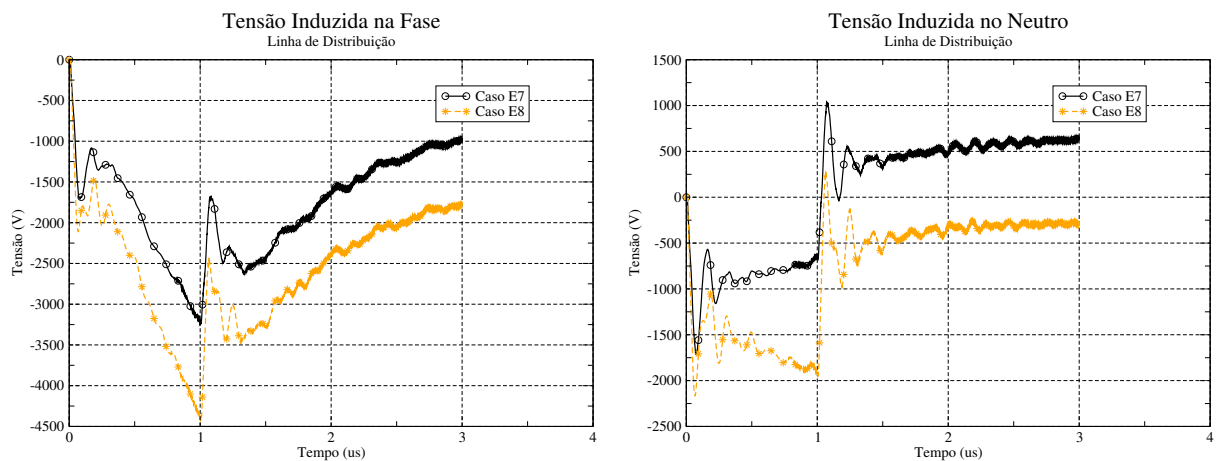


Figura E.24: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

## Casos E9, E10, E11, E12 e E13

As simulações dos Casos E9 e E10 são similares aos Casos E3 e E4 respectivamente, porém, não há o aterramento do neutro das tomadas (Figura E.3) e a malha de aterramento foi modificada para um sistema de aterramento disposto ao redor da residência, na forma de anel, conforme mostra a Figura E.25.

O sistema de aterramento disposto em anel contorna toda a estrutura da residência a uma distância de 1m da mesma. O anel está a 0,5m da superfície do solo e a cada 2m o anel possui uma haste vertical de 0,5m (Figura E.25).

O Caso E11 é similar ao Caso E10, porém o neutro das tomadas da residência foi interligado ao anel de aterramento.

O Caso E12 é similar ao Caso E10, porém o condutor neutro do sistema de

distribuição não é aterrado (Figura E.25).

O Caso E13 é similar ao Caso E12, porém a distância entre as linhas de distribuição, que era de 0,20m (Figura E.25), passou a ser de 0,04m.

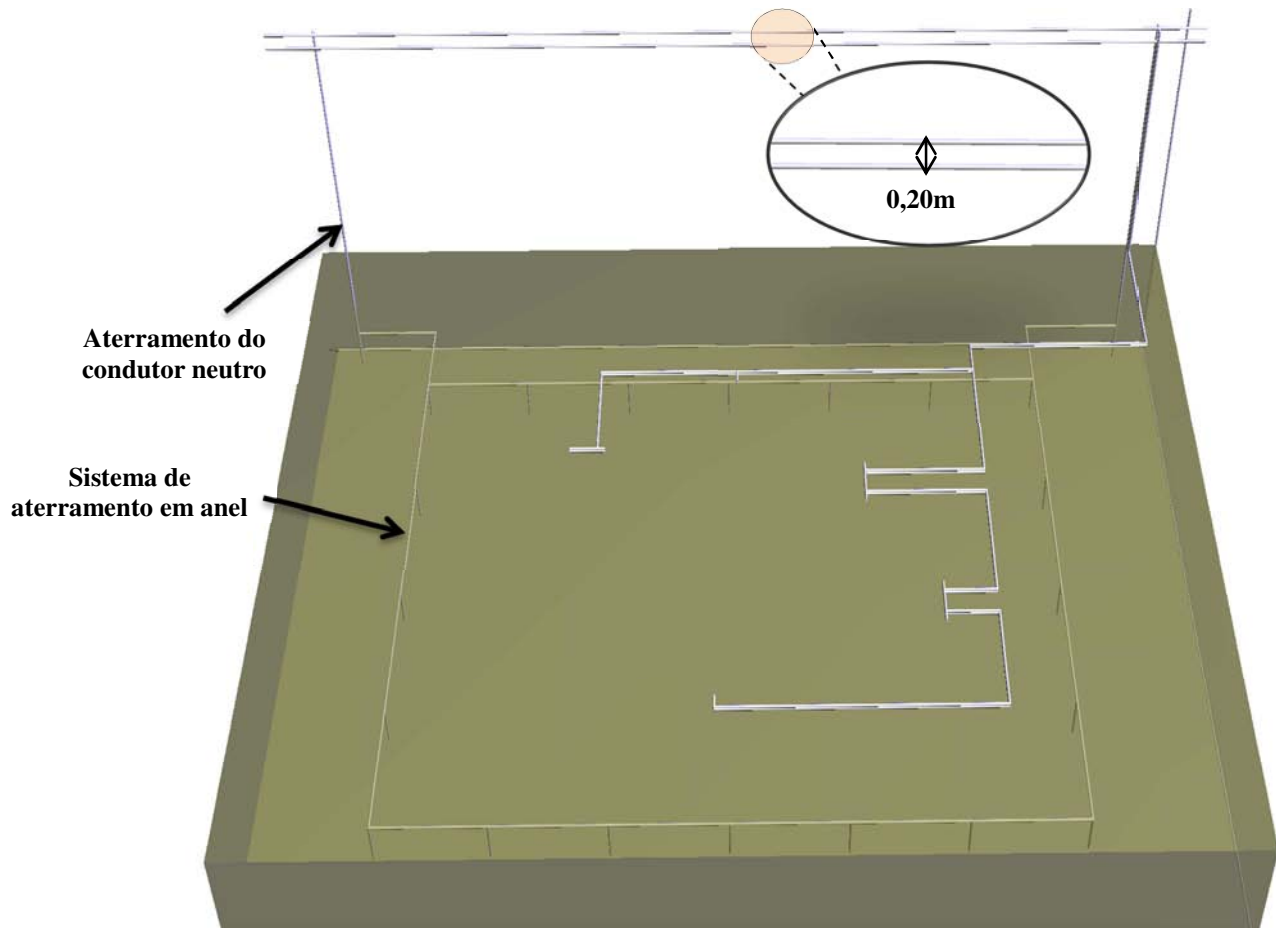


Figura E.25: Sistema de aterramento disposto em anel.

Nas Figuras E.26 – E.30 são apresentados os gráfico das oito tomadas, fase e neutro para os Casos E9 – E13.



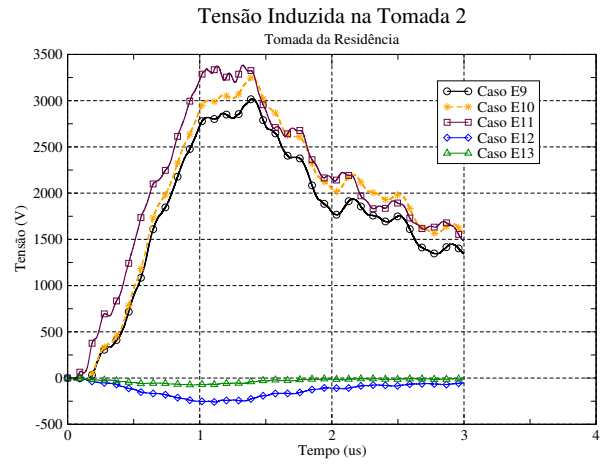
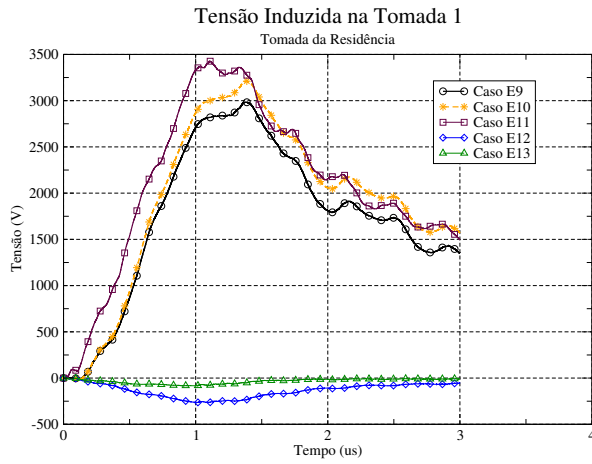


Figura E.26: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

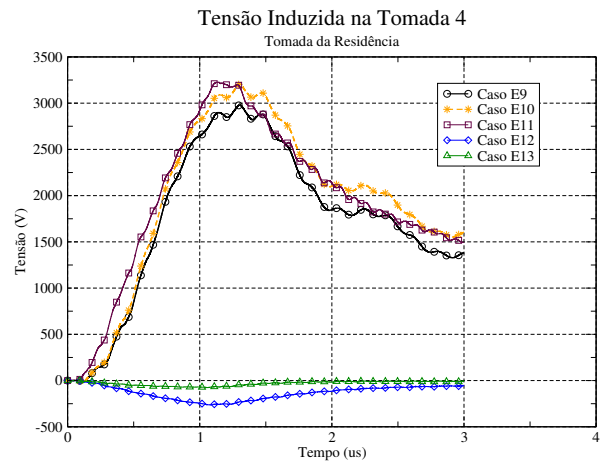
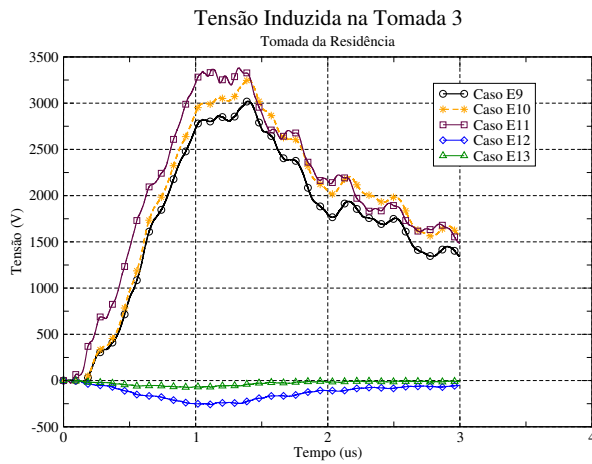


Figura E.27: Tensão induzida para (a) Tomada 3; (b) Tomada 4.

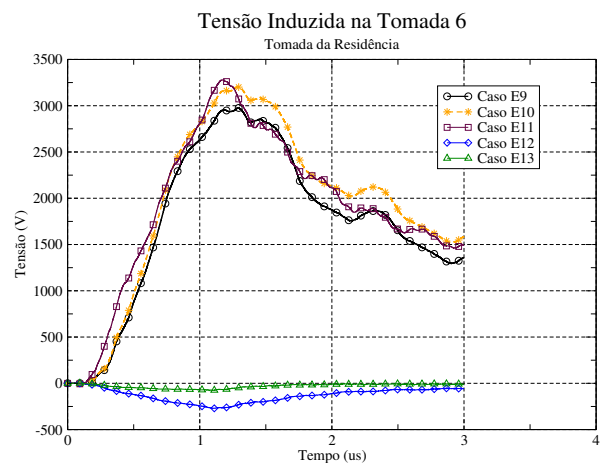
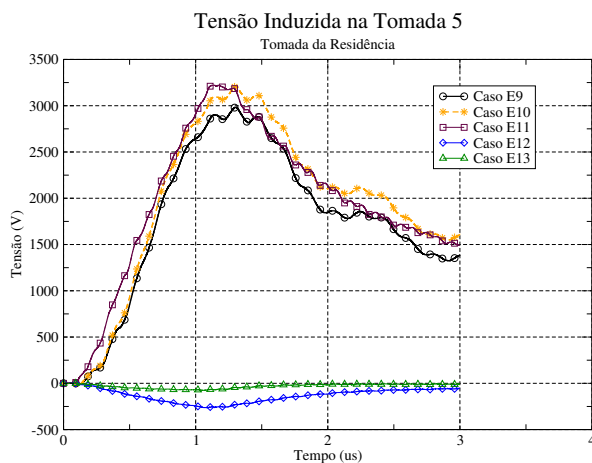


Figura E.28: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

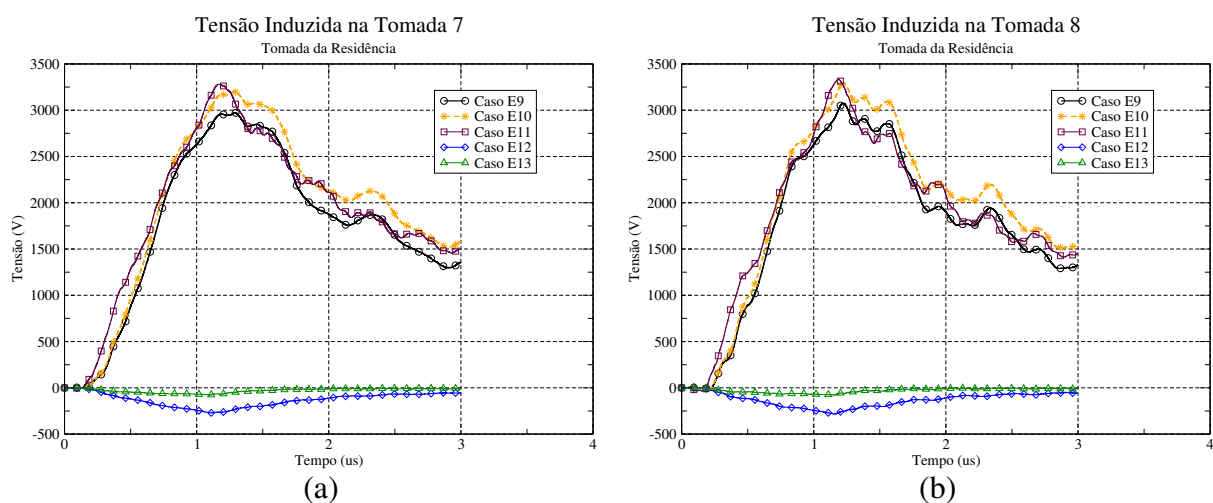


Figura E.29: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

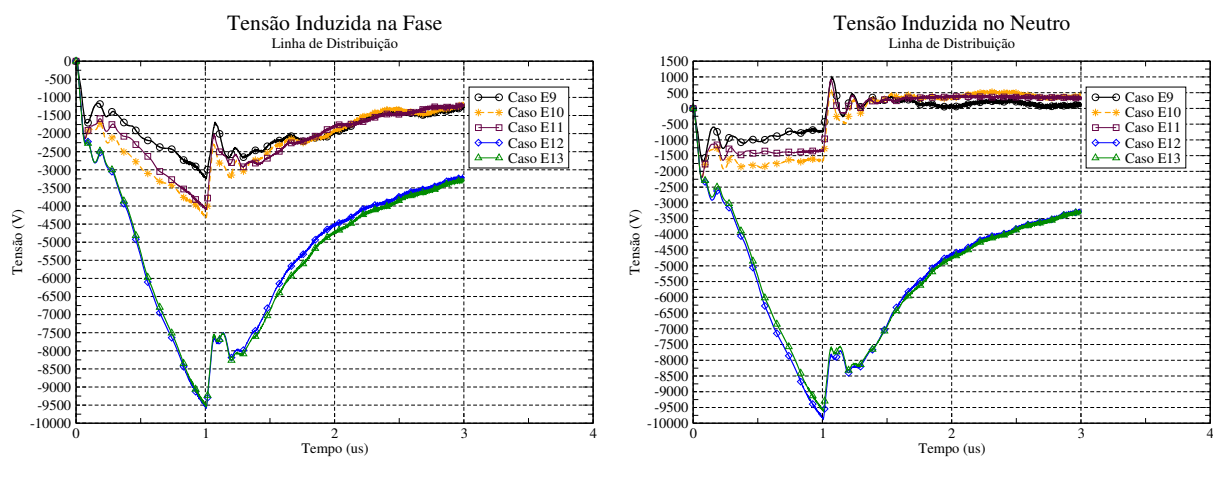


Figura E.30: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

Dessa forma, verifica-se que é de fundamental importância não aterrar o condutor neutro nas linhas de distribuição, pois tal procedimento tende a aumentar substancialmente as tensões induzidas entre os terminais das tomadas.

## Caso E14

A simulação do Caso E14 é similar a do Caso A, porém a distância entre as linhas de distribuição, que era de 0,20m, passou a ser de 0,04m. A altura do condutor neutro foi mantida a mesma utilizada no caso A. Nas Figuras E.31 – E.35 são apresentados os gráficos da tensão induzida para as oito tomadas, fase e neutro do Caso E14.

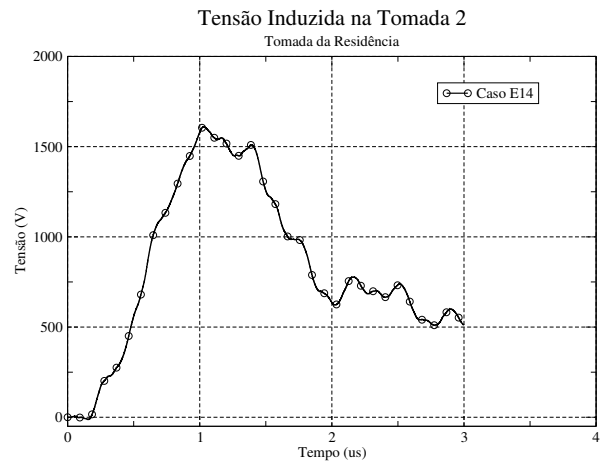
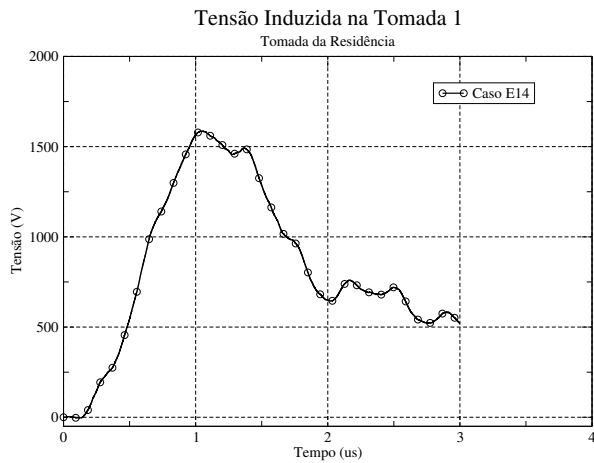


Figura E.31: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

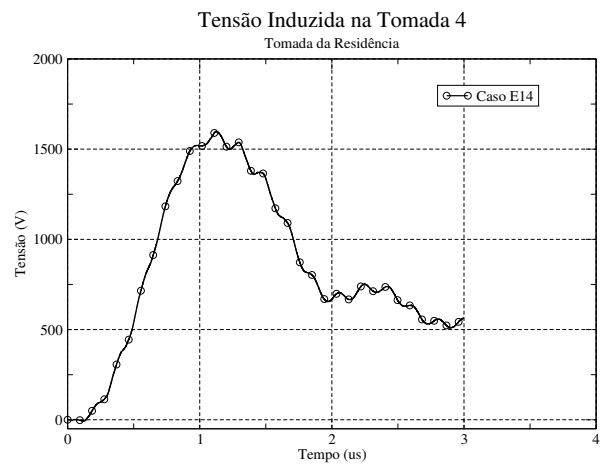
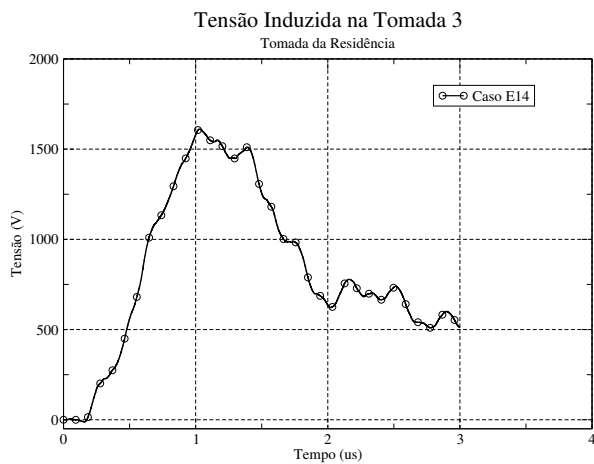


Figura E.32: Tensão induzida para (a) Tomada 3; (b) Tomada 4.

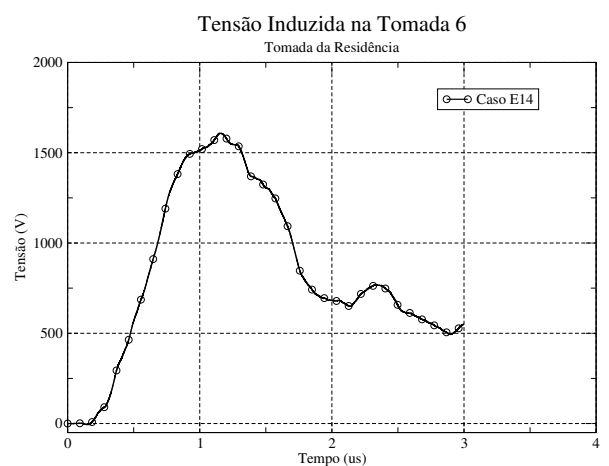
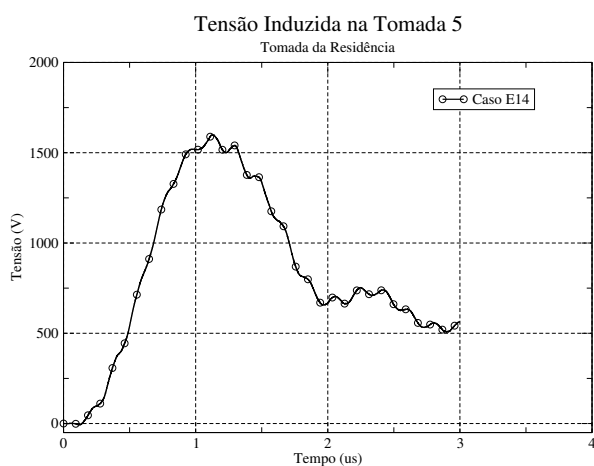


Figura E.33: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

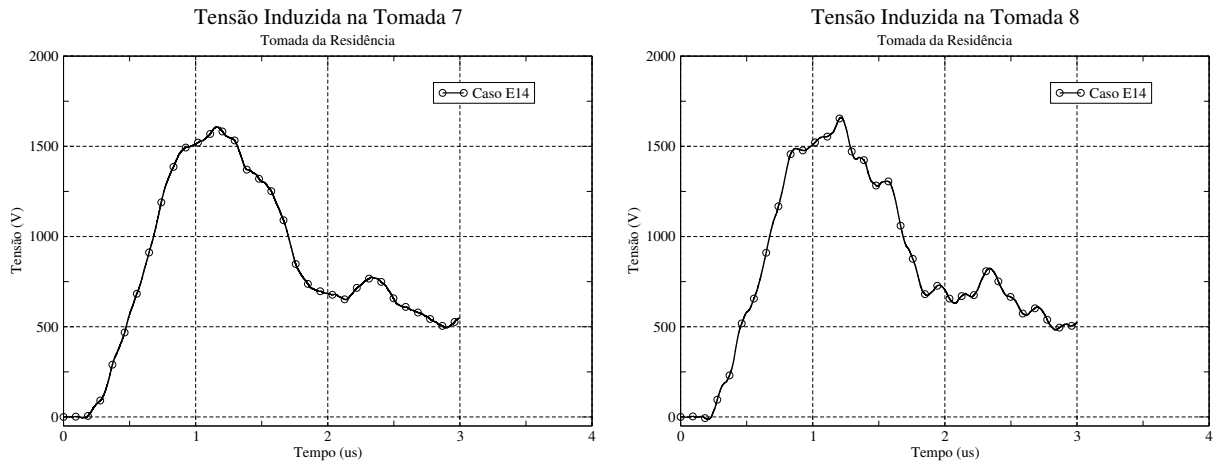


Figura E.34: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

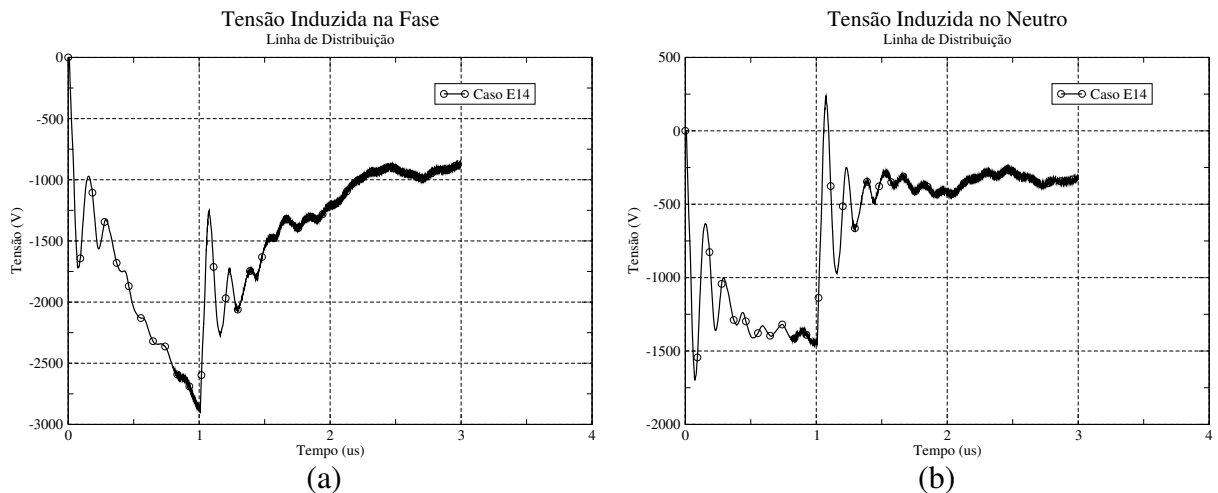


Figura E.35: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

Conforme esperado, a redução da distância entre as linhas de distribuição (fase e neutro) tende a reduzir as tensões induzidas entre os terminais das tomadas. Isto ocorre pois o módulo da função tensão induzida entre solo e fase (Figura E.35a) apresenta menores valores em relação ao Caso A (Figura 4.6a) e a tensão induzida entre solo e neutro praticamente não sofreu alterações em relação ao Caso A (Figura 4.6b), devido ao aterramento do condutor neutro.

## Caso E15

Este caso é similar ao Caso C, porém os condutores fase e neutro do sistema de distribuição estão posicionados à mesma altura ( $z = 6,48\text{m}$ ) e o condutor neutro não foi aterrado, conforme mostra a Figura E.36. O afastamento entre as linhas é de  $0,04\text{ m}$ .

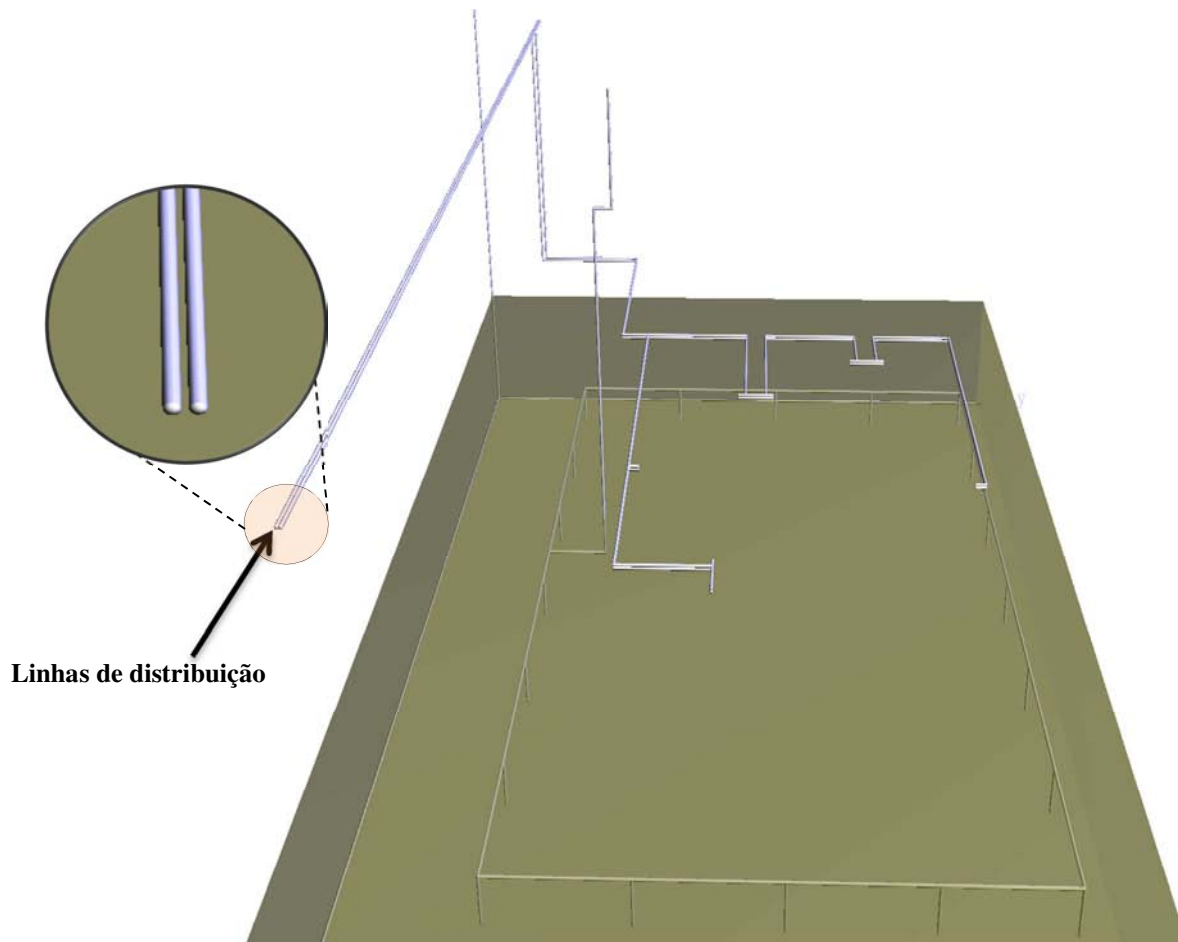


Figura E.36: Definição do posicionamento das linhas de distribuição para o Caso E15.

Nas Figuras E.37 – E.41, são apresentados os gráficos da tensão induzida obtida nas oito tomadas fase e neutro para o Caso E15

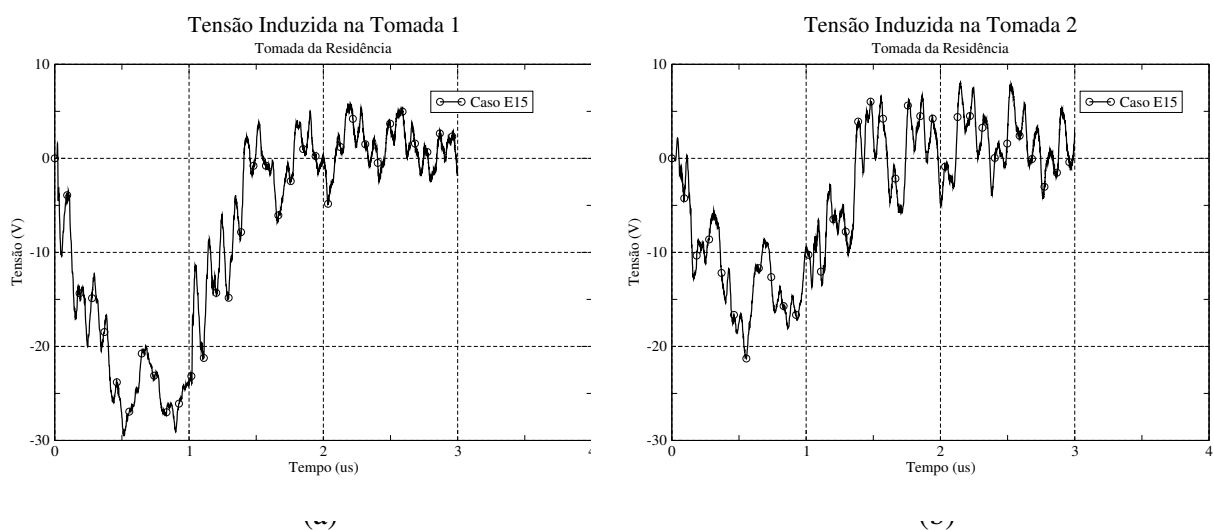


Figura E.37: Tensão induzida para (a) Tomada 1; (b) Tomada 2.

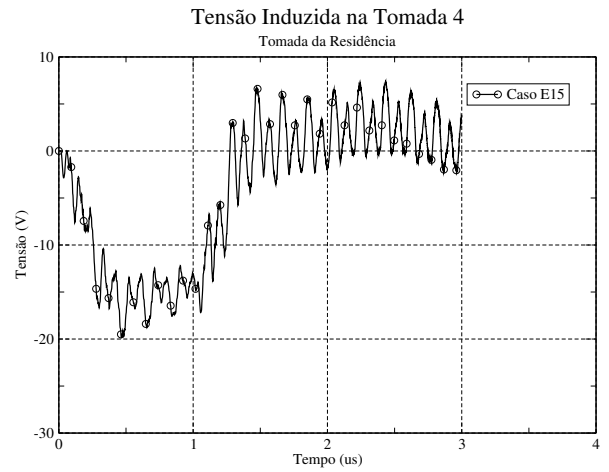
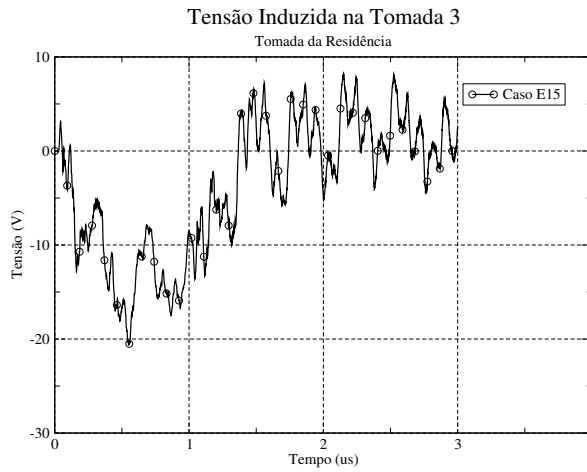


Figura E.38: Tensão induzida para (a) Tomada 3; (b) Tomada 4.

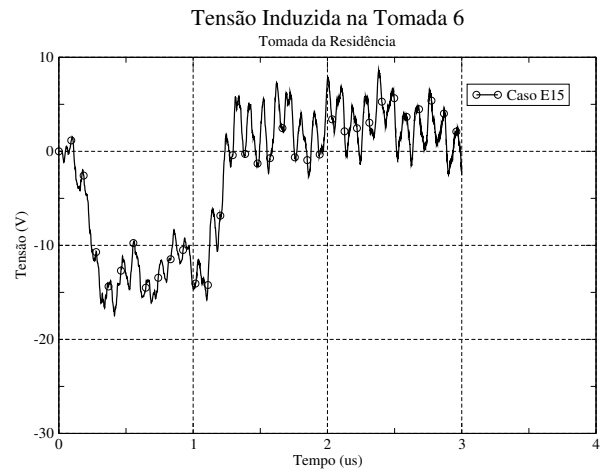
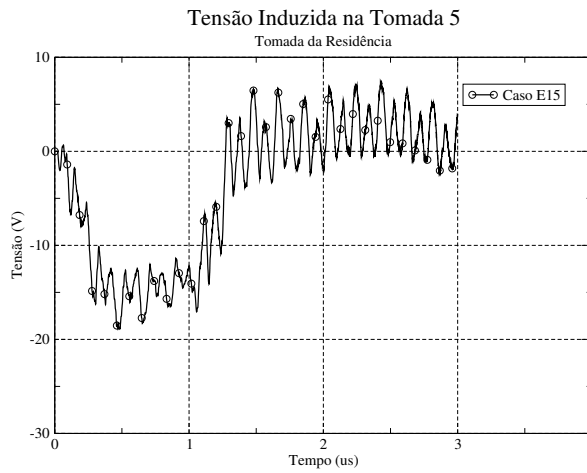


Figura E.39: Tensão induzida para (a) Tomada 5; (b) Tomada 6.

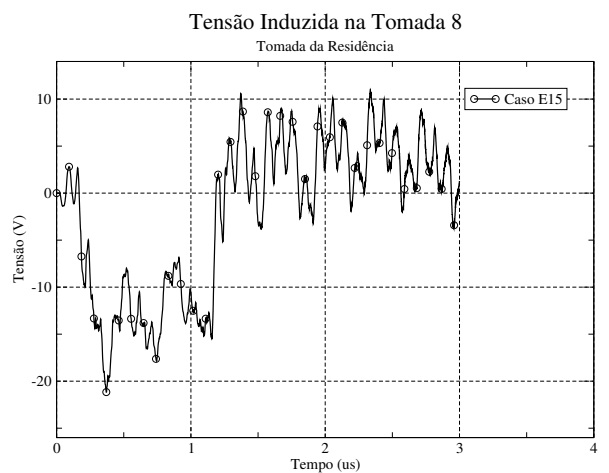
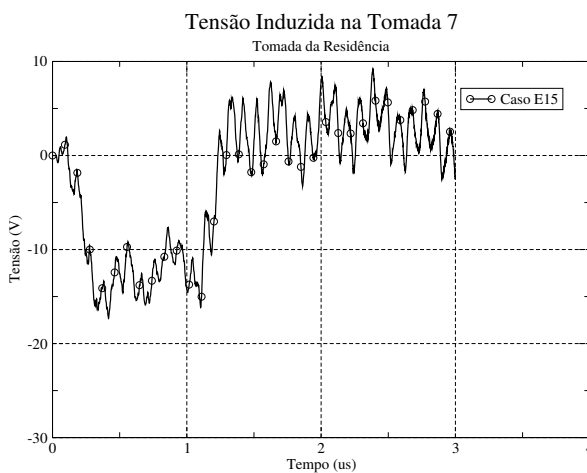


Figura E.40: Tensão induzida para (a) Tomada 7; (b) Tomada 8.

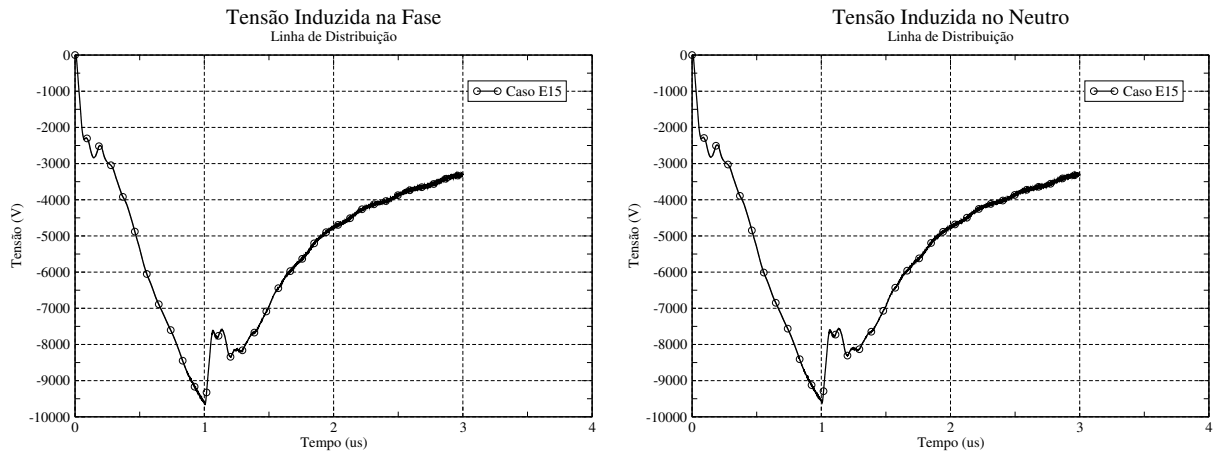
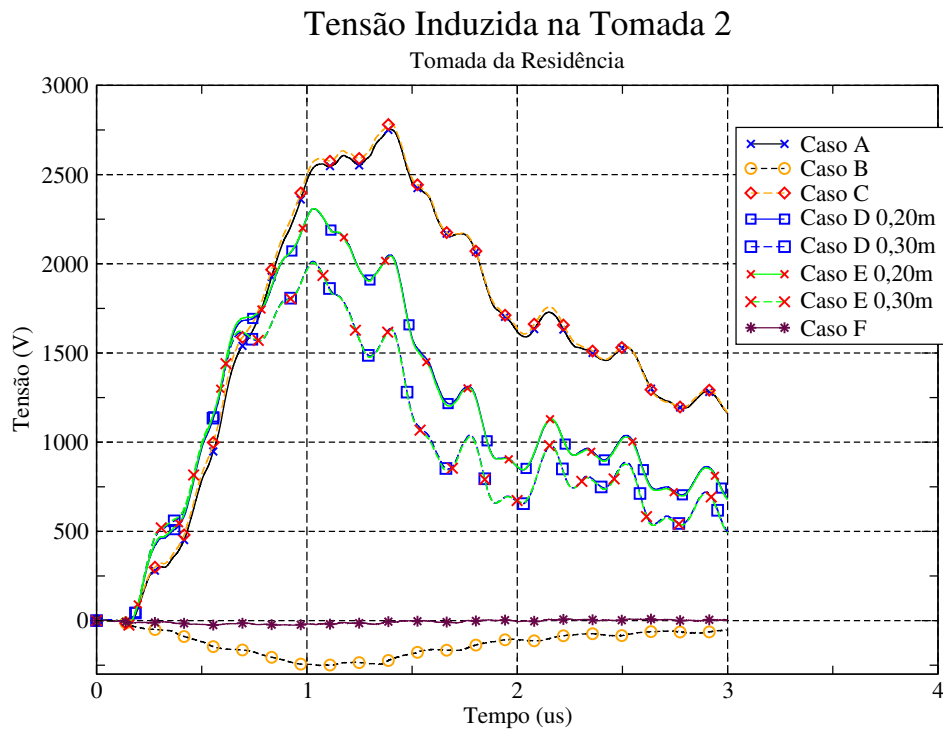


Figura E.41: Tensão induzida na linha de distribuição para (a) Fase; (b) Neutro.

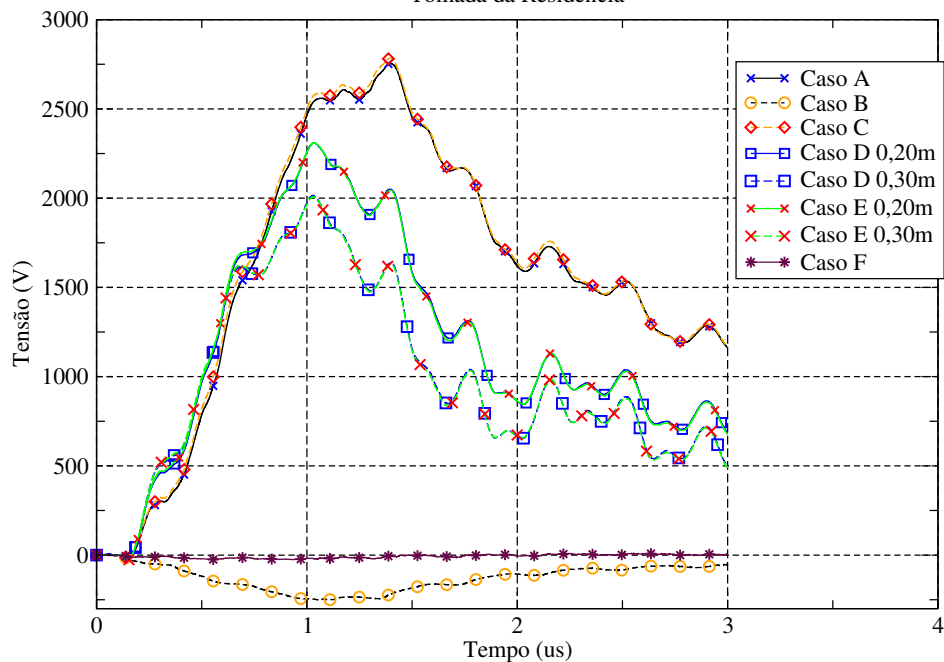
Tal como ocorreu com o caso anterior (E14), as tensões obtidas nas tomadas foram reduzidas em relação ao Caso C devido à maior proximidade entre os condutores fase e neutro. Isto demonstra a consistência física da implementação realizada neste trabalho.

### Gráficos da tensão induzida dos Casos A – F das tomadas da residência.



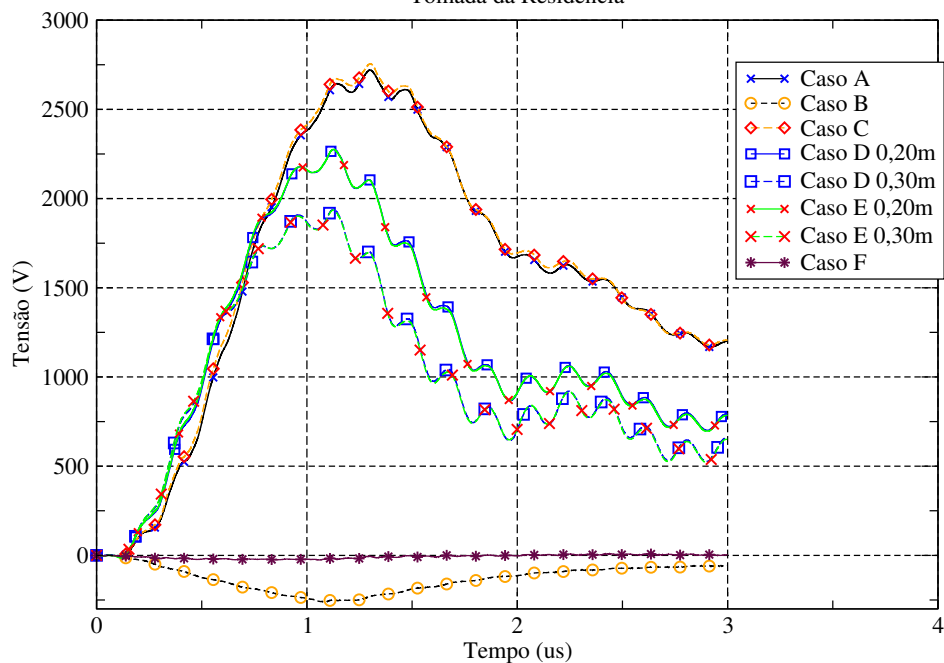
### Tensão Induzida na Tomada 3

Tomada da Residência



### Tensão Induzida na Tomada 4

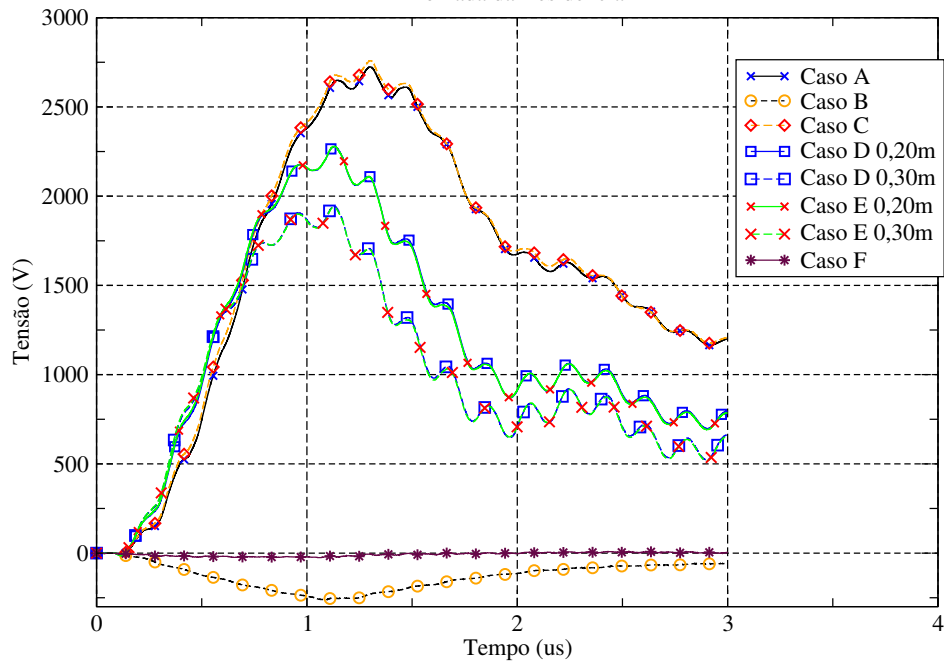
Tomada da Residência





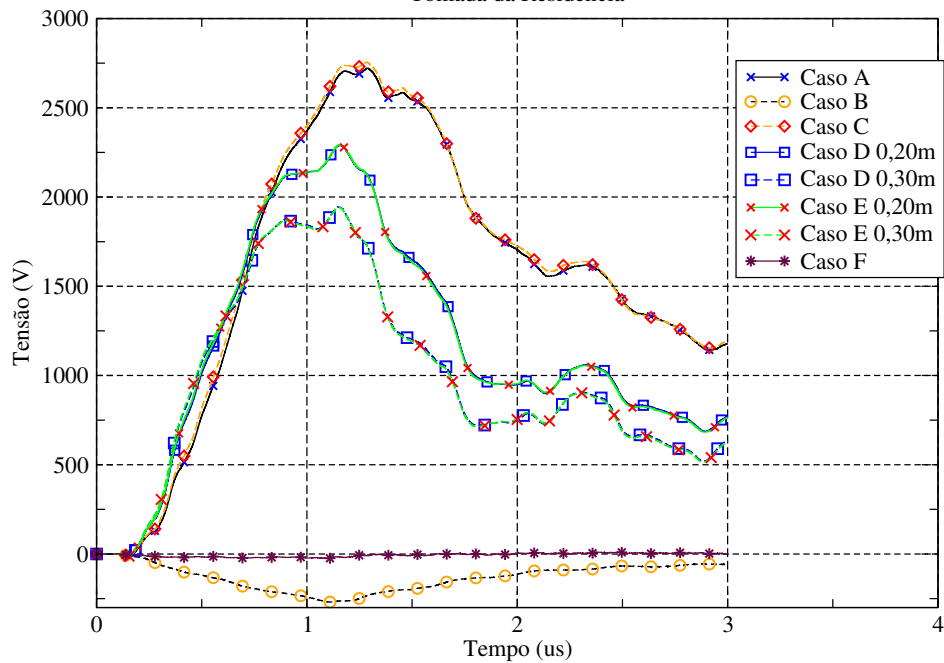
## Tensão Induzida na Tomada 5

Tomada da Residência



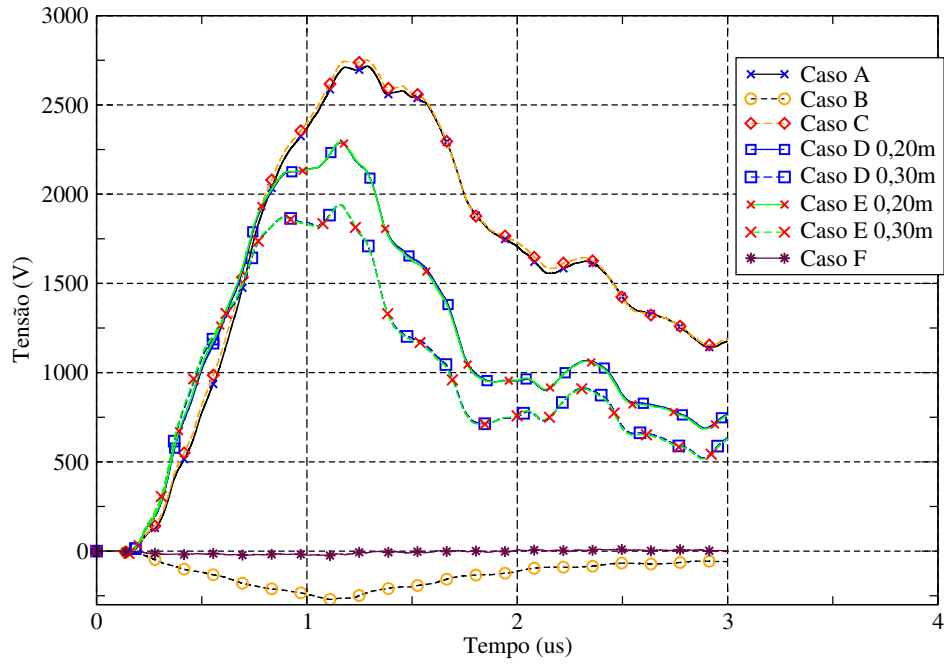
## Tensão Induzida na Tomada 6

Tomada da Residência



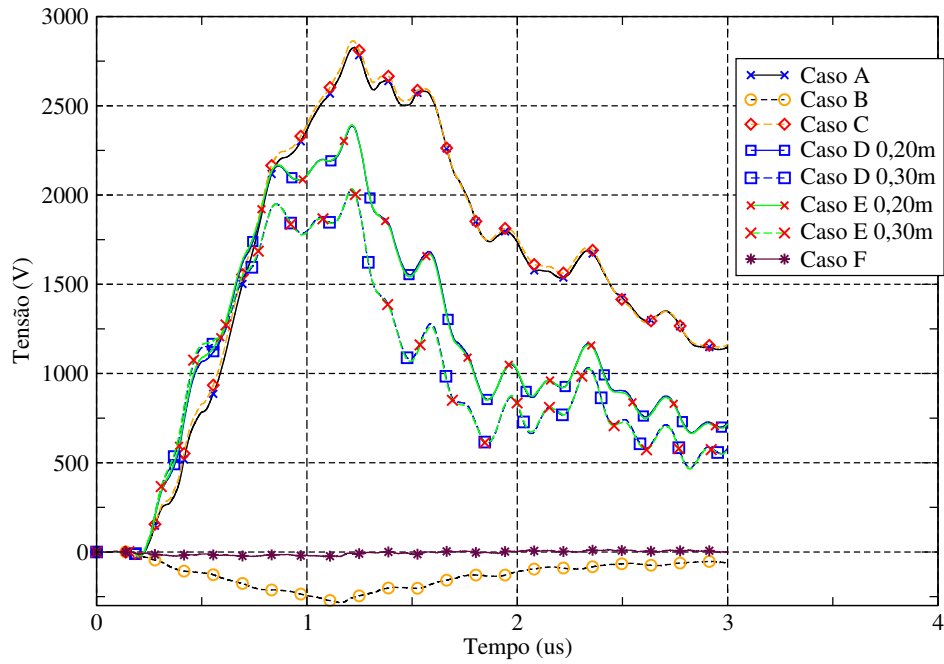
## Tensão Induzida na Tomada 7

Tomada da Residência



## Tensão Induzida na Tomada 8

Tomada da Residência



## Referências Bibliográficas

- [1] M. N. O. Sadiku, “*Numerical Techniques in Electromagnetics*”. 2nd ed., R. Nicole, J. Name Stand. Abbrev, New York: CRC Press, pp. 590 - 649, 2001.
- [2] D. R. de Melo, M. N. Kawakatsu, V. Dmitriev, K. Q. da Costa, “Aplicação do Método dos Momentos na Análise de Antenas,” XXXI CNMAC. Belém-Pará-Brasil, 2008.
- [3] A. Taflove and S. C. Hagness, “*Computational Electrodynamics, The Finite-Difference Time-Domain Method*”, 3rd ed. Artech House Inc, 2005.
- [4] C. E. R. Mercedes, V. F. R. Esquerre, A.M.F. Frasson, and H.E.H.Figueroa, “*Novel FEM Approach for the Analysis of Cylindrically Symmetric Photonic Devices*”, Journal of Lightwave Technology, vol. 27, n. 21, pp. 4717-4721, November 2009.
- [5] R. M. S. de Oliveira and C. L. S. S. Sobrinho, “*UPML Formulation for Truncating Conductive Media in Curvilinear Coordinates*”, Numerical Algorithms, vol. 46, pp. 295-319, Springer Netherlands, December 2007.
- [6] Dean, J.; Gibbs, M.R.J.; Schrefl, T.; , "Finite-element analysis on cantilever beams coated with magnetostrictive material," *Magnetics, IEEE Transactions on* , vol.42, no.2, pp. 283- 288, Feb. 2006, doi: 10.1109/TMAG.2005.861322 .
- [7] R. M. S. de Oliveira e C. L. S. S. Sobrinho, “Simulações Através do Método FDTD do Espalhamento Eletromagnético em uma Subestação de Potência Devido a Descargas Atmosféricas” , MOMAG, 2008.
- [8] K. S. Sampath, R. G. Rojas “*Application of the Helicopter Antenna Radiation Prediction Code (HARP) to Modeling Fixed Wing Aircraft*”, The Ohio State University ElectroScience Laboratory, Columbus, Ohio, 1993.

- [9] Mirian-Hosseiniabadi, S.-H.; Aghakasiri, Z.; Sadeghi, A.; Delfani, P.; Ghandehari, M.; , "Emphasizing experiences in teaching software engineering courses," *Education Technology and Computer (ICETC), 2010 2nd International Conference on* , vol.2, no., pp.V2-149-V2-153, 22-24 June 2010, doi: 10.1109/ICETC.2010.5529416.
- [10] Daboczi, T.; Kollar, I.; Simon, G.; Megyeri, T.; , "How to test graphical user interfaces," *Instrumentation & Measurement Magazine, IEEE* , vol.6, no.3, pp. 27- 33, Sept. 2003, doi: 10.1109/MIM.2003.1238336.
- [11] Pressman, R. S.; Engenharia de Software 5ª Edição. Rio de Janeiro: McGraw-Hill 2002.
- [12] Blanchette, J.; Summerfield, M; , C++ GUI Programming with Qt 3, Published Jan 15, 2004, Prentice Hall. Part of the Prentice Hall Open Source Software Development Series series.
- [13] Cohen, M.; Manssour, I. H.; OpenGL: uma abordagem prática e objetiva. São Paulo: Novatec, 2006.
- [14] R. M. S. de Oliveira, "Nova Metodologia para Análise e Síntese de Sistemas de Aterramento Complexos Utilizando o Método LN-FDTD, Computação Paralela Automática e Redes Neurais Artificiais", Tese (Doutorado em Engenharia Elétrica), ITEC - Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2008.
- [15] Johnny Marcus Gomes Rocha. CLUSTER BEOWULF: Aspectos de Projeto e Implementacao. 2003. Dissertação (Mestrado) - Programa de Pós Graduação em Engenharia Elétrica (PPGEE), ITEC, Universidade Federal do Pará, Belém.

- [16] H.P. Hofstee, "Power efficient processor architecture and the cell processor", in High-Performance Computer Architecture, 2005. HPCA-11. 11 th International Symposium on, San Francisco, pp. 258 - 262, Feb. 2005.
- [17] Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison Wesley, 2000.
- [18] Sommerville, Ian.; *Engenharia de Software 6ª Edição*. São Paulo: Addison Wesley, 2003.
- [19] T. Noda and S. Yokoyama, "Thin Wire Representation in Finite Difference Time Domain Surge Simulation," *IEEE Trans. On Power Delivery*, vol. 17, pp. 840-847, 2002.
- [20] Y. Baba, N. Nagaoka and A. Ametani, "Modeling of thin wires in a lossy medium for FDTD simulations," *IEEE Transactions on Electromagnetic Compatibility*, Vol. 47, no. 1, pp. 54-60, 2005.
- [21] <http://www.opengl.org> acessado em 25/01/2011.
- [22] K. Tanabe, "Novel method for analyzing the transient behavior of grounding systems based on the finite-difference time-domain method," in *Power Engineering Society Winter Meeting 2001, IEEE*, Columbus (OH USA) Vol. 3, 28 Jan.-1 Feb. 2001, pp.1128-1132.
- [23] Adolfo F. de O. Colares, Rodrigo Lisbôa Pereira, Rodrigo M. S. de Oliveira, Carlos Leonidas da S. S. Sobrinho , "Desenvolvimento de Interface Gráfica como Suporte para Soluções Numéricas das Equações de Maxwell" , MOMAG 2010.

- [24] Rodrigo Lisboa Pereira, Adolfo F. de O. Colares, Rodrigo M. S. de Oliveira, Carlos Leonidas da S. S. Sobrinho , "Redução do Tempo de Transferência de Dados Para o Método LN-FDTD em um Cluster Beowulf Utilizando Sockets" , MOMAG 2010.
- [25] Rodrigo Lisboa Pereira, Desenvolvimento de uma Biblioteca Eficiente Baseada em Sockets para Clusters e Cálculo de Tensões Induzidas em Linhas de Transmissão com Catenárias - 70 f. - 2011. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Elétrica, Centro Tecnológico, Universidade Federal do Pará, Belém.
- [26] Eduardo Tannus Tuma, Proposta de um Novo Modelo para Análise dos Comportamentos Transitório e Estacionário de Sistemas de Aterramento, Usando-se o Método FDTD, 2005. 143f. Tese (Doutorado em Engenharia Elétrica) - Programa de Pós Graduação em Engenharia Elétrica (PPGEE), ITEC, Universidade Federal do Pará, Belém.
- [27] Ricardo Hachem Thomé Chamié Filho, Análise de Tensão Induzida em Linhas de Distribuição de Energia Elétrica frente a uma Descarga Atmosférica - 103 f. - 2009. Dissertação (Mestrado) – Programa de Pós Graduação em Engenharia Elétrica (PPGEE), ITEC, Universidade Federal do Pará, Belém.
- [28] E. T. Tuma, “Proposta de um Novo Modelo para Análise dos Comportamentos Transitório e Estacionário de Sistemas de Aterramento, Usando-se o Método FDTD”, Tese de Doutorado– Programa de Pós Graduação em Engenharia Elétrica (PPGEE), ITEC, Universidade Federal do Pará, Belém, 2005.
- [29] Associação Brasileira de Normas Técnicas, *NBR 14136: Plugues e tomadas para uso doméstico e análogo até 20 A/250 V em corrente alternada – Padronização*, ABNT, Nov. 2002.
- [30] J. Mamede Filho, *Instalações Elétricas Industriais*. 7ª Edição, Editora LTC, 2007.

- [31] S. Visacro Filho, *Aterramentos Elétricos*. Editora Artliber, 2005.
- [32] F. H. Silveira, “Modelagem para Cálculo de Tensões Induzidas por Descargas Atmosféricas.”, Tese de Doutorado, PPGEE/UFMG, 2006.
- [33] Ricardo H. T. Chamié Filho, Rodrigo M. S. de Oliveira, Carlos Leonidas da S. S. Sobrinho , "Simulations of Lightning Strokes near Transmission Lines in Urban Environments by Using the Finite-Difference Time-Domain Method" , *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, Vol. 8, No. 1, pp. 114S-121S, June 2009.
- [34] G. F. Simmons, *Cálculo com geometria analítica*, 1ª edição. São Paulo: McGraw-Hill Ltda, 1987.
- [35] D. O. Ehrenburg, “Transmission Line Catenary Calculations,” *Transactions of the American Institute of Electrical Engineers*, vol. 54, pp. 719-728, 1935.