Pontifícia Universidade Católica
do Rio de Janeiro

Luis Alberto Rabanal Ramirez

Aplicação do Método das Diferenças Finitas no Dominio do Tempo na Análise de Cobertura em Ambientes Interiores

Tese de Doutorado

Tese apresentada ao Programa de Pós–graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica da PUC–Rio como requisito parcial para obtenção Do título de Doutor em Engenharia Elétrica

Orientador: Prof. Flavio José Vieira Hasselmann

Rio de Janeiro
Agosto de 2008

PUC-Rio - Certificação Digital Nº 0310507/CB

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Pontifícia Universidade Católica
do Rio de Janeiro

Luis Alberto Rabanal Ramirez

Aplicação do Método das Diferenças Finitas no Dominio do Tempo na Análise de Cobertura em Ambientes Interiores

Tese apresentada ao Programa de Pós–graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica do Centro Técnico Científico da PUC–Rio como requisito parcial para obtenção Do título de Doutor em Engenharia Elétrica. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Flavio José Vieira Hasselmann
Orientador
Departamento de Engenharia Elétrica — PUC–Rio

Prof. Gervásio Protásio dos Santos Cavalcante
Departamento de Engenharia Elétrica–UFPA–Belém

Prof. Erasmus Couto Brazil de Miranda
Departamento de Engenharia Elétrica–UCP–Petrópolis

Prof. Luiz Costa da Silva
Centro de Estudos em Telecomunicações–PUC–Rio

Prof. Luiz Alencar Reis da Silva Mello
Centro de Estudos em Telecomunicações–PUC–Rio–PUC–Rio

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 29 de Agosto de 2008

PUC-Rio - Certificação Digital Nº 0310507/CB

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Luis Alberto Rabanal Ramirez

Possui graduação em Física pela Universidad Nacional de Trujillo, Peru (1998), e mestrado em Engenheria Elétrica na área de concentração em Eletromagnetismo Aplicado pela Pontifícia Universidade Católica do Rio de Janeiro (2004). Recebeu bolsa de estudos da Fundação CAPES.

PUC-Rio - Certificação Digital Nº 0310507/CB

Ficha Catalográfica

Rabanal Ramirez, Luis Alberto

Aplicação do método das diferenças finitas no dominio do tempo na análise de cobertura em ambientes interiores / Luis Alberto Rabanal Ramirez; orientador: Flavio José Vieira Hasselmann. — 2008.

251 f.: Il. ; 30 cm

Tese (Doutorado em Engenharia Elétrica) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2008.

Inclui bibliografia.

1. Engenharia Elétrica – Teses. 2. Cobertura em interiores. 3. Métodos deterministicos. 4. Traçado de raios. 5. Diferencias finitas no dominio do Tempo. I. Hasselmann, Flavio José Vieira. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.13

Agradecimentos

PUC-Rio - Certificação Digital Nº 0310507/CB

Ao meu orientador Professor Flavio José Vieira Hasselmann pelo apoio, e incentivo para a realização deste trabalho.

Á CAPES e à PUC–Rio, pelos auxílios concedidos.

A toda minha familia em especial a meus irmãos José, Misael, Aldo.

Aos meus colegas e amigos do CETUC (Fabricio, Marco Aurelio, Pedro, Maiquel, Sandro, André, Rodrigo, Marcio, Fabio, Claudio, Kathy, Marcela, Janaina, Jõao)

Aos professores: Luiz Costa da Silva, José Bergmann, Luiz Silva Mello, Emmanoel Costa, Marco Antonio Grivet.

Ao pessoal do CETUC pela ajuda, em especial á Maria Lucia.

PUC-Rio - Certificação Digital Nº 0310507/CB

Resumo

Ramirez, Luis Alberto Rabanal; Hasselmann, Flavio José Vieira. **Aplicação do Método das Diferenças Finitas no Dominio do Tempo na Análise de Cobertura em Ambientes Interiores**. Rio de Janeiro, 2008. 251p. Tese de Doutorado — Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

O interesse crescente em redes locais sem fio de banda larga e altas taxas de transmissão requer o modelamento cada vez mais fidedigno do canal de radiopropagação, em particular em ambientes interiores. A utilização de métodos óticos de rastreamento do campo eletromagnético ao longo de trajetórias (raios) entre transmissor e observador, via seu espalhamento nos constituintes morfológicos (obstáculos) do ambiente, como a UTD implementada em estudos anteriores no CETUC, é adequada para a análise de cobertura desses sistemas em (e através de) corredores e ambientes sem se levar em conta outros constituintes como mobliliário. Sendo estes últimos de dimensões da ordem dos comprimentos de onda envolvidos e/ou dispostos em quantidade e localização de forma a tornar impraticável a implementação completa de um traçado de raios abrangente, torna-se necessário o emprego de um método híbrido de análise como a implementação do casamento do campo rastreado oticamente (via UTD) até a penetração em um ambiente, com seu espalhamento por objetos interiores conforme determinado pelo método (numérico) de Diferenças Finitas no Domínio do Tempo (FDTD), o que foi motivo de análise, em trabalhos anteriores, em cenários bi-dimensionais. Visando fornecer subsídios à extensão futura desta técnica híbrida a problemas tri-dimensionais vetoriais, este trabalho explora a aplicação do método FDTD na análise de cobertura de ambientes interiores. Em particular, após uma revisão detalhada do método e de sua implementação em conjunto com Camadas Perfeitamente Casadas (PML), é explorada sua aplicação à análise de cobertura em ambientes típicos, porém sem mobiliário, e resultados são comparados, no domínio do tempo e da freqüência, mediante inversão por Fourier, com aqueles obtidos por rastreamento de raios segundo a UTD, validando os códigos desenvolvidos; a formulação desta última, aqui utilizada, é também revista e listada. Concluindo, a inserção de obstáculo simples (3-Dim) no cenário anterior, ilustra a adequação do método FDTD para levar em conta a presença de mobiliário na análise de cobertura de ambientes de interesse.

Palavras–chave

Cobertura em interiores. Métodos deterministicos. Traçado de raios. Diferencias finitas no dominio do Tempo.

Abstract

Ramirez, Luis Alberto Rabanal; Hasselmann, Flavio José Vieira. **Application of Finite Difference Time Domain in the Analysis of Indoor Coverage**. Rio de Janeiro, 2008. 251p. PhD Thesis — Department of Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

The growing interest in broadband wireless local networks and high transmission rates requires an increasingly reliable modeling of radio propagation channels, specially for indoor scenarios. The use of optical methods of electromagnetic field tracking along (ray) trajectories between transmission and observation points, via high-frequency analysis of scattering by morphological constituents (obstacles) of the environment, such as UTD, fits well into the coverage analysis of corridors and empty rooms. While usual furniture dimensions are of the order of working wavelengths and disposed such as to render intractable the implementation of a comprehensive ray tracing mechanism, the need arises for a hybrid method of analysis that allows for the matching of an UTD description of EM field tracking up to its penetration in a certain environment with the indoor scattering - by furniture and alike - analysis as carried out by the (numerical) FDTD method. This has been reported in the literature in connection with canonical two-dimensional scenarios. In order to construct the building blocks of the extension of such a hybrid technique in order to encompass three-dimensional vector problems as well, the present work addresses the application of FDTD to indoor coverage analysis. Starting with an in-depth review of the method and of its implementation with companion Perfect Matching Layers (PML), its application to coverage analysis of typical (empty) indoor scenarios is explored and results are compared, both in time and frequency (via appropriate Fourier inversions) domains, with those obtained via ray tracing, so as to validate developed numerical codes; a comprehensive review of UTD aspects and formulation is also listed herein. To conclude, a simple 3-D obstacle is inserted in previous scenarios thereby illustrating the adequacy of FDTD to consider the presence of furniture in the coverage analysis of environments of practical interest.

Keywords

Indoor coverage. Determinist methods. Ray tracing. Finite difference time domain.

PUC-Rio - Certificação Digital Nº 0310507/CB

Sumário

1 Introdução **14**

2 Revisão Bibliográfica: Modelos Para Cálculos De Cobertura Em Ambientes Interiores. **17**
2.1 Métodos estatísticos para cobertura em ambientes interiores. 17
2.2 Métodos determinísticos para cobertura interiores 20

3 Método De Diferenças Finitas No Domínio Do Tempo (FDTD) **23**
3.1 Visão Geral 23
3.2 Formulação do método FDTD 26
3.3 Algoritmo de Yee - Consistência, convergência e estabilidade 31
3.4 Algoritmo de Yee - Memória e tempo de CPU 32
3.5 Método FDTD - Condições Absorventes 35
3.6 Camada perfeitamente Casada (PML) 35
3.7 Camada Uniaxial Perfeitamente Casada - UPML 41
3.8 Fontes de excitação 52

4 Métodos Assintóticos **74**
4.1 A Ótica Geométrica 74
4.2 Refração 85
4.3 Campo Difratado 89

5 Método Híbrido **97**
5.1 Modelo de Ying Wang 97
5.2 Modelo de Sebastien Reynaud 101
5.3 Métodologia proposta para um novo modelo 104

6 Resultados **108**
6.1 Cálculo da Cobertura numa WLAN usando FDTD [54] 108
6.2 Emprego do método FDTD para análise de cobertura em ambientes interiores [55] 113
6.3 Modelagem de redes sem fio em ambientes interiores [56] 118
6.4 Resultados com inserção de mobiliário simples 122

7 Conclusoes e Sugestões **130**
7.1 Sugestões para trabalhos futuros 132

A Aspectos do lançamento e captação de raios **141**
A.1 Antena Receptora (Rx) 141
A.2 Antena Transmissora (Tx)[3] 142

B Códigos corresponde aos programas **146**
B.1 Códigos correspondentes ao método de traçado de raios 146
B.2 Códigos correspondentes ao método FDTD 230

PUC-Rio - Certificação Digital Nº 0310507/CB

Lista de figuras

1.1 Taxa de transmissão versus mobilidade [1] 14

3.1 Dominio Computacional 24
3.2 Célula unitária da malha de Yee 27
3.3 Distribuição das componentes de $\vec{E}$ e $\vec{H}$ no domínio computacional. 27
3.4 Interação de pulso gaussiano com um bloco metálico 34
3.5 Pulso gaussiano propagando-se no domínio e interagindo com a PML (a) passo temporal 90 (b) passo temporal 140 36
3.6 Distribuição das condutividades - Camada PML- 2D 39
3.7 Pulso gaussiano interagindo com o bloco PEC e com as fronteiras do domínio computacional a) sem PML, b) com PML. 39
3.8 PML 3D - especificação de algumas condutividades 42
3.9 Pulso Gaussiano propagando-se no dominio computacional 3D rodeado por UPML 51
3.10 Pulso Gaussiano propagando-se no dominio computacional 3D rodeado por UPML, ambiente sem portas 51
3.11 (a)Pulso Gaussiano com componente dc (b) Pulso duplo-gaussiano sem componente DC 53
3.12 Divisão do domínio computacional em regiões: campo total / campo espalhado 54
3.13 Onda plana propagando-se na região de campo total (a) ângulo de incidência 90°, (b) ângulo de incidência 0° 55
3.14 Condições de conexão típicas entre as regiões. Polarização TMz. 55
3.15 Aplicação da técnica TFSF numa malha unidimensional e a distribuição dos componentes de campo. 56
3.16 Detalhes da interface campo total campo espalhado. 58
3.17 Cálculo eficiente do campo incidente para polarização TMz-2D 60
3.18 Propagação da onda incidente: direção e polarização 61
3.19 Interface TFSF-3D 62
3.20 Componentes $Ex(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $j = j_0$ no plano XZ. da figura 3.19 62
3.21 Componentes $Ex(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $j = j1$ no plano XZ. da figura 3.19 63
3.22 Componentes $Ex(\rightarrow)$ e $Ey(\uparrow)$ sôbre a face $k = k_0$ no plano XZ. da figura 3.19 64
3.23 Componentes $Ey(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $k = i_0$ no plano YZ. da figura 3.19 65
3.24 Componentes $Ey(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $i = i_1$ no plano YZ. da figura 3.19 66
3.25 Onda plana propagando-se na região de campo total, ângulo de incidência: a) 50° e b)75° 71
3.26 Onda plana propagando-se na região de campo total, ângulo de incidência a) 125° e b) 237°. 71
3.27 Onda plana propagando-se na região de campo total, ângulo de incidência a) 300° e b) 345°. 71

PUC-Rio - Certificação Digital Nº 0310507/CB

3.28 Onda plana propagando-se na região de campo total ângulo de incidência 0° . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$. 72
3.29 Onda plana propagando-se na região de campo total ângulo de incidência 0° . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$. 72
3.30 Onda plana propagando-se na região de campo total ângulo de incidência 0° . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$. 72
3.31 Onda plana, ângulos de incidência 130° e 0°, para diferentes passos. Interagindo com uma lámina quadrada. 73
3.32 Onda plana, ângulos de incidência 130° e 0°, para diferentes passos. Interagindo com uma lámina quadrada. 73

4.1 Tubo de raios em um meio não homogêneo. 74
4.2 Relação entre os raios e as frentes de onda. 75
4.3 Tubo de raios astigmático. 76
4.4 Vista 2D de cenário de 20x20x6 m, com um transmissor na posição (-10,0,3), mostrando os raios em visada direta (verde) sem atingir as paredes do corredor, os raios (azul) que originaram raios refletidos, transmitidos e difratados. . 77
4.5 Vista 2D de cenário de 20x20x6 m, com um transmissor na posição (-5,0,2), mostrando os raios em visada direta (azul) sem atingir as paredes do corredor, os raios (vermelho) que originam raios refletidos, transmitidos e difratados. 77
4.6 Sistema de coordenadas esféricas associado à antena transmissora. 78
4.7 Vista 2D de cenário 20x20x6 m com antena na posição (-9, 0,2): primeira reflexão nos prédios representada pelos raios em preto, primeira reflexão nos prédios, mas saindo do cenário em vermelho; raios sem atingir os prédios, saindo do cenário, em azul. 79
4.8 Sistemas de coordenadas fixos aos raios para a reflexão. 79
4.9 Esquema de refração: meios 1 e 3 iguais. 82
4.10 Coeficente de reflexão de Fresnel soft para paredes com permissividade relativa (4.0-0.1j) e espessuras 30 cm e infinito, em função do angulo de incidencia e para a frequencia de 4 GHz. 84
4.11 Coeficente de reflexão de Fresnel hard para paredes com permissividade relativa (4.0-0.1j) e espessuras 30 cm e infinito, em função do angulo de incidencia e para a frequencia de 4 GHz. 84
4.12 Esquema de reflexão e refração dos raios na interface entre dois meios diferentes. 87
4.13 Raios difratados nas quinas dos predios com Tx em (0,0,2). 89
4.14 Raio incidente sôbre uma aresta e divisão da área ao redor em três regiões distintas. 90
4.15 Cone de difração e sistemas de coordenadas fixos aos raios para a difração. 91
4.16 Vista do plano normal à aresta; indice "proj" refere-se à projeção no plano da vista. 93
4.17 Variação de $N^{\pm}$ com linhas partilhadas às fronteiras de transição 96

PUC-Rio - Certificação Digital Nº 0310507/CB

5.1 Cenário interior com áreas de interesse fechadas por caixas virtuais $A_1B_1C_1D_1$ e $A_2B_2C_2D_2$. Diferentes sombras e texturas são usadas para representar diferentes materiais [34] 99
5.2 Redimensionamento do retângulo $A_1B_1C_1D_1$ da figura (5.1), ilustrando a interface entre os métodos RT/FDTD [34] 100
5.3 Esquema da fonte de excitação (adaptada de [34]) 101
5.4 Divisão do dominio computacional em regiões para aplicação do método 102
5.5 Principio de cálculo da matriz de espalhamento [4] 103
5.6 Subdomínios RT/FDTD 105
5.7 Detalhe do subdomínio FDTD - Caixa $A_1B_1C_1D_1$ 105
5.8 Fontes externas que contribuem para o campo no subdomínio FDTD 106
5.9 Fontes externas que contribuem para o campo no subdomínio FDTD 107
5.10 Principio de Equivalência [53] 107

6.1 Cenário em ambiente interno (15x15m), com paredes divisórias em preto e portas em verde. 109
6.2 Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes e portas PEC, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns.$ 110
6.3 Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes PEC e sem portas, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns.$ 110
6.4 Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes PEC e sem portas, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns.$ 111
6.5 Distribução das antenas no cenário considerado 111
6.6 Pôtencia normalizada recebida em (a)Rx1, (b)Rx2, (c)Rx3, (d)Rx4 112
6.7 cenário em ambiente interior $(15x15x3\ m)$, com paredes divisórias em azul e portas em vermelho 114
6.8 Dipolo usado no programa (a) modelagem, (b) radiação no plano vertical, (c) radiação espacial. 114
6.9 Pulso gaussiano utilizado no gap (a) dominio do tempo (b) espectro de freqüência. 115
6.10 Cobertura pela componente Ez de ambiente com paredes e portas PEC: (a) primeiros instantes $13,85ns$ da propagação da energia, (b) aos $27,13ns$ a energia atinge as camadas UPML sendo absorvidas. 116
6.11 Cobertura pela componente Ez de ambiente com paredes de concreto e portas de madeira: (a) decorridos $13,85ns$, (b) decorridos $27,13ns.$ 116
6.12 Distribução das antenas no ambiente considerado. 117
6.13 Potência recebida normalizada em Rx1. 118
6.14 Potência recebida normalizada em Rx2. 119
6.15 Potência recebida normalizada em Rx3. 120
6.16 Potência recebida normalizada em Rx4. 121
6.17 Comparação das potências recebidas em $Rx1$ e $Rx4$, normalizadas com relação ao máximo em Rx1. 122
6.18 Pulso gausiano $(6\text{-}3)$ para excitação de uma antena dipolo, (a) no dominio do tempo, (b) espectro de freqüência. 123

PUC-Rio - Certificação Digital Nº 0310507/CB

6.19 Comparação de potência recebida em Rx1, normalizada com relação ao máximo, calculada por RT e FDTD. 124
6.20 Comparação de potência recebida em Rx2, normalizada com relação ao máximo, calculada por RT e FDTD. 124
6.21 Comparação de potência recebida em Rx3, normalizada com relação ao máximo, calculada por RT e FDTD. 125
6.22 Comparação de potência recebida em Rx4, normalizada com relação ao máximo, calculada por RT e FDTD. 125
6.23 Comparação da variação com a freqüência da potência recebida em $Rx1$ e calculada usando RT/UTD e FDTD. 126
6.24 Comparação da variação com a freqüência da potência recebida em $Rx2$ e calculada usando RT/UTD e FDTD. 126
6.25 Comparação da variação com a freqüência da potência recebida em $Rx3$ e calculada usando RT/UTD e FDTD. 127
6.26 Comparação da variação com a freqüência da potência recebida em $Rx4$ e calculada usando RT/UTD e FDTD. 127
6.27 Cenário com uma distribução de antenas e um bloco de madeira de $(2x2x1\ m)$ no interior 127
6.28 (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx1$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $6,5\ ns$. 128
6.29 (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx2$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $13,2\ ns$. 128
6.30 (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx3$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $19,5\ ns$. 129
6.31 (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx4$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $26,2\ ns$. 129

A.1 Problema de dupla contagem na esfera de recepção [70] 141
A.2 Problema de dupla contagem na esfera de recepção [70] 142
A.3 Problema de dupla contagem na esfera de recepção 142
A.4 Icosaedro formado por 20 triângulos 143
A.5 Grade esférica com espaçamento angular uniforme 144
A.6 (a) Um poliedro de 20 lados (b) uma extrapolação da superfície da esfera, (c) vértices de lançamento geocêntrico. 144
A.7 Face com diferentes frequências de subdivisão : (a) N=2, (b) N=3, (c) N=4, (d) face extrapolada. 144
A.8 Sistema de Coordenadas triangular $(N = 5)$ 145

PUC-Rio - Certificação Digital Nº 0310507/CB

Lista de tabelas

2.1 Valores para o índice de desvanecimento. (Método 1SM) 18
2.2 Lw_i para tipos diferentes de paredes e diversas frequências. 19
2.3 Alguns valores das perdas de propagação Lw_i para 2 tipos diferentes de paredes usados na equação (2-4) 20

4.1 Regiões definidas pelas fronteiras de sombra 90

6.1 Parâmetros característicos dos materiais considerados nas simulações [57] 109

A.1 Atributos de uma esfera geodésica em função da freqüência de subdivisão N. 145

PUC-Rio - Certificação Digital Nº 0310507/CB

PUC-Rio - Certificação Digital Nº 0310507/CB

Dedicatoria:

A toda minha familia, por toda a ajuda, compresão e carinho em todos os momentos,
Luis Alberto.

1
Introdução

A cada dia que passa os serviços oferecidos pelos sistemas sem fio requerem cada vez maior largura de banda. Nos sistemas das próximas geraçẽs estes requerimentos serão ainda mais exigentes como é ilustrado na figura (1.1) Adicionalmente os futuros usuários serão caracterizados por serem ubíquos (em um grau maior que os atuais) e desejaram dispor destes serviços em diferentes lugares com alta qualidade. Esta situação gera um interesse crescente pela criação de aplicações que permitam modelar de forma mais fidedigna o canal de radiopropagação, visando a implementação de diferentes sistemas sem fio.

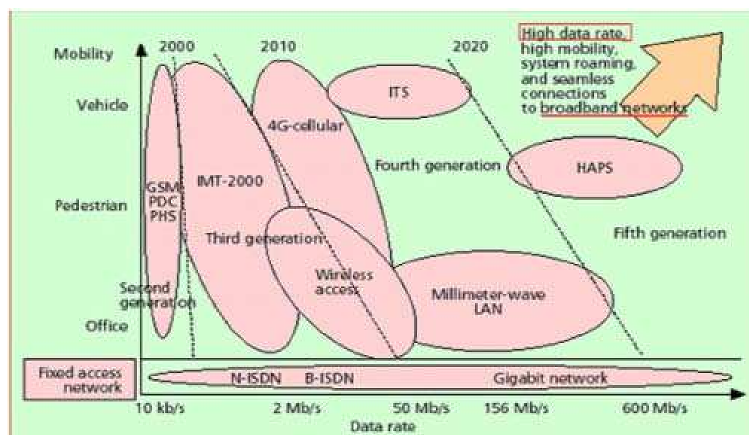

Figura 1.1: Taxa de transmissão versus mobilidade [1]

Existe um grande interesse na caracterização de sistemas sem fio em ambientes interiores. A predição da perda por múlti-percurso, assim como o cálculo das interferências é de vital importáncia para um planejamento otimizado.

Cenários interiores apresentam características singulares, como alta densidade de tráfego, cobertura definida pela forma e quantidade de objetos nos ambientes, qualidade de serviço afeitada pela presença de mobília e pessoas, etc., razões pelas quais os modelos usados em outros tipos de cenários não podem ser aplicados, necessitando-se de modelos específicos.

Dentro dos vários tipos de modelos usados nestes cenários, cabe destacar a existência dos modelos semi-empíricos que combinam técnicas de métodos estatísticos e métodos determinísticos.

PUC-Rio - Certificação Digital Nº 0310507/CB

A utilização de métodos óticos de rastreamento do campo eletromagnético ao longo de trajetórias (raios) entre transmissor e observador, via seu espalhamento nos constituintes morfológicos (obstáculos) do ambiente, como a Teoria Uniforme da Difração (UTD) implementada em estudos anteriores [2, 3] , é adequada para a análise de cobertura desses sistemas em (e através de) corredores e ambientes sem levar em conta outros constituintes como mobiliário. Sendo estes últimos de dimensões da ordem dos comprimentos de onda envolvidos e/ou dispostos em quantidade e localização de forma a tornar impraticável a implementação completa de um traçado de raios abrangente, torna-se necessário o emprego de um método híbrido de análise como a implementação do casamento do campo rastreado oticamente (via UTD) até a penetração em um ambiente, com seu espalhamento por objetos interiores conforme determinado pelo método (numérico) de Diferenças Finitas no Domínio do Tempo (FDTD), o que foi motivo de análise, em trabalhos anteriores, em cenários bi-dimensionais [4, 5].

Visando fornecer subsídios à extensão futura desta técnica híbrida a problemas tri-dimensionais vetoriais, este trabalho explora a aplicação do método FDTD na análise de cobertura de ambientes interiores. Em particular, após uma revisão detalhada do método e de sua implementação em conjunto com Camadas Perfeitamente Casadas (PML) é explorada sua aplicação à análise de cobertura em ambientes típicos, porém sem mobiliário, e resultados são comparados, no domínio do tempo e da freqüência, mediante inversão por Fourier, com aqueles obtidos por rastreamento de raios segundo a UTD, validando os códigos desenvolvidos; a formulação desta última, aqui utilizada [3] é também revista e adaptada. Concluindo, a inserção de obstáculo simples (3-D) no cenário anterior, ilustra a adequação do método FDTD para levar em conta a presença de mobiliário na análise de cobertura de ambientes de interesse.

O trabalho foi organizado da seguinte forma: No capitulo 2 é apresentada uma revisão bibliográfica dos diferentes métodos existentes para cálculos de cobertura em ambientes interiores, destacando as vantagens e desvantagens de cada um deles. Devido às diferentes características dos métodos (Traçado de Raios, FDTD), estes são apresentados individualmente em capítulos separados. No capitulo 3 é apresentado detalhadamente o método numérico de Diferenças Finitas no Domínio do Tempo (FDTD) e, no capitulo 4, se apresenta um resumo do método determinístico Traçado de Raios (RT) cuja informação abrangente pode ser encontrada em [2]. No Capitulo 5 é apresentada uma revisão bibliográfica dos métodos híbridos existentes usados em ambientes interiores. No Capitulo 6 são apresentados resultados comparativos dos métodos FDTD

PUC-Rio - Certificação Digital Nº 0310507/CB

e RT/UTD refrentes à sua aplicação no cálculo de cobertura de ambientes interiores vazios, tanto no dominio do tempo como no dominio da frequencia; exemplos de implementação do método FDTD com a presença de mobiliario simples são também apresentados. O capitulo 7 são apresentadas as conclusões do trabalho e sugestões para desenvolvimentos futuros. No capitulo 8 são listadas as referências bibliográficas e no Apêndice A, aspectos de lançamento e captação de raios, enquanto, no Apêndice B, são apresentadas as listagens dos programas de computador implementados neste trabalho.

PUC-Rio - Certificação Digital Nº 0310507/CB

2
Revisão Bibliográfica: Modelos Para Cálculos De Cobertura Em Ambientes Interiores.

Os ambientes interiores são cenários considerados de maior complexidade com relação a ambientes exteriores, devido às características singulares que apresentam, como a presença de um número maior de objetos espalhadores. Frequentemente de tamanhos e materiais diferentes, influenciando fortemente na propagação entre o transmissor e receptor.

Para o cálculo de sua cobertura, os métodos usados são classificados em dois grandes grupos: métodos estatísticos e métodos determinísticos. Há também métodos que combinam estes dois enfoques, classificados como semi-determinísticos [6] e semi-empíricos [7]. No grupo de métodos determinísticos, há ainda aqueles que usam duas técnicas (suas melhores características) de forma a potencializar sua eficiência no cálculo de cobertura em interiores [8, 5, 9, 10]. Neste último grupo, encontra-se o modelo explorado neste trabalho.

2.1
Métodos estatísticos para cobertura em ambientes interiores.

Os métodos estatísticos são baseados em campanhas de medidas e têm a vantagem de ser simples e computacionalmente eficientes para ambientes interiores. No entanto, como as partes internas dos edifícios diferem em tamanho, forma e tipo de materiais, os parâmetros dos métodos estatísticos variam fortemente de ambiente para ambiente, sendo difícil estabelecer um modelo padrão.

Os principais modelos estatísticos para cálculo de cobertura em interiores são: modelo "One Slope" (1SM)[11], COST 231-modelo Motley-Keenan [12], modelo Rappaport (modelo Motley-Keenan-modificado) [13], modelo Valenzuela [14], COST 231-Multi Parede/Andar (MWM) [15] e Modelo de atenuação linear (LAM) [7], cujas características princípais serão sumarizadas a seguir.

PUC-Rio - Certificação Digital Nº 0310507/CB

2.1.1
Modelo "one-slope"

O modelo one-slope [11] considera uma dependência linear entre as perdas de propagação no percurso, em dB, e o logaritmo da distancia(d):

$$L = L_0 \ n \ log(d) \qquad [dB] \tag{2-1}$$

é o modelo mais simples e fácil de usar, já que só a distancia entre o transmissor e receptor aparece como parâmetro de entrada. L_0 em (2-1) representa uma perda-referência, por exemplo, a um metro de distancia do transmissor e "n" denota um índice de desvanecimento. Alguns valores para "n" são listados na Tabela 2.1

Cenário	n	Mecanismos de propagação dominantes
Corredor	1.4 - 1.9	onda guiada
Quartos grandes e vazios	2	visada direta (LOS)
Quartos mobiliados	3	LOS + Multipercurso
Quartos densamente mobiliados	4	Espalhamento por obstáculos, sem Linha de visada direta
Multi-andar	5	Atenuação parede, chão

Tabela 2.1: Valores para o índice de desvanecimento. (Método 1SM)

Como desvantagen, requer cuidado na escolha do tipo de parede relacionado com o expoente "n", de forma a minimizar o desvio padrão. Gera grandes erros quando é usado para cálculo das perdas entre vários andares de um edifício, sendo necessária a inclusão de um fator de correção da perda no chão, incluído em [16].

2.1.2
Modelo "Motley-Keenan"

O modelo Motley-Keenan [12] sugere que a perda média por multipercurso PL seja considerada como uma função da distância e calculada a partir da perda em espaço-livre $PL_0(d)$ e do número de paredes entre o transmissor(Tx) e receptor(Rx), como em (2-2) Já que o método considera somente as paredes do ambiente interior e ignora fortes espalhadores, como uma estante cheia de papeis (grau de atenuação algumas vezes maior que uma parede), os resultados obtidos são pouco representativos. Inicialmente, este método considerava somente paredes de um mesmo material (Lw_i), tendo sido feitas modificações de forma a considerar paredes de tipos diferentes (F_{wall}) e o chão (F_{floor}), permitindo obter melhores resultados.

PUC-Rio - Certificação Digital Nº 0310507/CB

Na versão inicial, se apresenta como:

$$PL\ (d)\ =\ PL_0(d)\ +\ \sum_{i=1}^{I} Lw_i \qquad \text{(2-2)}$$

Na versão modificada, se apresenta como:

$$PL\ (d)\ =\ PL_0(d)\ +\ 10n\ log(d/d_0)\ +\ \sum F_{wall}\ +\ \sum F_{floor} \qquad \text{(2-3)}$$

Na Tabela 2.2 são apresentados alguns valores de atenuação (Lw_i) para os tipos mais comuns de paredes.

Obstáculo	1,8 GHz	2,4 GHz	5,2 GHz
Concreto espesso (sem janela)	13	17	36
Vidraça	2	13	15
Parede com janela (valor exato depende da razão entre áreas de janelas e de concreto)	2 a 13	13 a 17	15 a 36

Tabela 2.2: Lw_i para tipos diferentes de paredes e diversas frequências.

PUC-Rio - Certificação Digital Nº 0310507/CB

2.1.3 Modelo "Modelo Multi-parede / andar"

O modelo de multi-parede/andar [15] calcula a perda de propagação no percurso em função da perda no espaço livre, somada com perdas introduzidas pela parede e chãos atravessados no percurso direto entre o transmissor e o receptor. Foi observado que a perda total introduzida pelo chão é uma função não-linear do número de andares atravessados. Esta característica é levada em conta através de um fator empírico b.

$$PL(d)\ =\ PL_0(d)\ +\ L_c\ +\ \sum_{i}^{I}\ m_i Lw_i\ +\ m_f^{\left[\frac{mf+2}{mf+1}-b\right]}\ PL_f \qquad [dB] \quad \text{(2-4)}$$

onde: L_c constante de perda, m_i número de paredes atravessadas pela sinal, m_f número de andares atravessados, Lw_i perda da propagação na parede de tipo i, $PL_f = 18.3$ dB perda da propagação entre andares adjacentes, $b = 0.46$ fator de correção empirico, I número de tipos de paredes.

A constante de perda na equação (2-4), é um termo que resulta quando as perdas nas paredes são determinadas através de medida usando a regressão linear múltipla, assumindo normalmente um valor próximo de zero. O terceiro termo em (2-4) expressa a perda total nas paredes como uma soma das perdas

individuais em cada parede entre transmissor e receptor. Por razões práticas, o número de diferentes tipos de paredes deve-se manter pequeno. Uma subdivisão em dois tipos de parede (fina e grossa) é proposta de acordo com a tabela 2.3. é importante notar que o fator de perda em (2-4) não representa perda física da parede mas os coeficientes do modelo que são otimizados utilizando os dados de perda no percurso medidos. Por conseguinte, os fatores de perda incluem implicitamente o efeito da mobília como também os efeitos guiados dos percursos através dos corredores.

Alguns valores utilizados para a equação (2-4) são especificados na Tabela 2.3

Tipo de parede	Lw_i	Descrição
Fina	3.4 dB	Parede divisória (não suporta carga),por exemplo: lâminas de gesso, tábuas de madeira ou paredes finas de concreto $< 10cm$)
Grossa	6.9 dB	Parede suportando carga ou paredes grossas ($> 10cm$), de, por exemplo concreto ou tijolo.

Tabela 2.3: Alguns valores das perdas de propagação Lw_i para 2 tipos diferentes de paredes usados na equação (2-4)

2.2 Métodos determinísticos para cobertura interiores

Vários métodos determinísticos têm sido usados para o cálculo da cobertura em ambientes interiores, entre os quais estão o método dos momentos (MoM) [17], método da equação parabólica (PE) [18], o método da Equação Integral (IE) [19, 20], e o método "Fast Multipole"(FMM) [21, 22].

Os mais utilizados atualmente são o método Traçado de Raios (RT) acoplado à Teoria Uniforme da Difração (UTD) e o método numérico de Diferenças Finitas no Domínio do tempo (FDTD), de interesse para os objetivos deste trabalho.

O método RT é um método que opera em freqüências altas, rastreando o campo eletromagnético ao longo das possíveis trajetórias (raios) entre o transmissor e receptor interagindo com os componentes do cenário. Devido aos traçado de raios, este modelo proporciona informação, não somente, sobre o espalhamento do sinal, mais também sobre o ângulo de chegada (de muito interesse em antenas "smart") no receptor, um resultado difícil de obter com outros métodos.

PUC-Rio - Certificação Digital Nº 0310507/CB

O método RT faz uso da ótica geométrica (GO) e da Teoria Uniforme da Difração (UTD), para analisar os mecanismos de espalhamento (reflexão, difração, refração) envolvidos nas interações das ondas eletromagnéticas com o cenário. O calculo destes mecanismos nos permite ter informação da amplitude e fase / retardo do sinal que chega ao receptor e seus detalhes serão revisto no capitulo 3. Diversas aplicações do método de RT aos cálculos de cobertura em ambientes interiores, utilizando modelos em 2 ou 3 dimensões, assim como diferentes modelamentos elétricos das paredes e obstáculos envolvidos, podem ser encontrados em [23, 24].

Uma análise dos trabalhos realizados até hoje revela, entretanto, a dificuldade em se incluir apropriadamente a variedade e disposição arbitrária de mobiliário e objetos (de dimensões variáveis em comparação aos comprimentos de onda usuais) no modelamento de cenários interiores, constituindo, freqüentemente, uma forte limitação na aplicação do método RT.

O método FDTD é uma técnica simples e efetiva que permite modelar a distribuição dos campos dentro de um ambiente interior, por exemplo um quarto ou um escritório com mobília (onde os móveis podem ser constituídos de materiais diferentes), com as equações de Maxwell dependentes do tempo sendo convertidas em equações de diferenças finitas com respeito a posições de cálculo de campo em uma malha. Estas equações são resolvidas no domínio do tempo, dada a posição de uma fonte eletromagnética e uma completa descrição de seu entorno em termos dos parâmetros constitutivos elétricos. Devido à possibilidade de cálculo da propagação da energia em posições dentro da malha, será possível identificar zonas com boa cobertura. A principal dificuldade na aplicação do método FDTD está relacionada com a necessidade de uma grande quantidade de memória para sua implementação numérica.

O uso do método FDTD em radiopropagação é mais recente em relação ao método RT como se pode constatar em [25, 26], sendo inicialmente usado para validar os resultados obtidos por RT [27]. Para lidar com o problema da memória, variantes do método FDTD têm surgido como o apresentado em [28] que é uma versão reduzida do FDTD. Em outro trabalho [25] trabalha-se em 2D e se apresenta uma metodologia de conversão de resultados para 3D. Em [29] é apresentado um modelo simplificado para calcular o perfil de potência de retardo.

Como mencionado anteriormente, um dos mecanismos tratados por RT é a reflexão especular e nem sempre as estruturas internas das paredes em ambientes interiores são consideradas mas, para cenários como hospitais, pode-se tornar relevante [30]. Em [31] são apresentados cálculos comparativos da perda de propagação por multipercurso em cenários interiores usando RT e FDTD,

PUC-Rio - Certificação Digital Nº 0310507/CB

evidenciando-se, a necessidade de considerar a estrutura interna das paredes e, em algumas situações, diferenças de 10 dB são observadas. Em cenários interiores é também comum que as antenas receptoras e/ ou transmissoras estejam próximas de um objeto espalhador, situação em que os cálculos via RT decrescem em precisão [32] que pode se agravar quando considerados sistemas MIMO [33]. Mas a situação mais critica acontece quando as dimensões do objeto espalhador são da ordem do comprimento de onda de trabalho, contrariando as premissas de aplicabilidade do método de rastreamento ótico.

Diante do exposto, a união dos métodos RT e FDTD de forma a aproveitar suas vantagens no trato de ambientes interiores é uma conseqüência natural e tem como trabalhos pioneiros os relatados em [8][18],[34],[35]. Estes trabalhos tratam em sua maioria de cenários em 2D, uma vez que o método FDTD precisa de uma quantidade considerável de memória para realizar os cálculos.

PUC-Rio - Certificação Digital Nº 0310507/CB

3
Método De Diferenças Finitas No Domínio Do Tempo (FDTD)

Neste capitulo é apresentado o método FDTD e suas diferentes técnicas complementares. O núcleo do método FDTD é considerado simples e de fácil entendimento [36, 37]. A seção 3.1 corresponde à visão geral introduzindo o método, características, modo de trabalho, pontos fortes e fracos, na seção (3.3) é descrita a formulação matemática do método.

Para abordar problemas de espalhamento, como é o caso de interesse neste trabalho, é necessário usar técnicas complementares e, de forma a permitir simular a propagação da onda no espaço livre, deve-se usar técnicas conhecidas "Absorbing Boundary Conditions" (ABC). Estas técnicas criam um meio absorvente ideal, composto de um número pequeno de células que não provocam a reflexão das ondas incidentes, conforme revisto na seção (3.5). De forma a considerar ondas planas como fontes externas para dominios de interesse, é necessário considerar a técnica "Campo Total / Campo Espalhado", abordada na seção (3.6).

PUC-Rio - Certificação Digital Nº 0310507/CB

3.1
Visão Geral

O método FDTD foi proposto inicialmente por Yee [38] em 1966 e, desde então, continua constantemente sendo complementado e considerado um método de ampla aceitação e aplicabilidade em diferentes áreas. Neste método, as equações de Maxwell, em sua forma diferencial, são transformadas em equações de diferenças finitas centrais que são, então, resolvidas em um esquema conhecido com o nome de "leap-frog", isto é, o campo elétrico é resolvido em um dado instante de tempo, o campo magnético é resolvido no próximo instante e o processo se repete ciclicamente.

3.1.1
Como o FDTD trabalha ?

Quando as equações de Maxwell em forma diferencial são examinadas, pode-se observar que a derivada com relação ao tempo do campo elétrico $\vec{E}$

é dependente do rotacional do campo magnético $\vec{H}$. Isto resulta na equação básica do método FDTD, em que o novo valor do campo $\vec{E}$ é dependente do valor antigo do campo de $\vec{E}$ (dai a diferença no tempo) e da diferença para o valor do campo $\vec{H}$ inmediatamente antes do ponto do espaço onde se calcula o campo elétrico $\vec{E}$. O campo $\vec{H}$ é encontrado de forma similar, sendo seu novo valor dependente do valor no instante de tempo anterior e da diferença no valor do campo $\vec{E}$ em relação a outro ponto no espaço.

3.1.2 Usando FDTD

As principais etapas na utilização do método FDTD podem ser descritas, sequencialmente:

(a) Um domínio computacional, onde há interesse em obter as soluções do problema em questão, deve ser estabelecido, como ilustrado na figura (3.1). Os campos $\vec{E}$ e $\vec{H}$ serão determinados em pontos deste domínio e os parâmetros característicos dos materiais empregados deverão ser fixados em cada célula no seu interior.

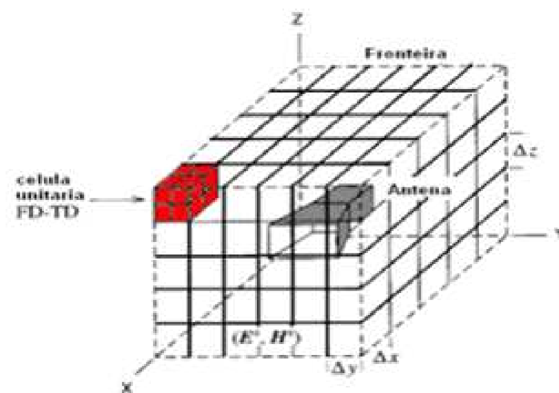

Figura 3.1: Dominio Computacional

(b) Uma vez estabelecido o domínio computacional, é preciso especificar uma fonte e sua excitação, isto é, o campo eletromagnético associado à onda incidente no domínio.

(c) Como os campos $\vec{E}$ e $\vec{H}$ são determinados diretamente o resultado da simulação é, usualmente, o campo eletromagnético em um ponto ou em uma serie de pontos dentro do domínio computacional.

PUC-Rio - Certificação Digital Nº 0310507/CB

3.1.3
Quais são os pontos fortes do método FDTD ?

FDTD é um método de modelagem usado em varios tipos de aplicações, quando usado na modelagem do campo elétromagnético no domínio do tempo em conjunto com um pulso como fonte, o processamento válido, para uma ampla faixa de frequências, é implementado em apenas uma execução da simulação. Pode-se, então, construir uma seqüência de quadros para mostrar a evolução da propagação do campo eletromagnético através do cenário.

A FDTD permite também ao usuário especificar o tipo de material em todos os pontos dentro do domínio computacional, de modo que podem ser considerados meios de propagação e objetos de constituições diferenciadas.

3.1.4
Quais são os pontos fracos do método FDTD ?

O principal problema associado ao método FDTD é a grande demanda computacional requerida em termos de memória. Como ele requer que o todo o domínio seja usado para criar a malha, e os elementos de malha deverão ser pequenos comparados ao menor comprimento de onda, implicará na escolha de domínios computacionais grandes e em processos lentos e computacionalmente pesados. Para estes casos ainda se usam supercomputadores ou redes com processamento paralelo, mas a contínua redução dos custos dos computadores e os avanços da tecnologia têm impulsionado as pesquisas com FDTD, conforme registrado em [39].

Um segundo problema relativo à implementação FDTD, mas já superado, era sua aplicação a problemas abertos (espalhamento eletromagnético), pois não havia formulações de truncagem para o método. O domínio computacional deve ser finito e condições especiais nas fronteiras devem ser estabelecidas, devendo permitir a simulação do espaço livre. Estas condições especiais são conhecidas como condições de fronteira absorventes (ABC - “Absorbing Boundary Conditions”). A primeira ABC (com sucesso aceitável) foi publicada em 1981 por Mur [40] e usada amplamente por muitos anos, ainda que, hoje, seja considerada uma técnica limitada devido à pouca eficiência de absorção no caso de incidências obliquas das ondas e, principalmente, nas chamadas regiões de canto [41]. Em 1994, foi publicada uma técnica de absorção extremamente eficiente por Berenger [42], chamada PML (“Perfectly Matched Layer”), seguida em 1996 da “Unixial PML” [43], e em 2000, a “Convolutional PML” [44].

O maior inconveniente com o método FDTD é sua inabilidade para lidar com limites curvos. Tais limites são aproximados com as chamadas escadas quando modeladas em grades cartesianas, introduzindo erros de segunda ordem

PUC-Rio - Certificação Digital Nº 0310507/CB

na técnica. Este problema é tratado com grades não uniformes, outros sistemas de coordenadas [36].

3.2
Formulação do método FDTD

Considerações gerais.

Em 1966, Kane Yee [38] propôs expressar em forma discreta (equações em diferenciais finitas) as equações de Maxwell. Neste esquema, divide-se a região de interesse em células cúbicas de coordenadas (ver figura 3.1):

$$(i, j, k) = (i\Delta x, j\Delta y, k\Delta z) \tag{3-1}$$

sendo $\Delta x, \Delta y, \Delta z$ os incrementos espaciais. Uma função dependente do espaço e do tempo se escreve:

$$F^n(i, j, k) = F(i\Delta x, j\Delta y, k\Delta z, n\Delta t) \tag{3-2}$$

sendo Δt, o incremento no tempo e n o numero de passos no tempo.

As derivadas espaciais e temporais da função são programadas utilizando uma aproximação em diferenças finitas centrais, avaliadas em células cúbicas na forma:

$$\frac{\partial(i, j, k)}{\partial x} = \frac{F^n(i + 1/2, j, k) - F^n(i - 1/2, j, k)}{\Delta x} \tag{3-3}$$

$$\frac{\partial(i, j, k)}{\partial t} = \frac{F^{n+1/2}(i, j, k) - F^{n-1/2}(i, j, k)}{\Delta t} \tag{3-4}$$

3.2.1
Algoritmo de Yee - Caso 3D

Para o caso tri-dimensional (3D), o domínio computacional é dividido em pequenas células. As componentes de $\vec{E}$ e $\vec{H}$ são distribuídas não espaço, com as componentes de $\vec{E}$ posicionadas nas bordas e as componentes de $\vec{H}$ nas faces, como se pode observar na figura (3.2).

No cubo de Yee pode-se observar que cada componente de campo $\vec{E}$ é rodeada de quatro componentes de campo $\vec{H}$ e, similarmente, cada componente de $\vec{H}$ é rodeada por quatro componentes $\vec{E}$, o que pode ser visto em detalhe na figura (3.3)

Em um meio linear, homogéneo, isotrópico e livre de fontes, as equações de Maxwell podem ser escritas como:

$$\nabla \times \vec{E} = \mu \frac{\partial \vec{H}}{\partial t} \tag{3-5}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

PUC-Rio - Certificação Digital Nº 0310507/CB

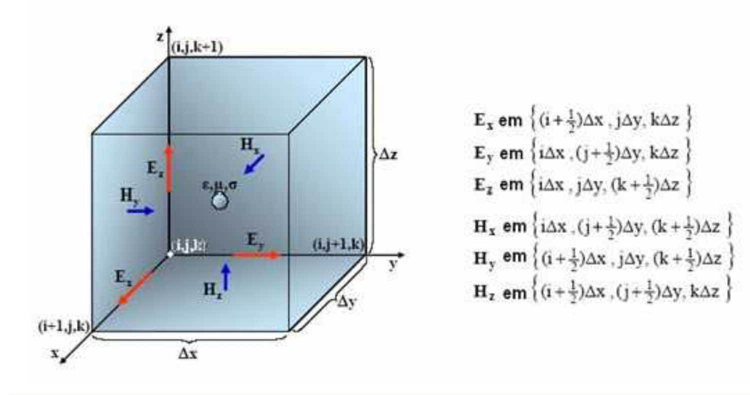

Figura 3.2: Célula unitária da malha de Yee

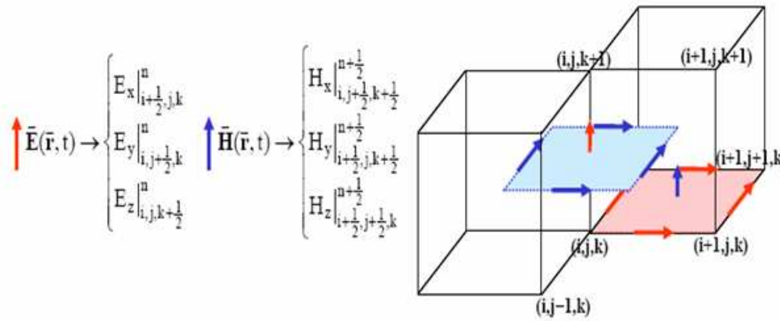

Figura 3.3: Distribuição das componentes de $\vec{E}$ e $\vec{H}$ no domínio computacional.

$$\nabla \times \vec{H} = \sigma \vec{E} + \epsilon \frac{\partial \vec{E}}{\partial t} \tag{3-6}$$

Explicitando a equação (3-5):

$$\left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}\right) = -\mu \frac{\partial H_x}{\partial t} \Longrightarrow \frac{\partial H_x}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y}\right) \tag{3-7}$$

$$-\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right) = -\mu \frac{\partial H_y}{\partial t} \Longrightarrow \frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right) \tag{3-8}$$

$$\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right) = -\mu \frac{\partial H_z}{\partial t} \Longrightarrow \frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}\right) \tag{3-9}$$

Explicitando a equação (3-6):

$$-\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x\right) = \epsilon \frac{\partial E_x}{\partial t} \Longrightarrow \frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x\right) \tag{3-10}$$

$$\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y\right) = \epsilon \frac{\partial E_y}{\partial t} \Longrightarrow \frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y\right) \tag{3-11}$$

$$\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) = \epsilon \frac{\partial E_y}{\partial t} \Longrightarrow \frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) \quad (3\text{-}12)$$

Para obter as equações de diferenças finitas em três dimensões, substitui-se as equações (3-3) e (3-4) nas equações (3-7) a (3-12) resultando:

$$\begin{aligned} H_x^{n+1/2}(i,j,k) &= H_x^{n-1/2}(i,j,k) + \left(\frac{\Delta t}{\mu(i,j,k)}\right)\left[\frac{E_y^n(i,j,k+1/2) - E_y^n(i,j,k-1/2)}{\Delta z}\right. \\ &\left. - \frac{E_z^n(i,j+1/2,k) - E_z^n(i,j-1/2,k)}{\Delta y}\right] \end{aligned} \quad (3\text{-}13)$$

$$\begin{aligned} H_y^{n+1/2}(i,j,k) &= H_y^{n-1/2}(i,j,k) + \left(\frac{\Delta t}{\mu(i,j,k)}\right)\left[\frac{E_z^n(i+1,j,k) - E_z^n(i-1/2,j,k)}{\Delta x}\right. \\ &\left. - \frac{E_x^n(i,j,k+1/2) - E_x^n(i,j,k-1/2)}{\Delta z}\right] \end{aligned} \quad (3\text{-}14)$$

$$\begin{aligned} H_z^{n+1/2}(i,j,k) &= H_z^{n-1/2}(i,j,k) + \left(\frac{\Delta t}{\mu(i,j,k)}\right)\left[\frac{E_x^n(i,j+1/2,k) - E_x^n(i,j-1/2,k)}{\Delta y}\right. \\ &\left. - \frac{E_y^n(i+1/2,j,k) - E_y^n(i-1/2,j,k)}{\Delta x}\right] \end{aligned} \quad (3\text{-}15)$$

$$\begin{aligned} E_x^{n+1}(i,j,k) &= Ca(i,j,k)E_x^n(i,j,k) + Cb(i,j,k)\left[\frac{H_z^{n+1/2}(i,j+1/2,k) - H_z^{n+1/2}(i,j-1/2,k)}{\Delta y}\right. \\ &\left. - \frac{H_y^{n+1/2}(i,j,k+1/2) - H_y^{n+1/2}(i,j,k-1/2)}{\Delta z}\right] \end{aligned} \quad (3\text{-}16)$$

$$\begin{aligned} E_y^{n+1}(i,j,k) &= Ca(i,j,k)E_y^n(i,j,k) + Cb(i,j,k)\left[\frac{H_x^{n+1/2}(i,j,k+1/2) - H_z^{n+1/2}(i+1/2,j,k)}{\Delta z}\right. \\ &\left. - \frac{H_z^{n+1/2}(i+1/2,j,k) - H_z^{n+1/2}(i-1/2,j,k)}{\Delta x}\right] \end{aligned} \quad (3\text{-}17)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$
\begin{aligned}
E_z^{n+1}(i,j,k) &= Ca(i,j,k)E_z^n(i,j,k) + Cb(i,j,k)\left[\frac{H_y^{n+1/2}(i+1/2,j,k) - H_y^{n+1/2}(i-1/2,j,k)}{\Delta x}\right. \\
&\left. - \frac{H_x^{n+1/2}(i,j+1/2,k) - H_x^{n+1/2}(i,j-1/2,k)}{\Delta y}\right]
\end{aligned} \tag{3-18}
$$

onde:

$$
Ca(i,j,k) = \frac{1 - \frac{\sigma(i,j,k)\Delta t}{2\epsilon(i,j,k)}}{1 + \frac{\sigma(i,j,k)\Delta t}{2\epsilon(i,j,k)}} \tag{3-19}
$$

$$
Cb(i,j,k) = \frac{\frac{\Delta t}{\epsilon(i,j,k)}}{1 + \frac{\sigma(i,j,k)\Delta t}{2\epsilon(i,j,k)}} \tag{3-20}
$$

3.2.2
Algoritmo de Yee - Caso 2D

Para o caso bi-dimensional (2D), remove-se a variação com uma das variáveis (por exemplo, z) nas equações (3-7) a (3-12). Todas as derivadas com relação a "z" são nulas e a estrutura não apresenta variação na direção z. Resulta:

Modo TM_z

$$
\frac{\partial H_x}{\partial t} = -\frac{1}{\mu}\frac{\partial E_z}{\partial y} \tag{3-21}
$$

$$
\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\frac{\partial E_z}{\partial x} \tag{3-22}
$$

$$
\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) \tag{3-23}
$$

Para obter as equações de diferenças finitas em 2D, substitui-se as equações (3-3) e (3-4) nas equações (3-21) a (3-23)

$$
\begin{aligned}
H_x^{n+1/2}(i,j) &= H_x^{n-1/2}(i,j) + \left(\frac{\Delta t}{\mu\Delta y}\right) \\
&\quad [E_z^n(i,j-1/2) - E_z^n(i,j+1/2)]
\end{aligned} \tag{3-24}
$$

$$
\begin{aligned}
H_y^{n+1/2}(i,j) &= H_y^{n-1/2}(i,j) + \left(\frac{\Delta t}{\mu\Delta x}\right) \\
&\quad [E_z^n(i+1/2,j) - E_z^n(i-1/2,j)]
\end{aligned} \tag{3-25}
$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\begin{aligned} E_z^{n+1}(i,j) &= E_z^n(i,j) + \left(\frac{\Delta t}{\epsilon \Delta x}\right)[H_y^{n+1/2}(i+1/2,j) \\ &\quad -H_y^{n+1/2}(i-1/2,j) + H_x^{n+1/2}(i,j-1/2) \\ &\quad -H_x^{i,j+1/2}(i,j+1/2)) - \Delta x \sigma E_z^n(i,j)] \end{aligned} \tag{3-26}$$

Modo TE_z

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \sigma E_x\right) \tag{3-27}$$

$$\frac{\partial E_y}{\partial t} = -\frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial x} + \sigma E_y\right) \tag{3-28}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}\right) \tag{3-29}$$

Para obter as equações de diferenças finitas em 2D, substitui-se as equações (3-3) e (3-4) nas equações (3-27) a (3-29)

$$\begin{aligned} E_x^{n+1}(i,j) &= E_x^n(i,j) + \left(\frac{\Delta t}{\epsilon \Delta y}\right)[H_z^{n+1/2}(i,j+1/2) \\ &\quad -H_z^{n+1/2}(i,j-1/2) - \sigma \Delta y E_x^n(i,j)] \end{aligned} \tag{3-30}$$

$$\begin{aligned} E_y^{n+1}(i,j) &= E_y^n(i,j) + \left(\frac{\Delta t}{\epsilon \Delta y}\right)[H_z^n(i,j+1/2) \\ &\quad -H_z^n(i,j-1/2) - \sigma \Delta y E_y^n(i,j)] \end{aligned} \tag{3-31}$$

$$\begin{aligned} H_z^{n+1/2}(i,j) &= H_z^{n-1/2}(i,j) + \left(\frac{\Delta t}{\mu \Delta y}\right)[E_x^n(i,j+1/2) - E_x^n(i,j-1/2) \\ &\quad +E_y^n(i-1/2,j) - E_y^n(i+1/2,j)] \end{aligned} \tag{3-32}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

3.2.3 Algoritmo de Yee - Caso 1D

Tomando um dos dois modos (2D TM ou 2D TE) e tirando as derivadas com relação a "z" ou "y". Isto implica que teremos somente propagação na direção do eixo "x".

$$\frac{\partial H_x}{\partial t} = 0 \tag{3-33}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x}\right) \tag{3-34}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \sigma E_z\right) \tag{3-35}$$

$$\frac{\partial E_x}{\partial t} = -\sigma E_x \frac{1}{\epsilon} \quad (3\text{-}36)$$

$$\frac{\partial E_y}{\partial t} = -\frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial x} + \sigma E_y\right) \quad (3\text{-}37)$$

$$\frac{\partial H_z}{\partial t} = -\frac{1}{\mu}\left(\frac{\partial E_y}{\partial x}\right) \quad (3\text{-}38)$$

As equações acima representam o núcleo do método FDTD para que o algoritmo as acomode apropriadamente, certas condições relacionadas com os valores discretos no tempo e no espaço devem ser satisfeitas.

3.3 Algoritmo de Yee - Consistência, convergência e estabilidade

Quando se utilizam equações de diferenças em substituição de uma formulação diferencial, é indispensável conhecer:

– Se, quando os incrementos decrescem, a equação em diferenças realmente se aproxima do problema diferencial e, assim sendo, diz-se que o esquema em diferencias é consistente.

– Se, quando os incrementos decrescem, a solução das equações de diferenças converge para a solução do problema diferencial e, assim sendo, diz-se que o esquema em diferenças converge para a solução real.

– Nos critérios anteriores, o tamanho dos incrementos é um fator determinante para a estabilidade do método e deve ser escolhido de forma que os campos eletromagnéticos não mudem substancialmente de um nó para o próximo na malha que corresponde ao domínio computacional.

– Para que um sistema de equações de diferenças seja estável, a dimensão da célula unitária (do incremento espacial) deverá ser uma fração do comprimento de onda. Em [37] sugere-se um passo menor que $\lambda/10$, oseja um comprimento de onda deve ser representado por, no mínimo, dez células e, em [45], sugere-se $\lambda/20$. Com relação ao incremento temporal, a estabilidade da solução se obtém aplicando o critério de Courant [37] que estabelece a condição:

$$\Delta t \leq \frac{1}{v\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}}} \quad (3\text{-}39)$$

onde: $v = \frac{1}{\sqrt{\epsilon\mu}}$ representa a velocidade de propagação no meio e, quando todas as células são do mesmo tamanho $\Delta x = \Delta y = \Delta z = \Delta$ (3-39) é reescrita como :

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\Delta t \leq \frac{1}{v\sqrt{3}} \tag{3-40}$$

– A equação (3-39) assegura que a distancia que a onda se propaga no intervalo de tempo Δt não seja maior que a largura da célula. Em malhas não uniformes, reescreve-se

$$\Delta t \leq \frac{1}{v_{max}\sqrt{\frac{1}{(\Delta x_{min})^2} + \frac{1}{(\Delta y_{min})^2} + \frac{1}{(\Delta z_{min})^2}}} \tag{3-41}$$

onde v_{max}, é a velocidade máxima de propagação da onda no modelo e Δx_{min}, Δy_{min}, Δz_{min} são os tamanhos mínimos em cada uma das direções.

3.4 Algoritmo de Yee - Memória e tempo de CPU

Cálculos com FDTD podem ser executados em uma gama extensa de computadores que disponham dos recursos necessários para executar a simulação, respeitados os critérios de estabilidade numérica estabelecidos por Courant [37].

3.4.1 Memória

A exigência de memória necessária para simular um problema com N células é determinada pela equação (3-42) introduzida em [45] para números reais de precisão simples.

$$ReqMem\ [bytes] = 28 \times Lx \times Ly \times Lz \times \left(\frac{Num\ amostras\ \lambda}{\lambda}\right)^3 \tag{3-42}$$

onde:

28 [bytes]	refre-se a seis componentes de campo mais uma referencia para o material, cada uma ocupando (4 bytes)
Lx	Dimensão da malha para o eixo x
Ly	Dimensão da malha para o eixo y
Lz	Dimensão da malha para o eixo z
$Num\ Amostras\ \lambda$	Numero de amostras consideradas para cada comprimento de onda (Equação (3-39)
λ	Comprimento de onda

Considerando dupla presisão, reescreve-se :

PUC-Rio - Certificação Digital Nº 0310507/CB

$$ReqMem\ [bytes] = 56 \times Lx \times Ly \times Lz \times \left(\frac{Num\ Amostras\ \lambda}{\lambda}\right)^3 \quad (3\text{-}43)$$

3.4.2 Tempo de cálculo

É possível determinar o tempo de cálculo total, quando se conhece a velocidade de operação do computador e o número de operações de ponto flutuante requeridas. O número de operações de ponto flutuante é dado por [45]:

$$\begin{aligned} Num\ Operacoes &= 10\sqrt{3}N^{-4/3} \left(\frac{6\ componentes}{celula}\ \frac{10\ Operacoes}{Componente}\right) \\ Num\ Operacoes &= 600\sqrt{3}N^{-4/3} \end{aligned} \quad (3\text{-}44)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

onde $N = Nx.Ny.Nz$ representa o produto dos números de células em x, y ,z na malha 3D, respectivamente, calculadas através de:

$$Nx = \frac{Lx}{\Delta x}, \quad Ny = \frac{Ly}{\Delta y} \quad Nz = \frac{Lz}{\Delta z} \quad (3\text{-}45)$$

Exemplo de aplicação

Calcule-se a memória requerida e o tempo de CPU usando as equações (3-43) e (3-44) considerando o núcleo do método FDTD-3D com seis componentes de campo elétrico e de campo magnético por célula, onde cada componente é descrito por 4 bytes (simples precisão) ou 8 bytes (dupla precisão) e em que cada uma das seis componentes precisa de 1 referência (para ter acesso a uma tabela com as propriedades do material). Considere-se também um ambiente de $6x15x3m^3$ e uma freqüência de $1.2GHz$.

Cálculos da memória

- Cálculo do comprimento de onda

$$\lambda = \frac{3 \times 10^8 m/s}{1.2 \times 10^9 1/s} = 0.25\ m$$

- $Lx = 6m$, $Ly = 15m$, $Lz = 3m$
- Numero de amostras por comprimento de onda = 10.

$$ReqMem\ = 56 \times 6 \times 15 \times 3 \times \left(\frac{10}{0.25}\right)^3 = 0.9677\ Gbytes$$

- considerando 12 amostras por comprimento de onda

$$ReqMem \;= 56 \times 6 \times 15 \times 3 \times \left(\frac{12}{0.25}\right)^3 = 1.6722 \; Gbytes$$

Cálculos do tempo de CPU

– Primeiro, calcula-se o número total de células, usando (3-44)

$$\begin{aligned} \Delta x &= \frac{\lambda}{12} = \frac{0,25\ m}{12} = 0,0208\ m, \qquad \Delta x = \Delta y = \Delta z = 0,0208\ m \\ Nx &= \frac{6}{0.0208} = 288,4615, \quad Ny = \frac{15}{0,0208} = 721,1538, \\ Nz &= \frac{3}{0,0208} = 144,2308 \end{aligned} \tag{3-46}$$

então $N = Nx \times Ny \times Nz = 29.901.312,0$ e

$$Num\ Operacoes = 600\ \sqrt{3}\ (29.901.312,0)^{4/3} = 9,6449 \times 10^{12}$$

– Considerando que este número de operações será efetuado em um computador que realiza $1,8 \times 10^9$ operações de ponto flutuante por segundo, então o tempo total de calculo para o caso considerado será de $5358,3\ seg$ ou 1 hora 29 minutos 18 segundos.

Com a finalidade de testar as equações apresentadas, considerou-se o problema da propagação de um pulso gaussiano gerado no centro da malha uniforme de 150×150 células, polarização TMz, tamanho do passo $x = 0,0003$, t segundo o critério de Courant (3-39), bloco metálico (condutor elétrico perfeito). Ilustrado na fig. 3.4.

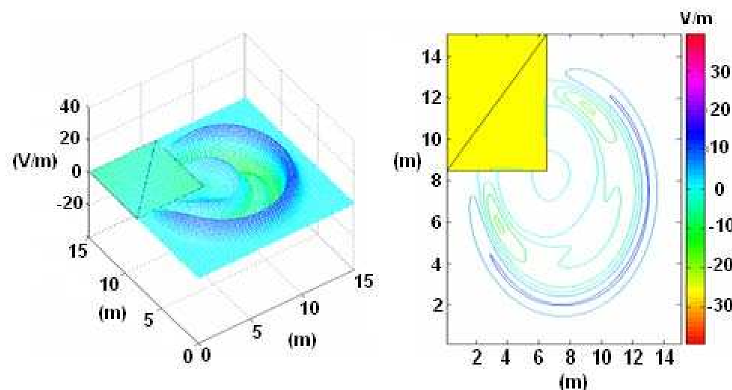

Figura 3.4: Interação de pulso gaussiano com um bloco metálico

Na simulação, a evolução do bloco é parada antes de o pulso atingir os limites da malha, de forma a evitar reflexões espúrias, as quais podem ser evitadas usando camadas absorventes, conforme apreciado na seguinte seção.

PUC-Rio - Certificação Digital Nº 0310507/CB

3.5 Método FDTD - Condições Absorventes

Para o tratamento de problemas de espalhamento eletromagnético em regiões abertas, o domínio computacional deve ser restrito e condições de fronteira absorventes (ABC)deverão ser estabelecidas.

é necessário restringir o domínio computacional, devido à limitada quantidade de memória disponível no computador e nestas situações, se utilizam ABCs nos limites da malha de forma a simular uma propagação ao infinito, evitando, assim, reflexões espúrias. Existem várias ABCs, sendo as mais conhecidas: Condição simples apresentada por Taflove e Brodwin [46], Condição absorvente de MUR de primeira e segunda ordem de precisão [40], Camada perfeitamente casada (PML) de Berenger [42], Convolutional (CPML) [44] de Roden e Gedney, Uniaxial PML (UPML) de Gedney [43].

Condições absorventes de bom desempenho são cruciais para a obtenção respostas exatas no domínio da freqüência, erros de absorção abaixo de -40 dB (condição absorvente de Mur) são desejáveis. Berenger publicou seu trabalho sobre PML(1994) reportando erros absorção abaixo de aproximadamente -140 dB [37].

3.6 Camada perfeitamente Casada (PML)

A PML, originalmente proposta em 2D [42], é uma condição de fronteira absorvente considerada perfeitamente casada porque as ondas eletromagnéticas, que se propagam em seu interior, se atenuam até quase desaparecer, sem produzir reflexões na interface meio com a PML, independentemente do ângulo de incidência e da freqüência de operação. Isto é conseguido substituindo o meio de propagação por outro especialmente projetado para o fim de absorção. Na fig. 3.5, pode-se observar a eficiência da camada PML, com o pulso gaussiano se propagando sem reflexões, conservando sua simetria e conseguindo portanto, simular a propagação em espaço livre.

A PML é formada por camadas absorventes colocadas em torno do domínio de estudo. A espessura da região é escolhida de acordo com o problema que se deseja solucionar [46], dentro de esta região, a componente de campo paralela à direção de propagação é dividida, possibilitando que perdas individuais para cada componente sejam designadas, de forma a criar um meio cuja impedância de onda independe do ângulo de incidência e da freqüência.

Considere-se a propagação de um campo eletromagnético de componentes Ex, Ey, eHz (modo TEz) no espaço bidimensional (x, y) constituído por um

PUC-Rio - Certificação Digital Nº 0310507/CB

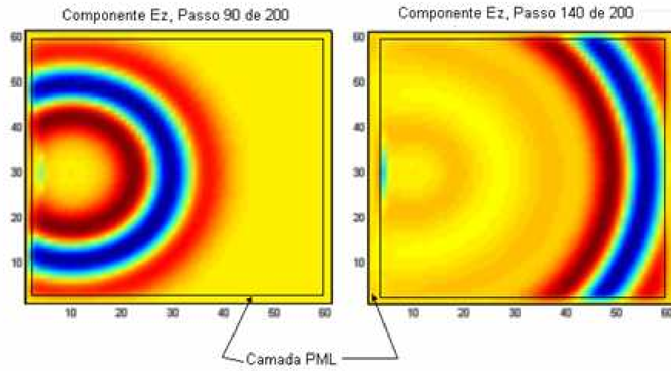

Figura 3.5: Pulso gaussiano propagando-se no domínio e interagindo com a PML (a) passo temporal 90 (b) passo temporal 140

meio com perdas elétricas e magnéticas. As três equações de Maxwell (equações (3-27)-(3-29)) se reduzem a:

$$\epsilon\frac{\partial E_x}{\partial t} + \sigma E_x = \frac{\partial H_z}{\partial y} \tag{3-47}$$

$$\epsilon\frac{\partial E_y}{\partial t} + \sigma E_y = \frac{\partial H_z}{\partial x} \tag{3-48}$$

$$\mu\frac{\partial H_z}{\partial t} + \sigma^* H_z = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \tag{3-49}$$

onde ϵ e μ são a permissividade eletrica e a permeabilidade do meio, e σ,σ^* representam as condutividades elétrica e magnética associadas às possíveis perdas no meio.

Holland [47]mostro que, se a expressão (3-50) é satisfeita no meio com perdas, então se verifica a condição de casamento de impedâncias $\eta_1 = \eta$ e uma onda plana que incide normalmente na interface não é refletida. Este trabalho serviu de base para a PML de Berenger.

$$\frac{\sigma}{\epsilon} = \frac{\sigma^*}{\mu} \tag{3-50}$$

Com a finalidade de conseguir uma adaptação perfeita para qualquer outro ângulo de incidência, Berenger propôs descompor, dentro da região PML, a componente Hz em duas sub-componentes H_{zx} e H_{zy} e introduzir um novo conjunto de condutividades $(\sigma_x, \sigma_x^*, \sigma_y, \sigma_y^*)$

3.6.1
Analise da PML - 2D

Para analisar a PML-2D considera-se o caso TEz, onde as equações de Maxwell são modificadas no meio PML conforme segue. Berenger [42], em sua

PUC-Rio - Certificação Digital Nº 0310507/CB

formulação também conhecida como "não maxweliana", dividiu a componente de campo Hz usando $Hz = H_{zx} + H_{zy}$ e introduzindo, neste ponto, um novo grau de liberdade na especificação das perdas e a adaptação da impedância,

$$\mu\frac{\partial H_z}{\partial t} + \sigma * H_z = \left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right) \Rightarrow \begin{cases} \mu\frac{\partial H_{zx}}{\partial t} + \sigma_x^* H_{zx} = -\frac{\partial E_y}{\partial x} \\ \\ \mu\frac{\partial H_{zy}}{\partial t} + \sigma_y^* H_{zy} = -\frac{\partial E_x}{\partial y} \end{cases} \tag{3-51}$$

$$\epsilon\frac{\partial E_x}{\partial t} + \sigma E_x = \frac{\partial H_z}{\partial y} \quad \Rightarrow \quad \epsilon\frac{\partial E_x}{\partial t} + \sigma_y E_x = \frac{\partial(H_{zx} + H_{zy})}{\partial y} \tag{3-52}$$

$$\epsilon\frac{\partial E_y}{\partial t} + \sigma E_y = -\frac{\partial H_z}{\partial z} \quad \Rightarrow \quad \epsilon\frac{\partial E_y}{\partial t} + \sigma_x E_y = -\frac{\partial(H_{zx} + H_{zy})}{\partial x} \tag{3-53}$$

se, nas equações (3-51)-(3-53):

$\sigma_x = \sigma_y = \sigma_x^* = \sigma_y^* = 0$	obtém-se as equações de Maxwell no vácuo.
$\sigma_x = \sigma_y$ e $\sigma_x^* = \sigma_y^* = 0$	obtém-se as equações de Maxwell correspondentes a um meio condutor com perdas
$\sigma_x = \sigma_y$ e $\sigma_x^* = \sigma_y^*$	obtém-se as equações de Maxwell para um meio absorvente hipotético, cuja impedância estaria adaptada para a incidência normal de qualquer onda plana originada no vácuo.
$\sigma_x = \sigma_y^* = 0$	o médio PML pode absorver uma onda plana de componentes (E_y, H_{zx}) propagando-se ao longo do eixo "x", mas não absorve uma onda propagando-se ao longo do eixo "y"; a situação inversa é verdadeira no caso que $\sigma_x = \sigma_x^* = 0$.

Assim, tem-se que nos meios PML caracterizados por parâmetros $(\sigma_x, \sigma_x^*, 0, 0)$ e $(0, 0, \sigma_y, \sigma_y^*)$, e cujas condutividades satisfaçam 3-50, não produzirâo nenhuma reflexão nas superfícies de separação normais aos eixos "x" e "y" respectivamente.

Similarmente, para o caso TMz, seguem as equações de Maxwell modificadas na PML, para as quais Berenger dividiu a componente de campo Ez na forma $Ez = E_{zx} + E_{zy}$:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\epsilon\frac{\partial E_z}{\partial t} + \sigma H_z = -\left(\frac{\partial H_x}{\partial y} - \frac{\partial H_y}{\partial x}\right) \Rightarrow \begin{cases} \epsilon\frac{\partial E_{zx}}{\partial t} + \sigma_x E_{zx} = \frac{\partial H_y}{\partial x} \\ \\ \epsilon\frac{\partial E_{zy}}{\partial t} + \sigma_y E_{zy} = -\frac{\partial H_x}{\partial y} \end{cases} \tag{3-54}$$

$$\mu\frac{\partial H_x}{\partial t} + \sigma_y^* H_z = -\frac{\partial E_z}{\partial y} \quad \Rightarrow \quad \mu\frac{\partial H_x}{\partial t} + \sigma_y^* H_z = -\frac{\partial (E_{zx} + E_{zy})}{\partial y} \tag{3-55}$$

$$\mu\frac{\partial H_y}{\partial t} + \sigma_x^* H_y = -\frac{\partial E_z}{\partial x} \quad \Rightarrow \quad \mu\frac{\partial H_y}{\partial t} + \sigma_x^* H_y = \frac{\partial (E_{zx} + E_{zy})}{\partial x} \tag{3-56}$$

A região que rodeia o domínio computacional, proposta por Berenger com a finalidade de simular o espaço livre, está representada na figura (3.6).As componentes de condutividade σ_x, σ_x^* e σ_y, σ_y^* são distribuídas de forma que evitem qualquer reflexão na superfície de separação entre o meio que corresponde à região da simulação e a região PML. Na parte externa à região PML usa-se um condutor perfeito (PEC).

Dado que uma transição forte do valor da condutividade na interface do meio com a PML pode provocar reflexões indesejadas, para uma determinada espessura (δ) da PML, se escolhem os valores das condutividades correspondentes às camadas, de forma que variem desde zero na interface (quando $\rho = 0$) até um valor máximo σ_{max} (quando $\rho = \delta$) ao final da PML. Berenger propôs que esta variação seja realizada em concordância com a seguinte relação:

$$\begin{aligned} \sigma(\rho) &= \sigma_{max}\left(\frac{\rho}{\delta}\right)^n \\ \sigma^*(\rho + 1/2) &= \sigma_{max}^*\left(\frac{\rho + 1/2}{\delta}\right)^n \end{aligned} \tag{3-57}$$

Com relação a (3-57), os perfis mais comuns são o linear (n=1), o parabólico (n=2) e o cúbico (n=3).

Geralmente, a parte externa da PML é rodeada por um condutor elétrico perfeito (PEC) de forma que as ondas enfraquecidas que atingem este condutor voltarão ao domínio computacional (figura 3.7). Para uma região de espessura δ, pode-se definir o seguinte coeficiente de reflexão teórico [37]:

$$R(\varphi) = [R(0)]^{cos\varphi} \tag{3-58}$$

onde φ é o ângulo que a onda incidente forma com a PML e $R(0)$ é dado por (3-59)

$$R(0) = e^{-2\frac{\sigma_{max}}{(n+1)\epsilon_0 c}\delta} \tag{3-59}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

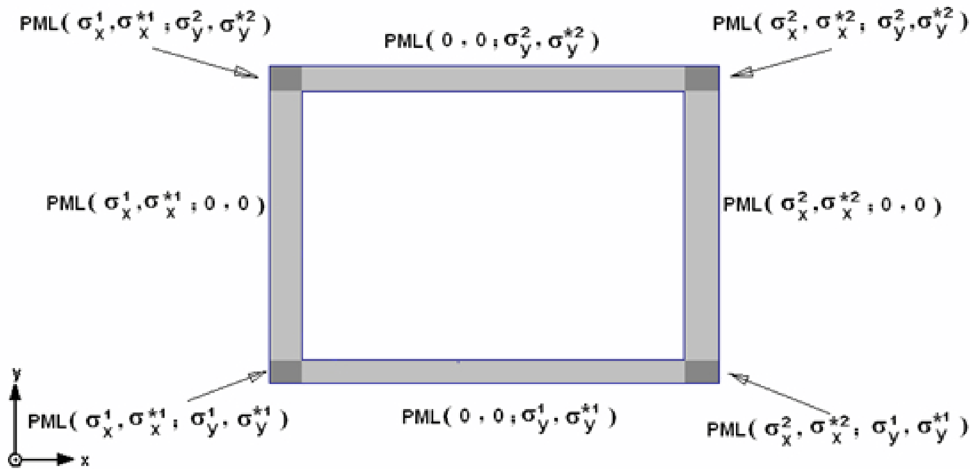

Figura 3.6: Distribuição das condutividades - Camada PML- 2D

O fator de reflexão em (3-59) depende do produto $(\sigma\dot{\delta})$, teòricamente, poderiam se obter valores tão pequenos quanto se desejasse, incrementando o valor de δ , de σ ou ambos. Valores típicos para (3-59) são $\delta = 8$, $n = 3$, e, escolhendo um valor de fator de reflexão para incidência normal $R(0) < 10^{-4} ou R(0) < 10^{-5}$, o valor máximo da condutividade pode ser calculado de (3-59) na forma:

$$\sigma_{max} = -\frac{(n+1)\epsilon_0 \; c \; ln(R(0))}{2\delta} \tag{3-60}$$

A figura (3.7) ilustra a propagação de um pulso gaussiano interagindo com um bloco PEC e com as fronteiras do domínio computacional. A figura (3.7).a ilustra as reflexões espúrias produzidas nas fronteiras do domínio computacional e a figura (3.7.b) ilustra para um mesmo instante de tempo, a absorção das ondas pela PML.

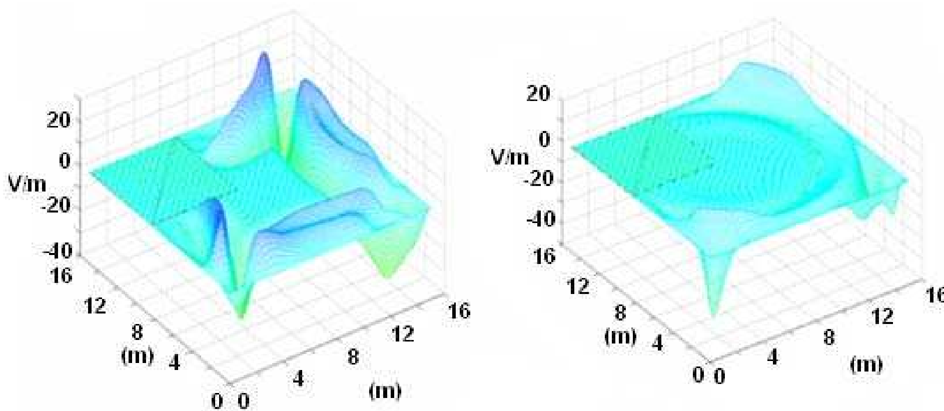

Figura 3.7: Pulso gaussiano interagindo com o bloco PEC e com as fronteiras do domínio computacional a) sem PML, b) com PML.

PUC-Rio - Certificação Digital Nº 0310507/CB

Passo do tempo exponencial

A atenuação da amplitude das ondas que se propagam em um meio com grandes perdas como é a PML, é tão rápida que pode provocar efeitos não desejados (como reflexões das ondas que incidem sobre ele). Uma forma de se evitar isto consiste em modificar o modo em que o campo evolui no tempo. Assim, em vez de utilizar uma aproximação em diferencias finitas centrais Δt, é mais conveniente utilizar uma evolução no tempo exponencial [37], definiendo-se os coeficientes Ca, Cb, Da, Db, para a camada PML:

$$\begin{aligned} r_e = \frac{\sigma(m)\ \Delta t}{\epsilon_0}, \quad Ca(i) = e^{-r_e}, \quad Cb(i) = -\frac{e^{-r_e} - 1}{\Delta x\ \sigma(i)} \\ r_m = \frac{\sigma^*(m)\ \Delta t}{\mu_0}, \quad Da(i) = e^{-r_m}, \quad Db(i) = -\frac{e^{-r_m} - 1}{\Delta x\ \sigma^*(i)} \end{aligned} \tag{3-61}$$

Analise da PML - 3D

Para o caso 3D, a subdivisão do campo é estendida, com as seis componentes de campo das equações de Maxwell sendo divididas em duas sub-componentes cada:

$$H_x = H_{xy} + H_{xz}, \quad H_y = H_{yz} + H_{yx}, \quad H_z = H_{zx} + H_{zy} \tag{3-62}$$

resultando, para as seis equações escalares (3-16)-(3-15) as expressões:

$$\epsilon \frac{\partial E_{xy}}{\partial t} + \sigma_{ey} E_{xy} = \frac{\partial (H_{zx} + H_{zy})}{\partial y} \tag{3-63}$$

$$\epsilon \frac{\partial E_{xz}}{\partial t} + \sigma_{ez} E_{xz} = -\frac{\partial (H_{yz} + H_{yx})}{\partial z} \tag{3-64}$$

$$\epsilon \frac{\partial E_{yz}}{\partial t} + \sigma_{ez} E_{yz} = \frac{\partial (H_{xy} + H_{xz})}{\partial z} \tag{3-65}$$

$$\epsilon \frac{\partial E_{yx}}{\partial t} + \sigma_{ex} E_{yx} = -\frac{\partial (H_{zy} + H_{zy})}{\partial x} \tag{3-66}$$

$$\epsilon \frac{\partial E_{zx}}{\partial t} + \sigma_{ex} E_{zx} = \frac{\partial (H_{yz} + H_{yx})}{\partial x} \tag{3-67}$$

$$\epsilon \frac{\partial E_{zy}}{\partial t} + \sigma_{ey} E_{zy} = -\frac{\partial (H_{xy} + H_{xz})}{\partial y} \tag{3-68}$$

Similarmente, considerando:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$E_x = E_{xy} + E_{xz}, \quad E_y = E_{yz} + E_{yx}, \quad E_z = E_{zx} + E_{zy} \tag{3-69}$$

Obtém-se:

$$\mu \frac{\partial H_{xy}}{\partial t} + \sigma_{my} H_{xy} = -\frac{\partial (E_{zx} + E_{zy})}{\partial y} \tag{3-70}$$

$$\mu \frac{\partial H_{xz}}{\partial t} + \sigma_{mz} H_{xz} = -\frac{\partial (E_{yz} + E_{yx})}{\partial z} \tag{3-71}$$

$$\mu \frac{\partial H_{yz}}{\partial t} + \sigma_{mz} H_{yz} = -\frac{\partial (E_{xy} + E_{xz})}{\partial z} \tag{3-72}$$

$$\mu \frac{\partial H_{yx}}{\partial t} + \sigma_{mx} H_{yx} = \frac{\partial (E_{zy} + E_{zy})}{\partial x} \tag{3-73}$$

$$\mu \frac{\partial H_{zx}}{\partial t} + \sigma_{mx} H_{zx} = -\frac{\partial (E_{yz} + E_{yx})}{\partial x} \tag{3-74}$$

$$\mu \frac{\partial H_{zy}}{\partial t} + \sigma_{my} H_{zy} = \frac{\partial (E_{xy} + E_{xz})}{\partial y} \tag{3-75}$$

De modo similar ao que acontecia em duas dimensões, uma onda propagando-se ao longo da direção x que incida sobre a região PML caracterizada pelos parâmetros $(\sigma_x, 0, 0, \sigma_x^*, 0, 0)$, onde o par de parâmetros (σ_x, σ_x^*) satisfaça a relação 3-50 não sofrerá reflexão, independentemente do ângulo de incidência e de sua freqüência, motivo pelo qual a figura (3.6) pode ser generalizada para 3 dimensões. Para uma PML-3D, a equação (3-50), de forma a facilitar a inserção das condutividades, é re-escrita como:

$$\frac{\sigma_n}{\epsilon} = \frac{\sigma_n^*}{\mu} \tag{3-76}$$

onde $n$denota o eixo ortogonal à superfície planar e as outras condutividades (ao longo dos eixos tangenciais à interface) serão nulos.

Em uma quina diedra onde duas PMLs ortogonais se sobrepõem, dois pares de condutividades são diferentes de zero (aquelas que são diferentes de zero nas PML vizinhas) e em uma quina triedra onde três PMLs se sobrepõem, tódas as seis condutividades são diferentes de zero.

3.7 Camada Uniaxial Perfeitamente Casada - UPML

Como para a implementação da PML-3D, são necessárias 12 equações, precisa-se de uma quantidade considerável de memória e, para 3D, é comum usar a PML uniaxial formulada por Gedney [43]. A UPML não segue o esquema de divisão dos campos e é chamada de PML Maxweliana.

PUC-Rio - Certificação Digital Nº 0310507/CB

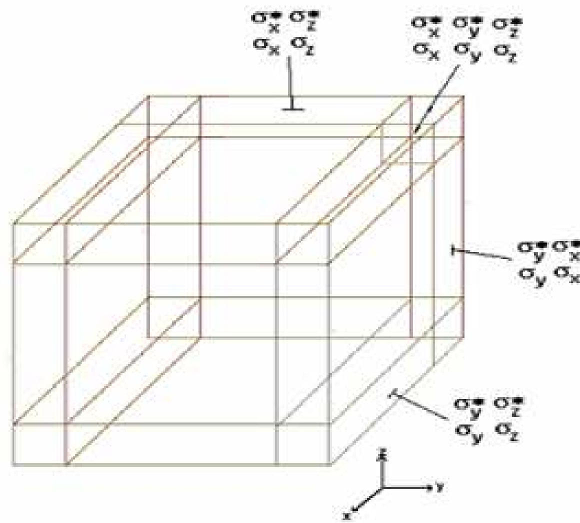

Figura 3.8: PML 3D - especificação de algumas condutividades

As equações de Maxwell em rotacional, em sua forma armônica, podem ser escritas na UPML de forma geral, como [37]:

$$
\begin{aligned}
\nabla \times \vec{H} &= j\omega\epsilon\bar{s}E \\
\nabla \times \vec{E} &= -j\omega\epsilon\bar{s}H
\end{aligned}
\tag{3-77}
$$

onde o tensor $\bar{s}$, que representa a relação constitutiva associada à atenuação normal em todos os eixos, é representado por:

$$
\begin{aligned}
\bar{s} &= \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_x & 0 \\ 0 & 0 & 1/s_x \end{bmatrix} \cdot \begin{bmatrix} 1/s_y & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1/s_y \end{bmatrix} \cdot \begin{bmatrix} 1/s_z & 0 & 0 \\ 0 & 1/s_z & 0 \\ 0 & 0 & 1/s_z \end{bmatrix} \\
\bar{s} &= \begin{bmatrix} s_y s_z/s_x & 0 & 0 \\ 0 & s_x s_z/s_y & 0 \\ 0 & 0 & s_x s_y/s_z \end{bmatrix}
\end{aligned}
\tag{3-78}
$$

com,

$$
s_x = k_x + \frac{\sigma_x}{j\ \omega\epsilon}, \qquad s_y = k_y + \frac{\sigma_y}{j\ \omega\epsilon}, \qquad s_z = k_z + \frac{\sigma_z}{j\ \omega\epsilon} \tag{3-79}
$$

onde k_x, k_y, k_z são constantes de atenuação de forma a absorver a onda eletromagnética.

Pode-se definir o espaço FDTD como sendo um meio uniaxial, tendo em cada região seu valor específico de condutividade. Das definições acima,

PUC-Rio - Certificação Digital Nº 0310507/CB

escolhem-se os parâmetros para tôda a malha FDTD, seguindo os critérios (as condutividades elétricas possuem localização igual à PML) abaixo:

– Parte interior da malha isotrópica e sem perdas:

$$s_x = s_y = s_z = 1 \rightarrow \begin{cases} \sigma_x = \sigma_y = \sigma_z = 0 \\ \\ k_x = k_y = k_z = 1 \end{cases}$$

– UPML entre x_{min} e x_{max} (sem interseção entre planos)

$$s_y = s_z = 1 \rightarrow \begin{cases} \sigma_y = \sigma_z = 0 \\ \\ k_y = k_z = 1 \end{cases}$$

– UPML entre y_{min} e y_{max} (sem interseção entre planos)

$$s_x = s_z = 1 \rightarrow \begin{cases} \sigma_x = \sigma_z = 0 \\ \\ k_x = k_z = 1 \end{cases}$$

– UPML entre z_{min} e z_{max} (sem interseção entre planos)

$$s_x = s_y = 1 \rightarrow \begin{cases} \sigma_x = \sigma_z = 0 \\ \\ k_x = k_z = 1 \end{cases}$$

– UPML entre x_{min}, x_{max} e y_{min}, y_{max} (interseção entre os planos x e y - cantos diedrais):

$$s_z = 1 \rightarrow \begin{cases} \sigma_z = 0 \\ \\ k_z = 1 \end{cases}$$

– UPML entre x_{min}, x_{max} e z_{min}, z_{max} (interseção entre os planos x e z - cantos diedrais):

$$s_y = 1 \rightarrow \begin{cases} \sigma_y = 0 \\ \\ k_y = 1 \end{cases}$$

– UPML entre y_{min}, y_{max} e z_{min}, z_{max} (interseção entre os planos y e z - cantos diedrais):

$$s_x = 1 \rightarrow \begin{cases} \sigma_x = 0 \\ \\ k_x = 1 \end{cases}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

– UPML na interseção de todos os planos (cantos triedais): usa-se o tensor completo em (3-78) , ou seja, tôdas as condutividades e, geralmente, com $k_s = 1$.

Considerando a lei de Ampère, com o auxílio do tensor dado pela expressão (3-78), são calculadas as componentes de campo elétrico:

$$\begin{bmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{bmatrix} = j\ \omega\epsilon \begin{bmatrix} \frac{s_y\ s_z}{s_x} & 0 & 0 \\ 0 & \frac{s_x\ s_z}{s_y} & 0 \\ 0 & 0 & \frac{s_x\ s_y}{s_z} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \tag{3-80}$$

inserindo (3-79) em (3-80) e transformando para o domínio do tempo, resultaria em uma convolução entre o tensor e o campo elétrico, o que é computacionalmente pesado [37]. Para superar isto, define-se:

$$D_x = \epsilon \frac{s_y}{s_x} E_x, \qquad D_y = \epsilon \frac{s_z}{s_y} E_y, \qquad D_z = \epsilon \frac{s_x}{s_z} E_y \tag{3-81}$$

Então, (3-81) é reescrita como :

$$\begin{bmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{bmatrix} = j\ \omega \begin{bmatrix} s_z & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & s_y \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \tag{3-82}$$

Substituindo s_x, s_y, e s_z de (3-79) em (3-82), e aplicando a transformada inversa de Fourier, com $j\ \omega f(\omega) \to (\partial\ /\partial t) f(t)$, resulta o sistema de equações diferenciais no domínio do tempo:

$$\begin{bmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{bmatrix} = \frac{\partial}{\partial t} \begin{bmatrix} k_z & 0 & 0 \\ 0 & k_x & 0 \\ 0 & 0 & k_y \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} + \frac{1}{\epsilon} \begin{bmatrix} \sigma_z & 0 & 0 \\ 0 & \sigma_x & 0 \\ 0 & 0 & \sigma_y \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \tag{3-83}$$

A partir do sistema de equações de (3-83), pode-se aplicar o algoritmo de Yee às induções elétricas Dx, Dy, eDz e seus respectivos campos magnéticos, gerando um conjunto de equações FDTD dadas por:

$$\begin{aligned} D_x\Big|_{i+1/2,j,k}^{n+1} &= \left[\frac{2\ \epsilon k_z - \sigma_z \Delta t}{2\ \epsilon k_z + \sigma_z \Delta t}\right] D_x\Big|_{i+1/2,j,k}^{n} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_z + \sigma_z \Delta t)\Delta}\right] \\ & \left[Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2} - Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2} + \right. \\ & \left. Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} - Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} \right] \end{aligned} \tag{3-84}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$D_y\Big|_{i,j+1/2,k}^{n+1} = \left[\frac{2\ \epsilon k_z - \sigma_x \Delta t}{2\ \epsilon k_x + \sigma_x \Delta t}\right] D_y\Big|_{i,j+1/2,k}^{n} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_x + \sigma_x \Delta t)\Delta}\right] \left[Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2} - Hx\Big|_{i,j+1/2,k-1/2}^{n+1/2} + Hz\Big|_{i-1/2,j+1/2,k}^{n+1/2} - Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2}\right] \quad (3\text{-}85)$$

$$D_z\Big|_{i,j,k+1/2}^{n+1} = \left[\frac{2\ \epsilon k_y - \sigma_y \Delta t}{2\ \epsilon k_y + \sigma_y \Delta t}\right] D_z\Big|_{i,j,k+1/2}^{n} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_y + \sigma_y \Delta t)\Delta}\right] \left[Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} - Hy\Big|_{i-1/2,j,k+1/2}^{n+1/2} + Hx\Big|_{i,j-1/2,k+1/2}^{n+1/2} - Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2}\right] \quad (3\text{-}86)$$

Para relacionar $\vec{D}$ e $\vec{E}$ usa-se (3-81). Considerando somente Dx, multiplicando ambos lados por s_x e substituindo seu valor em (3-79) , tem se:

$$\left[k_x + \frac{\sigma_x}{j\ \omega\epsilon}\right]\ D_x = \epsilon \left[k - y + \frac{\sigma_y}{j\ \omega\epsilon}\right]\ E_x \quad (3\text{-}87)$$

Multiplicando ambos lados por $j\ \omega$ e transformando para o domínio do tempo, resulta:

$$\frac{\partial}{\partial t}(k_x\ D_x) + \frac{\sigma_x}{\epsilon} D_x = \epsilon \left[\frac{\partial}{\partial t}(k_y\ E_x) + \frac{\sigma_y}{\epsilon} E_x\right] \quad (3\text{-}88)$$

Similarmente, as componentes Dy e Dz podem ser associadas as componentes respectivas de campo elétrico e o algoritmo de Yee pode ser empregado, resultando as equações FDTD para o campo elétrico:

$$E_x\Big|_{i+1/2,j,k}^{n+1} = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right] Ex\Big|_{i+1/2,j,k}^{n} + \left[\frac{1}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\epsilon}\right] \left[(2\ \epsilon k_x + \sigma_x\ \Delta t) D_x\Big|_{i+1/2,j,k}^{n+1} - (2\ \epsilon k_x - \sigma_x\ \Delta t) D_x\Big|_{i+1/2,j,k}^{n}\right] \quad (3\text{-}89)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$E_y\Big|_{i,j+1/2,k}^{n+1} = \left[\frac{2\ \epsilon k_z - \sigma_z\ \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right] Ey\Big|_{i,j+1/2,k}^{n} + \left[\frac{1}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\epsilon}\right] \left[(2\ \epsilon k_z + \sigma_z\ \Delta t) D_y\Big|_{i,j+1/2,k}^{n+1} - (2\ \epsilon k_z - \sigma_y\ \Delta t) D_y\Big|_{i,j+1/2,k}^{n}\right] \quad (3\text{-}90)$$

$$E_z\Big|_{i,j,k+1/2}^{n+1} = \left[\frac{2\ \epsilon k_x - \sigma_x\ \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right] Ey\Big|_{i,j,k+1/2}^{n} + \left[\frac{1}{(2\ \epsilon k_x + \sigma_x\ \Delta t)\epsilon}\right] \left[(2\ \epsilon k_z + \sigma_z\ \Delta t) D_z\Big|_{i,j,k+1/2}^{n+1} - (2\ \epsilon k_z - \sigma_z\ \Delta t) D_z\Big|_{i,j,k+1/2}^{n}\right] \quad (3\text{-}91)$$

Considerando as igualdades:

$$\begin{array}{lll} C1x = \left[\frac{2\ \epsilon k_x - \sigma_x\ \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right], & C2x = \left[\frac{2\ \epsilon \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right], & C3x = \left[\frac{2\ \epsilon k_x - \sigma_x\ \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right] \\ C1y = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right], & C2y = \left[\frac{2\ \epsilon \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right], & C3y = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right] \\ C1z = \left[\frac{2\ \epsilon k_z - \sigma_z\ \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right], & C2z = \left[\frac{2\ \epsilon \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right], & C3z = \left[\frac{2\ \epsilon k_z - \sigma_z\ \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right] \end{array} \quad (3\text{-}92)$$

$$\begin{array}{ll} C4ex = \left[\frac{1}{(2\epsilon k_x + \sigma_x \Delta t)\epsilon}\right], & C4hx = \left[\frac{1}{(2\epsilon k_x + \sigma_x \Delta t)\mu}\right] \\ C4ey = \left[\frac{1}{(2\epsilon k_y + \sigma_y \Delta t)\epsilon}\right], & C4hy = \left[\frac{1}{(2\epsilon k_y + \sigma_y \Delta t)\mu}\right] \\ C4ez = \left[\frac{1}{(2\epsilon k_z + \sigma_z \Delta t)\epsilon}\right], & C4hz = \left[\frac{1}{(2\epsilon k_z + \sigma_z \Delta t)\mu}\right] \end{array} \quad (3\text{-}93)$$

$$\begin{array}{ll} C5x = (2\epsilon k_x + \sigma_x \Delta t) & C6x = (2\epsilon k_x - \sigma_x \Delta t) \\ C5y = (2\epsilon k_y + \sigma_y \Delta t) & C6y = (2\epsilon k_y - \sigma_y \Delta t) \\ C5z = (2\epsilon k_z + \sigma_z \Delta t) & C6z = (2\epsilon k_z - \sigma_z \Delta t) \end{array} \quad (3\text{-}94)$$

e reescrevendo as equações (3-85)-(3-91) de forma a facilitar a programação, obtem-se:

$$D_x\Big|_{i+1/2,j,k}^{n+1} = C1z\ D_x\Big|_{i+1/2,j,k}^{n} + C2z\left[Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2} - Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2} + Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} - Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2}\right] \quad (3\text{-}95)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\begin{aligned} D_y\Big|_{i,j+1/2,k}^{n+1} &= C1x\ D_y\Big|_{i,j+1/2,k}^{n} + C2x\left[Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2} - Hx\Big|_{i,j+1/2,k-1/2}^{n+1/2} + \right. \\ &\left. Hz\Big|_{i-1/2,j+1/2,k}^{n+1/2} - Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2}\right] \end{aligned} \quad (3\text{-}96)$$

$$\begin{aligned} D_z\Big|_{i,j,k+1/2}^{n+1} &= C1y\ D_z\Big|_{i,j,k+1/2}^{n} + C2y\left[Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} - Hy\Big|_{i-1/2,j,k+1/2}^{n+1/2} + \right. \\ &\left. Hx\Big|_{i,j-1/2,k+1/2}^{n+1/2} - Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2}\right] \end{aligned} \quad (3\text{-}97)$$

$$E_x\Big|_{i+1/2,j,k}^{n+1} = C1y\ Ex\Big|_{i+1/2,j,k}^{n} + C4ey\left[C5x\ D_x\Big|_{i+1/2,j,k}^{n+1} - C6x\ D_x\Big|_{i+1/2,j,k}^{n}\right] \quad (3\text{-}98)$$

$$E_y\Big|_{i,j+1/2,k}^{n+1} = C1z\ Ey\Big|_{i,j+1/2,k}^{n} + C4ez\left[C5y\ D_y\Big|_{i,j+1/2,k}^{n+1} - C6y\ D_y\Big|_{i,j+1/2,k}^{n}\right] \quad (3\text{-}99)$$

$$E_z\Big|_{i,j,k+1/2}^{n+1} = C1x\ Ey\Big|_{i,j,k+1/2}^{n} + C4ex\left[C5z\ D_z\Big|_{i,j,k+1/2}^{n+1} - C6z\ D_z\Big|_{i,j,k+1/2}^{n}\right] \quad (3\text{-}100)$$

A atualização das componentes de campo elétrico requer dois passos: (1) obtenção dos novos valores de $\vec{D}$ e (2) uso destes valores para o cálculo de $\vec{E}$. Processo similar é aplicado para os valores de campo magnético, calculando-se primeiramente as induções magnéticas $\vec{B}$ e depois $\vec{H}$.

Para a dedução das equações de campo magnético, parte-se das equações (3-78) e (3-78)) e, após tratamento ao do campo elétrico e usando:

$$B_x = \mu\ \frac{s_y}{s_x}H_x, \quad B_y = \mu\ \frac{s_z}{s_y}H_y, \quad B_z = \mu\ \frac{s_x}{s_z}H_z \quad (3\text{-}101)$$

resulta o conjunto de equações FDTD para $\vec{B}$ e $\vec{H}$:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$B_x\Big|_{i,j+1/2,k+3/2}^{n+1} = \left[\frac{2\ \epsilon k_z - \sigma_z\ \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right] Bx\Big|_{i,j+1/2,k+1/2}^{n+1/2} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_z + \sigma_z\ \Delta t)\Delta}\right] \left[Ey\Big|_{i,j+1/2,k+1}^{n+1} - Ey\Big|_{i,j+1/2,k}^{n+1} + Ez\Big|_{i,j,k+1/2}^{n+1} - Ez_{i,j+1,k+1/2}^{n+1}\right] \tag{3-102}$$

$$B_y\Big|_{i+1/2,j,k+1/2}^{n+3/2} = \left[\frac{2\ \epsilon k_x - \sigma_x\ \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right] By\Big|_{i+1/2,j,k+1/2}^{n+1/2} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_x + \sigma_x\ \Delta t)\Delta}\right] \left[Ez\Big|_{i+1,j,k+1/2}^{n+1} - Ez\Big|_{i,j,k+1/2}^{n+1} + Ex\Big|_{i+1/2,j,k}^{n+1} - Ez_{i+1/2,j,k+1}^{n+1}\right] \tag{3-103}$$

$$B_z\Big|_{i+1/2,j+1/2,k}^{n+3/2} = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right] Bz\Big|_{i+1/2,j+1/2,k}^{n+1/2} + \left[\frac{2\ \epsilon \Delta t}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\Delta}\right] \left[Ex\Big|_{i+1/2,j+1,k}^{n+1} - Ex\Big|_{i+1/2,j,k}^{n+1} + Ey\Big|_{i,j+1/2,k}^{n+1} - Ey_{i+1,j+1/2,k}^{n+1}\right] \tag{3-104}$$

$$H_x\Big|_{i,j+1/2,k+1/2}^{n+3/2} = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right] Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2} + \left[\frac{1}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\Delta}\right] \left[(2\ \epsilon k_x + \sigma_x\ \Delta t)Bx\Big|_{i,j,k+1/2}^{n+3/2} - (2\ \epsilon k_x - \sigma_x\ \Delta t)Bx\Big|_{i,j,k+1/2}^{n+1/2}\right] \tag{3-105}$$

$$H_y\Big|_{i+1/2,j,k+1/2}^{n+3/2} = \left[\frac{2\ \epsilon k_z - \sigma_z\ \Delta t}{2\ \epsilon k_z + \sigma_z\ \Delta t}\right] Hy\Big|_{i+1/2,j,k+1/2}^{n+1/2} + \left[\frac{1}{(2\ \epsilon k_z + \sigma_z\ \Delta t)\Delta}\right] \left[(2\ \epsilon k_y + \sigma_y\ \Delta t)By\Big|_{i+1/2,j,k+1/2}^{n+3/2} - (2\ \epsilon k_y - \sigma_y\ \Delta t)By\Big|_{i+1/2,j+1/2,k}^{n+1/2}\right] \tag{3-106}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$H_z\Big|_{i+1/2,j+1/2,k}^{n+3/2} = \left[\frac{2\ \epsilon k_x - \sigma_x\ \Delta t}{2\ \epsilon k_x + \sigma_x\ \Delta t}\right] Hz\Big|_{i+1/2,j+1/2,k}^{n+1/2} + \left[\frac{1}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\Delta}\right] \left[(2\ \epsilon k_z + \sigma_z\ \Delta t)Bz\Big|_{i+1/2,j+1/2,k}^{n+3/2} - (2\ \epsilon k_y - \sigma_y\ \Delta t)Bz\Big|_{i+1/2,j+1/2,k}^{n+1/2}\right] \quad (3\text{-}107)$$

$$H_x\Big|_{i,j+1/2,k+1/2}^{n+3/2} = \left[\frac{2\ \epsilon k_y - \sigma_y\ \Delta t}{2\ \epsilon k_y + \sigma_y\ \Delta t}\right] Hx\Big|_{i,j+1/2,k+1/2}^{n+1/2} + \left[\frac{1}{(2\ \epsilon k_y + \sigma_y\ \Delta t)\Delta}\right] \left[(2\ \epsilon k_y + \sigma_y\ \Delta t)Bx\Big|_{i+1/2,j,k+1/2}^{n+3/2} - (2\ \epsilon k_y - \sigma_y\ \Delta t)By\Big|_{i+1/2,j,k+1/2}^{n+1/2}\right] \quad (3\text{-}108)$$

Considerando as igualdades (3-92) - (3-94), reescrevem-se as equações (3-89) - (3-108) de forma a facilitar sua implementação computacional.

$$B_x\Big|_{i,j+1/2,k+3/2}^{n+1} = C1z\ Bx\Big|_{i,j+1/2,k+1/2}^{n+1/2} + C2z\ \left[Ey\Big|_{i,j+1/2,k+1}^{n+1} - Ey\Big|_{i,j+1/2,k}^{n+1} + Ez\Big|_{i,j,k+1/2}^{n+1} - Ez_{i,j+1,k+1/2}^{n+1}\right] \quad (3\text{-}109)$$

$$B_y\Big|_{i+1/2,j,k+1/2}^{n+3/2} = C1x\ By\Big|_{i+1/2,j,k+1/2}^{n+1/2} + C2x\ \left[Ez\Big|_{i+1,j,k+1/2}^{n+1} - Ez\Big|_{i,j,k+1/2}^{n+1} + Ex\Big|_{i+1/2,j,k}^{n+1} - Ez_{i+1/2,j,k+1}^{n+1}\right] \quad (3\text{-}110)$$

$$B_z\Big|_{i+1/2,j+1/2,k}^{n+3/2} = C1y\ Bz\Big|_{i+1/2,j+1/2,k}^{n+1/2} + C2y\ \left[Ex\Big|_{i+1/2,j+1,k}^{n+1} - Ex\Big|_{i+1/2,j,k}^{n+1} + Ey\Big|_{i,j+1/2,k}^{n+1} - Ey_{i+1,j+1/2,k}^{n+1}\right] \quad (3\text{-}111)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\left.H_x\right|_{i,j+1/2,k+1/2}^{n+3/2} = C1y \left.Hx\right|_{i,j+1/2,k+1/2}^{n+1/2} + C4hy \left[C5x \left.Bx\right|_{i,j,k+1/2}^{n+3/2} - C6x \left.Bx\right|_{i,j,k+1/2}^{n+1/2} \right] \quad (3\text{-}112)$$

$$\left.H_y\right|_{i+1/2,j,k+1/2}^{n+3/2} = C1z \left.Hy\right|_{i+1/2,j,k+1/2}^{n+1/2} + C4hz \left[C5y \left.By\right|_{i+1/2,j,k+1/2}^{n+3/2} - C6y \left.By\right|_{i+1/2,j+1/2,k}^{n+1/2} \right] \quad (3\text{-}113)$$

$$\left.H_z\right|_{i+1/2,j+1/2,k}^{n+3/2} = C1x \left.Hz\right|_{i+1/2,j+1/2,k}^{n+1/2} + C4hx \left[C5z \left.Bz\right|_{i+1/2,j+1/2,k}^{n+3/2} - C6z \left.Bz\right|_{i+1/2,j+1/2,k}^{n+1/2} \right] \quad (3\text{-}114)$$

A condutividade da UPML é determinada de acordo com um fator de crescimento, comumente polinomial, sendo que a condutividade máxima na ultima camada da condição absorvente é calculada por:

$$\sigma_{max} = -\frac{(m+1)\ ln[R(0)]}{2\eta d} \quad (3\text{-}115)$$

onde $3 \leq m \leq 4$,$R(0)$ é o fator de reflexão, $\nu = \sqrt{\nu/\epsilon}$ é a impedância intrínseca do meio, e d é a espessura da UPML em metros. Considerando uma espessura de N camadas, para $N = 10$ usa-se $R(0) = e^{-16}$ e, para $N = 5, R(0) = e^{-8}$. Estes valores são considerados ótimos para a maioria das aplicações FDTD. As demais condutividades são determinadas com o uso de (3-116) Considerando uma variação em x, de 0 até d, essas condutividades são dadas por:

$$\sigma_x(x) = \left(\frac{x}{d}\right)^m \ \sigma_{max} \quad (3\text{-}116)$$

os valores de k podem variar de 1 em $x = 0$ até um valor $k_{max} \geq 1$ sendo calculados conforme:

$$k_x(x) = 1 + (k_{max} - 1)\ (x/d)^m \quad (3\text{-}117)$$

sendo comum utilizar $k = 1$ para toda a UPML. As principais vantagens da UPML em relação à PML são a maior estabilidade, o fato de não ser sensíveis

PUC-Rio - Certificação Digital Nº 0310507/CB

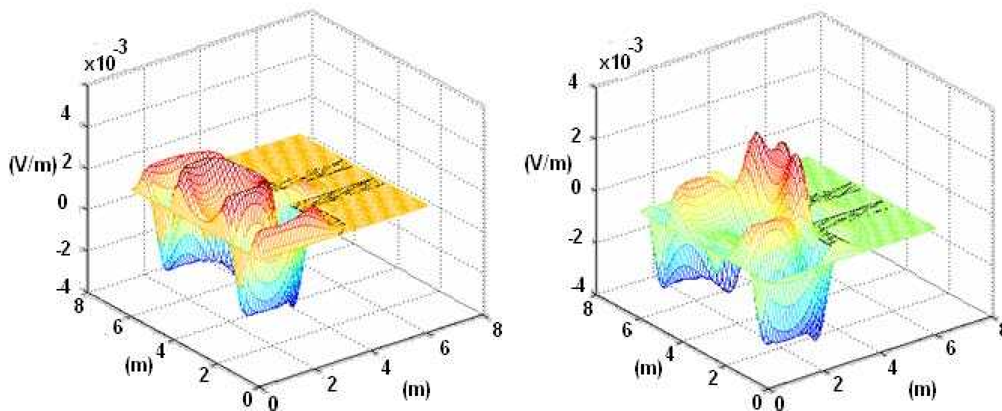

Figura 3.9: Pulso Gaussiano propagando-se no dominio computacional 3D rodeado por UPML

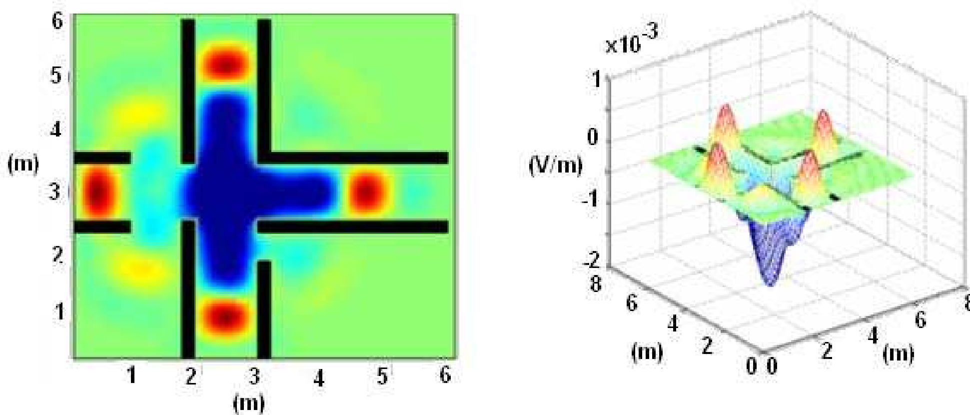

Figura 3.10: Pulso Gaussiano propagando-se no dominio computacional 3D rodeado por UPML, ambiente sem portas

PUC-Rio - Certificação Digital Nº 0310507/CB

a situações dependentes do tempo de simulação e parâmetros da malha, bem como uma maior facilidade de programação.

A figura (3.9) corresponde à propagação de um pulso gaussiano em um corredor 3D de paredes PEC, para diferentes passos temporais, vista no plano horizontal correspondente a $z = z_{fonte}$. Observa-se, por a fonte achar-se próxima da fronteira esquerda, a interação com a UPML nos primeiros passos temporais.

Na figura (3.10) apresenta-se outro teste da UPML, neste caso um cruzamento com algumas portas abertas e fonte no centro do cruzamento.

3.8 Fontes de excitação

3.8.1 Fontes suaves e rígidas

Fontes podem ser agrupadas em dois grandes grupos: fontes rígidas (hard sources) e fontes suaves (soft sources). Uma fonte rígida estabelece um conjunto de valores de campo, em um ou mais pontos de malha, dados por uma função temporal especificada como, por exemplo, em (3-118). Correspondendo a problemas eletromagnéticos nos quais o campo elétrico em alguns pontos é conhecido, e o objetivo é achar os valores dos campos radiantes em outros pontos. Uma das propriedades das fontes rígidas é que ondas propagando-se na direção da fonte são refletidas pela fonte.

$$E_z(i) = sen(2\pi f n \Delta t) \tag{3-118}$$

Para aplicações em que a análise deve ser feita ao longo de uma grande faixa de frequências, é comum a excitação por um pulso gaussiano (figura 3.11),cuja característica é que seu espectro de freqüência também apresenta comportamento gaussiano e, quanto mais estreita for a largura do pulso gaussiano (mais próximo a um impulso), mais largo será seu espectro de freqüência. A equação (3-119) representa um pulso gaussiano com componente dc.

$$f(t) = exp(-(t-t_0)^2/t_w^2) \tag{3-119}$$

onde t_w é a largura do pulso e t_0 o deslocamento da origem. Esta excitação deve começar suavemente com valores próximos de zero, evitando que variações bruscas causem oscilações indesejáveis. Para evitar estas oscilações usa-se um pulso duplo-gaussiano conforme ilustrado na figura (3.11).

Em [48] se apresenta uma função fonte de forma a evitar as componentes dc diferentes de zero, na forma:

$$f(t) = r(t) sen(\omega t)$$

$$r(t) = \begin{cases} 0 & t < 0 \\ 0.5[1 - cos(\omega t/(2\alpha))]) & 0 < t \leq \alpha T \\ 1 & t > \alpha T \end{cases} \tag{3-120}$$

onde T é o período da fonte harmônica no tempo e $\alpha = 1/2, 3/2, 5/2, ...$ Quando se está interessado em resultados para uma única freqüência, pode ser utilizado

PUC-Rio - Certificação Digital Nº 0310507/CB

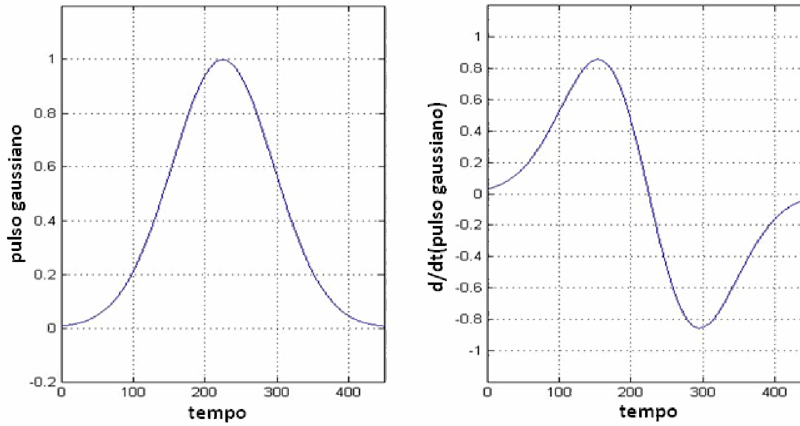

Figura 3.11: (a)Pulso Gaussiano com componente dc (b) Pulso duplo-gaussiano sem componente DC

um tipo de fonte conhecida como CW:

$$f(t) = \begin{cases} sen(2\pi ft)exp(-(t-t_0)^2/t_w^2 & \leq t_0 \\ sen(2\pi ft) & t_0 > t_0 \end{cases} \quad (3\text{-}121)$$

Em situções onde ondas refletidas podem alcançar a fonte, fontes de excitação suaves são necessárias. Para que uma onda refletida seja transparente à excitação, ou seja, não sofra excitações adicionais nos pontos de excitação condições como a proposta em [49] devem ser impostas tomamos por exemplo o caso unidimensional.

$$E_z(i) = sen(2\pi fn\Delta t) + E_z(i) \quad (3\text{-}122)$$

E_z é calculado normalmente no algoritmo e somado à senóide, resultando em um novo valor de Ez(i) de excitação. Essa excitação é suavizada porque se calcula o campo resultante no ponto de excitação, não sendo imposto forçadamente a cada iteração como na excitação rígida. Para o caso tridimensional, uma condição típica de fonte suave num plano ($y = j_s$), perto do plano da fonte (y=0), é:

$$E_z^n(i,j,k+1/2) = 1000sen(2\pi fn\Delta t) + E_z(i,j,k+1/2) \quad (3\text{-}123)$$

Em t=0, a frente de onda plana de freqüência f é considerada ativada. A propagação das ondas a partir desta fonte é simulada nos instantes seguintes ($n\Delta t$) em um processo cíclico considerado no algoritmo de Yee. A onda incidente é rastreada primeiro em direção do espalhador, interagindo com este via excitação de correntes superficiais, difusão, penetração e difração. A

PUC-Rio - Certificação Digital Nº 0310507/CB

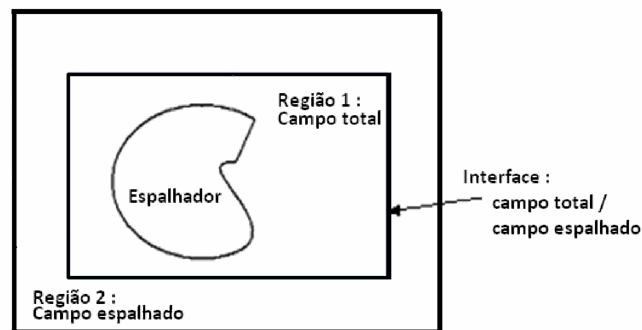

Figura 3.12: Divisão do domínio computacional em regiões: campo total / campo espalhado

evolução no tempo é feita até que um estado senoidal estacionário seja atingido.

3.8.2 Fonte: onda plana

Para simular uma fonte do tipo onda plana incidindo na malha FDTD, usa-se uma técnica chamada "campo total" / "campo espalhado" que, em resumo, consiste em: Inicialmente dividir o domínio computacional em uma região de campo total (região 1) interna e uma região de campo espalhado (região 2) externa, através de uma caixa virtual conhecida como caixa de Huygens (figura 3.12).

A região 1 é chamada região de campo total, composto da soma do campo incidente e do campo espalhado (usando a propriedade de linearidade das equações de Maxwell). O elemento espalhador de interesse encontra-se dentro desta região (fig 3.12). A região 2 externa à região 1 é denominada região de campo espalhado, onde se assume que só êste esteja presente.

A figura (3.13) ilustra esta técnica, para o caso de uma onda plana propagando-se dentro da região de campo total, em (a) para um ângulo de incidência de 90º e em (b), para um ângulo de incidência 0º. A região de campo espalhado acha-se entre a PML e a região de campo total.

Os planos exteriores que rodeiam a região 2, chamados de planos absorventes (PML), são usados para implementar as condições de radiação do espaço livre.

Nas faces da caixa de Huygens, correspondentes às fronteiras entre as duas regiões, e nas posições próximas às faces, os campos incidentes são definidos. As posições das faces são escolhidas de forma a coincidir com planos da malha onde os campos elétricos são definidos, e os campos magnéticos serão interpolados usando os vetores de campo magnético sobre o interior e o exterior da caixa virtual (Fig. 3.14).

PUC-Rio - Certificação Digital Nº 0310507/CB

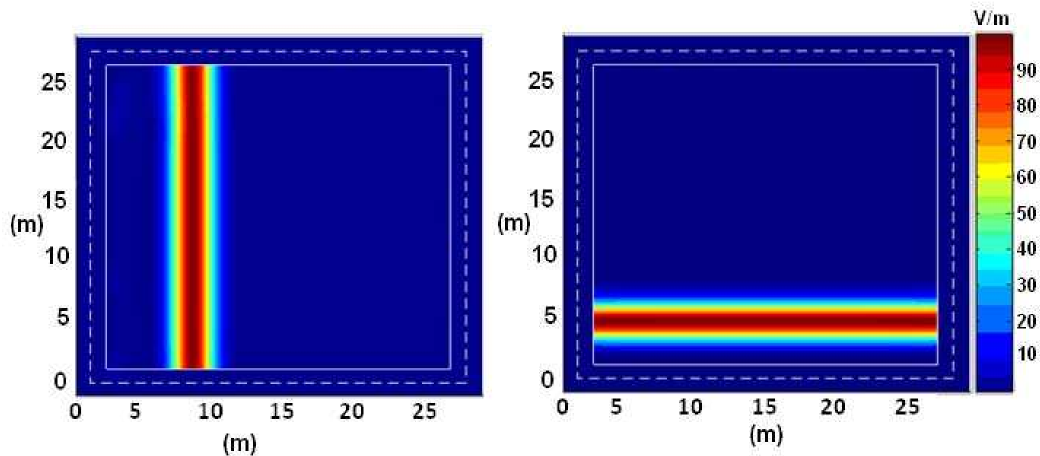

Figura 3.13: Onda plana propagando-se na região de campo total (a) ângulo de incidência 90º, (b) ângulo de incidência 0º

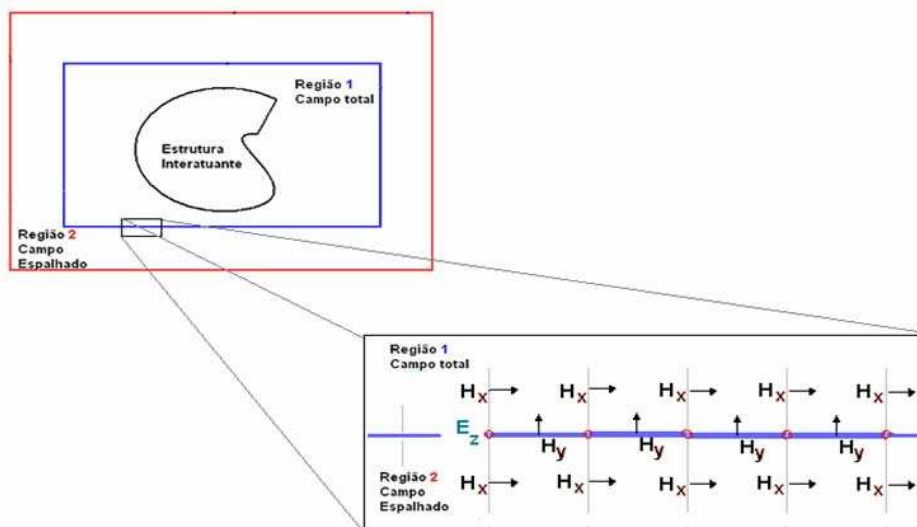

Figura 3.14: Condições de conexão típicas entre as regiões. Polarização TMz.

PUC-Rio - Certificação Digital Nº 0310507/CB

Formulação da técnica "Campo total/ Campo espalhado"

A técnica conhecida como: "Campo total / Campo espalhado" (TFSF) é comumente usada na modelagem de ondas planas. Considerando a fonte distante do espalhador (para distancias maiores que dez comprimentos de onda), o campo irradiado pela antena pode ser aproximado como uma onda plana [50], na maioria de problemas deste tipo o interesse esta em resultados em campo distante. A técnica TFSF pode ser usada para modelar pulso de longa duração ou iluminação senoidal para direções arbitrarias de propagação da onda.

Assume-se que o campo elétrico total e o campo magnético total podem ser descompostos na forma :

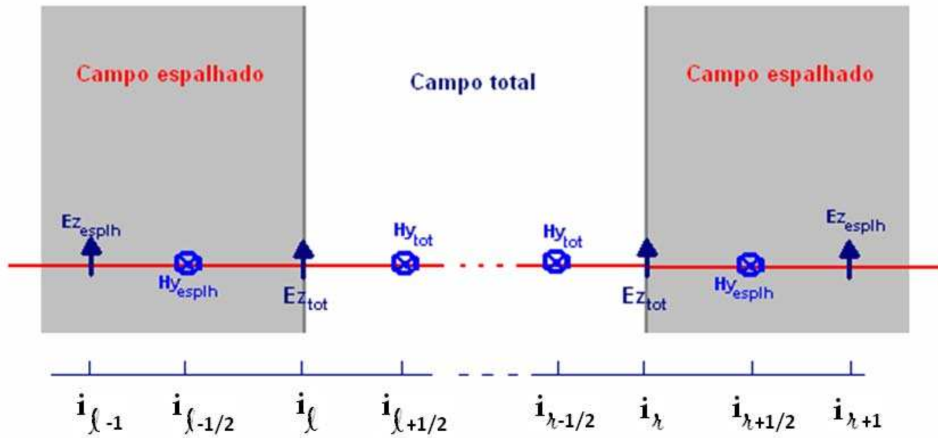

Figura 3.15: Aplicação da técnica TFSF numa malha unidimensional e a distribuição dos componentes de campo.

$$\vec{E}_{total} = \vec{E}_{inc} + \vec{E}_{esplh} \tag{3-124}$$
$$\vec{H}_{total} = \vec{H}_{inc} + \vec{H}_{esplh} \tag{3-125}$$

onde $\vec{E}_{total}(\vec{H}_{total})$ é o campo elétrico (magnético) total,$\vec{E}_{inc}(\vec{H}_{inc})$ é o campo elétrico (magnético) incidente, e $\vec{E}_{esplh}(\vec{H}_{esplh})$ é o campo elétrico (magnético) espalhado. Assumem-se conhecidos os valores dos campos incidentes na interface TFSF e em pontos próximos e para toda a evolução temporal. Está superfície de conexão (virtual) tem forma retangular fixa quando se considera o caso 2D e a forma de um cubo em 3D independente da geometria e das características constitutivas do espalhador.

Formulação TFSF-1D

Considere-se o caso da aplicação desta técnica em problemas unidimensionais, no espaço livre, vamos assumindo uma malha uniforme com componentes de campo Ez e Hy. A equação de atualização em diferenças para esta componente, segundo o algoritmo de Yee, é dado por:

$$E_z\Big|_i^{n+1} = E_z\Big|_i^{n} + \frac{\Delta t}{\epsilon_0 \Delta x}\left(H_y\Big|_{i+1/2}^{n+1/2} - H_y\Big|_{i-1/2}^{n+1/2}\right) \tag{3-126}$$

e, já que todas as componentes se acham em uma mesma região, a equação (3-126) é consistente considere-se agora, a situação apresentada na figura (3.15) em que se aplica a técnica TFSF.

O cálculo da componente $Ez_{tot}\Big|_{il}$ para esta nova distribução seguindo (3-126), será:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$E_{z,total}\Big|_{il}^{n+1} = E_{z,total}\Big|_{i}^{n} + \frac{\Delta t}{\epsilon_0 \Delta x}\left(H_{y,tot}\Big|_{il+1/2}^{n+1/2} - H_{y,esplh}\Big|_{il-1/2}^{n+1/2}\right) \quad (3\text{-}127)$$

observando-se que o ultimo componente de ((3-127)) quebra a consistência da equação, esta componente pertence ao campo espalhado e as outras componentes ao campo total, este problema é resolvido fazendo uso da linearidade das equações de Maxwell (3-126). A equação corrigida para atualização do campo elétrico na posição i_L passa a ser :

$$E_{z,total}\Big|_{il}^{n+1} = E_{z,total}\Big|_{i}^{n} + \frac{\Delta t}{\epsilon_0 \Delta x}\left(H_{y,tot}\Big|_{il+1/2}^{n+1/2} - H_{y,esplh}\Big|_{il-1/2}^{n+1/2}\right) - \frac{\Delta t}{\epsilon_0 \Delta x} H_{y,inc}\Big|_{il-1/2}^{n+1/2} \quad (3\text{-}128)$$

desde que:

$$-H_{y,esplh}\Big|_{il-1/2}^{n+1/2} - -H_{y,inc}\Big|_{il-1/2}^{n+1/2} = -H_{y,tot}\Big|_{il-1/2}^{n+1/2}$$

Observe-se que o termo adicional em (3-128) é uma correção, que será aplicada apenas nos pontos de fronteira entre as duas regiões, e o algoritmo de Yee continua sendo aplicado em todo o domínio computacional.

De forma semelhante, para a componente de campo magnético na posição $i_{l-1/2}$ segue:

$$H_{y,esplh}\Big|_{il-1/2}^{n+1/2} = H_{y,total}\Big|_{il-1/2}^{n-1/2} + \frac{\Delta t}{\mu_0 \Delta x}\left(E_{z,tot}\Big|_{il}^{n} - E_{z,esplh}\Big|_{il-1}^{n}\right) - \frac{\Delta t}{\mu_0 \Delta x} E_{z,inc}\Big|_{il}^{n} \quad (3\text{-}129)$$

As equações de atualização corrigidas para a interface TFSF no lado direito da fig 3.15) são:

$$E_{z,tot}\Big|_{iR}^{n+1} = E_{z,total}\Big|_{iR}^{n} + \frac{\Delta t}{\epsilon_0 \Delta x}\left(H_{y,esplh}\Big|_{iR+1/2}^{n+1/2} - H_{y,tot}\Big|_{iR-1/2}^{n+1/2}\right) + \frac{\Delta t}{\epsilon_0 \Delta x} H_{y,inc}\Big|_{iR+1/2}^{n+1/2} \quad (3\text{-}130)$$

$$H_{y,esplh}\Big|_{iR+1/2}^{n+1/2} = H_{y,esplh}\Big|_{iR+1/2}^{n-1/2} + \frac{\Delta t}{\mu_0 \Delta x}\left(E_{z,esplh}\Big|_{iR+1}^{n} - E_{z,tot}\Big|_{iR}^{n}\right) + \frac{\Delta t}{\mu_0 \Delta x} E_{z,inc}\Big|_{iR}^{n} \quad (3\text{-}131)$$

Formulação TFSF-2D

A técnica TFSF pode ser estendida para problemas bidimensionais, de forma a trabalhar com polarizações TM e TE. A onda plana gerada por esta técnica tem forma arbitrária no tempo e duração, assim como direção de propagação no domínio computacional. Consideremos a fig.3.16 correspondente ao caso TMz:

PUC-Rio - Certificação Digital Nº 0310507/CB

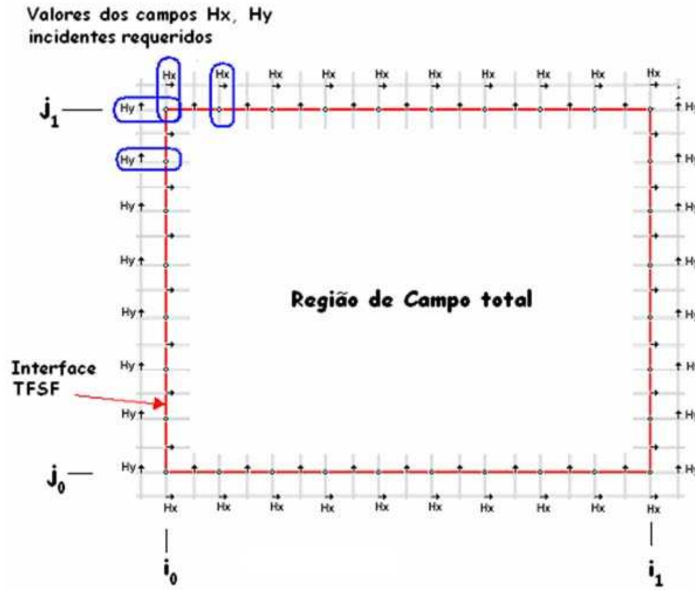

Figura 3.16: Detalhes da interface campo total campo espalhado.

PUC-Rio - Certificação Digital Nº 0310507/CB

Na figura (3.16) observa-se que a interface entre as componentes de campo espalhado e total tem forma retangular e cada face será tratada separadamente, para garantir a consistência das equações de atualização.

Face FRONT

As componentes Ez da face "front" $(i = i_0, \ldots, i_1; j = j_0)$ são:

$$E_z\Big|_{i,j_0}^{n+1} = \left\{ E_z\Big|_{i,j_0}^{n+1} \right\}_{equacao\ (3\text{-}26)} + \frac{\Delta t}{\epsilon_0 \Delta} H_{x,inc}\Big|_{i,j_0-1/2}^{n+1/2} \tag{3-132}$$

Face BACK

As componentes Ez da face "back" $(i = i_0, \ldots, i_1; j = j_1)$ são:

$$E_z\Big|_{i,j1}^{n+1} = \left\{ E_z\Big|_{i,j1}^{n+1} \right\}_{equacao\ (3\text{-}26)} - \frac{\Delta t}{\epsilon_0 \Delta} H_{x,inc}\Big|_{i,j1+1/2}^{n+1/2} \tag{3-133}$$

Face LEFT

As componentes Ez da face "left" $(i = i_0; j = j_0 \ldots, j_1)$ são:

$$E_z\Big|_{i_0,j}^{n+1} = \left\{ E_z\Big|_{i_0,j}^{n+1} \right\}_{equacao\ (3\text{-}26)} - \frac{\Delta t}{\epsilon_0 \Delta} H_{y,inc}\Big|_{i_0-1/2,j2}^{n+1/2} \tag{3-134}$$

Face RIGHT

As componentes Ez da face "right" $(i = i_1; j = j_0 \ldots, j_1)$ são:

$$E_z\Big|_{i_0,j}^{n+1} = \left\{ E_z\Big|_{i1,j}^{n+1} \right\}_{equacao\ (3\text{-}26)} + \frac{\Delta t}{\epsilon_0 \Delta} H_{y,inc}\Big|_{i1+1/2,j}^{n+1/2} \tag{3-135}$$

Da figura (3.16) observa-se que, precisando dos valores dos campos incidentes Hx e Hy e que estas componentes estando posicionadas a 0.5Δ fora da interface, na região de campo espalhado, precisa-se também garantir a consistência para estas componentes.

Elementos externos à face "FRONT"

As componentes Hx da face "front" $(i = i_0, \ldots, i_1; j = j_0 - 1/2)$ são:

$$H_x\Big|_{i,j_0-1/2}^{n+1/2} = \left\{ H_x\Big|_{i,j_0-1/2}^{n+1/2} \right\}_{equacao\ (3\text{-}24)} + \frac{\Delta t}{\mu_0 \Delta} E_{z,inc}\Big|_{i,j_0}^{n} \tag{3-136}$$

Elementos externos à face "BACK"

As componentes Hx da face "back" $(i = i_0, \ldots, i_1; j = j_1 + 1/2)$ são:

$$H_x\Big|_{i,j1-1/2}^{n+1/2} = \left\{ H_x\Big|_{i,j1-1/2}^{n+1/2} \right\}_{equacao\ (3\text{-}24)} - \frac{\Delta t}{\mu_0 \Delta} E_{z,inc}\Big|_{i,j1}^{n} \tag{3-137}$$

Elementos externos à face "LEFT"

As componentes Hy da face "left" $(i = i_0 - 1/2; j = j_0, \ldots, j_1)$ são:

$$H_y\Big|_{i_0-1/2,j}^{n+1/2} = \left\{ H_y\Big|_{i_0-1/2,j}^{n+1/2} \right\}_{equacao\ (3\text{-}24)} - \frac{\Delta t}{\mu_0 \Delta} E_{z,inc}\Big|_{i_0,j}^{n} \tag{3-138}$$

Elementos externos à face "RIGHT"

As componentes Hy da face "right" $(i = i_1 + 1/2; j = j_0, \ldots, j_1)$ são:

$$H_y\Big|_{i1+1/2,j}^{n+1/2} = \left\{ H_y\Big|_{i1+1/2,j}^{n+1/2} \right\}_{equacao\ (3\text{-}24)} + \frac{\Delta t}{\mu_0 \Delta} E_{z,inc}\Big|_{i1,j}^{n} \tag{3-139}$$

As equações (3-132)-(3-139) precisam, para sua implementação, dos valores dos campos elétrico e magnético referentes a uma onda incidindo sob ângulos arbitrários. Em [37] é apresentada detalhadamente, uma técnica otimizada (usa uma tabela de equivalências visando reduzir o numero de cálculos repetitivos) para gerar esta onda incidente. Considerando, por exemplo, uma onda plana incidente com polarização TMz, segundo a direção ϕ_{inc} (figura 3.17), o primeiro contato da onda com a interface TFSF é produzido na quina inferior esquerda. Considerando, um segmento que passe por este ponto de contato com esta direção e atravesse a região de campo total, pode-se implementar, neste segmento, uma malha FDTD-1D onde o tamanho do passo é o mesmo que na malha 2D, no inicio desta malha 1D, insere-se uma fonte e se calcula os campos incidentes propagando-se nesta malha. Dado que os campos incidentes na malha 2D têm a mesma fase, é possível obter o campo incidente sobre qualquer ponto do contorno, projetando o ponto onde se precisa do valor do

PUC-Rio - Certificação Digital Nº 0310507/CB

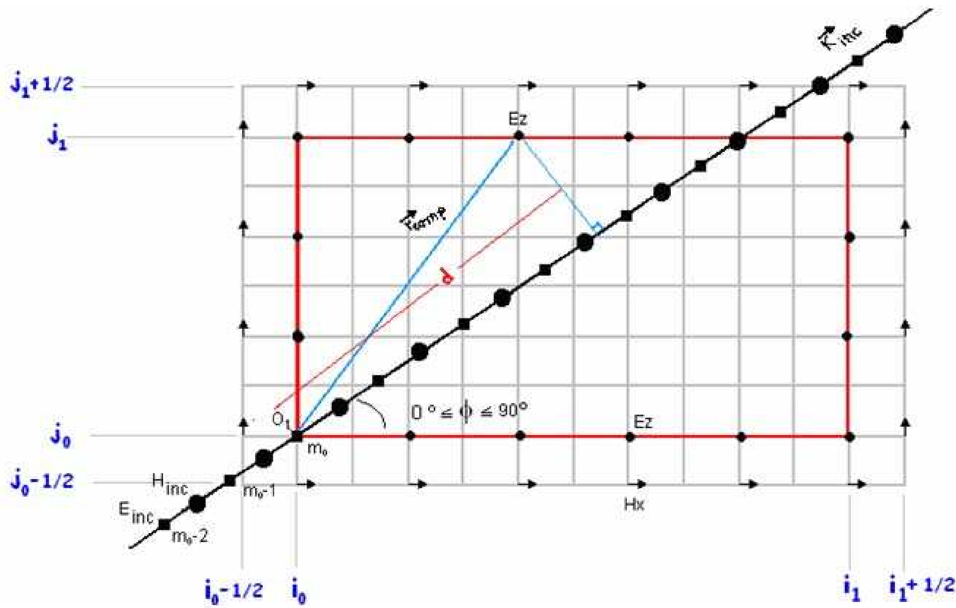

Figura 3.17: Cálculo eficiente do campo incidente para polarização TMz-2D

PUC-Rio - Certificação Digital Nº 0310507/CB

campo sobre o segmento. Em geral, a projeção sobre o segmento não coincidirá com a posição do valor do campo de interesse e se usa uma interpolação linear entre valores vizinhos. Conseqüentemente, gerando uma tabela de valores ao longo do segmento, é possível achar o valor do campo incidente no contorno sem precisar repetir os cálculos (figura 3.17).
Observemos na figura (3.17) que $O1$ é o primeiro ponto na região correspondente ao campo total que entra em contato com a frente de onda incidente. Para uma malha cartesiana, com elementos de malha quadrados, as outras componentes de campo na região de campo total são contactadas pela frente de onda depois de n_{delay} passos no tempo [37]:

$$n_{delay} = \frac{d\,\Delta}{[\nu_P(\phi)\Delta t]} \tag{3-140}$$

onde $\nu_P(\phi)$ é a velocidade de fase da onda incidente segundo o ângulo ϕ de propagação e "d" é a distancia do ponto O_1 à projeção vertical sobre o vetor de onda, podendo ser expressa como:

$$d = \hat{k}_{inc} \cdot \vec{r}_{comp} \tag{3-141}$$

onde $\hat{k}_{inc}$ é o vetor posição de O_1 na posição da componente de campo de interesse.

$$\hat{r}_{comp} = (i_{comp} - i_0)\hat{x} + (j_{comp} - j_0)\hat{y} \tag{3-142}$$

Considerando outros ângulos de incidencia, a frente de onda incidente irá fazer contato inicial em três novos pontos, representando as restantes três quinas da região de campo total (figura ...). Para cada um destes novos pontos

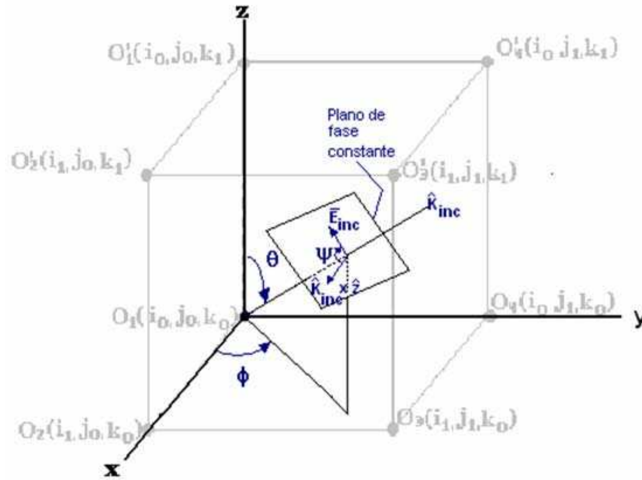

Figura 3.18: Propagação da onda incidente: direção e polarização

o vetor $\hat{r}_{comp}$ deverá ser modificado como segue:

– origem: $O_2(i_1, j_0)$

$$90^o < \phi \leq 180^o \qquad \hat{r}_{comp} = (i_{comp} - i_1)\hat{x} + (j_{comp} - j_0)\hat{y} \qquad (3\text{-}143)$$

– origem: $O_3(i_1, j_1)$

$$180^o < \phi \leq 270^o \qquad \hat{r}_{comp} = (i_{comp} - i_1)\hat{x} + (j_{comp} - j_1)\hat{y} \qquad (3\text{-}144)$$

– origem: $O_4(i_0, j_1)$

$$270^o < \phi \leq 360^o \qquad \hat{r}_{comp} = (i_{comp} - i_0)\hat{x} + (j_{comp} - j_1)\hat{y} \qquad (3\text{-}145)$$

Formulação TFSF-3D

As idéias apresentadas para a técnica TFSF-1D e 2D podem ser generalizadas para o espaço tridimensional. Para definir a direção e a polarização da onda plana incidente, usa-se um sistema de coordenadas esférico (figura (3.19)), o vetor de onda incidente $\vec{k}_{inc}$ é orientado segundo um ângulo $0^o < \phi \leq 180^o$ relativo ao eixo "+z" da malha e um ângulo $0^o < \phi \leq 360^o$ relativo ao eixo "+x". Para especificar a polarização da onda incidente, define-se um vetor de referencia $\hat{k}_{inc} \times \hat{z}$ no plano da frente de onda (plano de fase constante)incidente e um ângulo $0^o < \psi \leq 360^o$ de orientação do vetor campo elétrico incidente $\vec{E}_{inc}$ relativo a seu vetor refêrencia. Esta forma de especificar a polarização do campo elétrico é usada para todos os casos de incidência da onda, exceto para $\theta = 0^o$ e $\theta = 180^o$, onde ϕ pode ser usado para descrever a orientação de $\vec{E}_{inc}$ relativa ao eixo "+x".

PUC-Rio - Certificação Digital Nº 0310507/CB

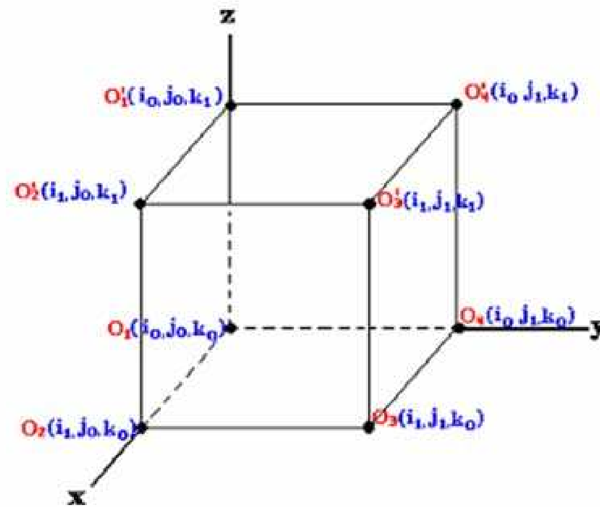

Figura 3.19: Interface TFSF-3D

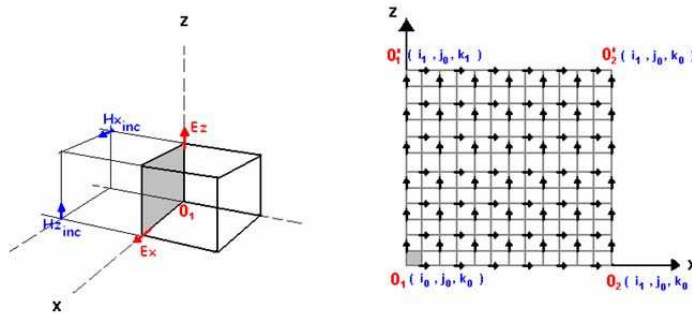

Figura 3.20: Componentes $Ex(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $j = j_0$ no plano XZ. da figura 3.19

Considerando a figura (3.20), correspondente à interface TFSF em 3D, as fronteiras entre as regiões de campo total e campo espalhado são faces que formam um cubo de vértices opostos (i_0, j_0, k_0) e (i_1, j_1, k_1).
Cada face do cubo tem duas componentes de campo elétrico tangenciais cujo tipo e localização são especificado nas figuras (3.21a)-(3.21f).

Face j $= j_0$ As equações que garantem a consistência são:

$$Para \quad Ex(i = i_0 + 1/2, \ldots, i_1 - 1/2; j = j_0; k = k_0, \ldots, k_1);$$
$$Ex\Big|_{i,j_0,k}^{n+1} = \left\{ Ex\Big|_{i,j_0,k}^{n+1} \right\}_{Eq.(3-16)} - Cb(m) H_{z,inc}\Big|_{i,j_0-1/2,k}^{n+1/2} \tag{3-146}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

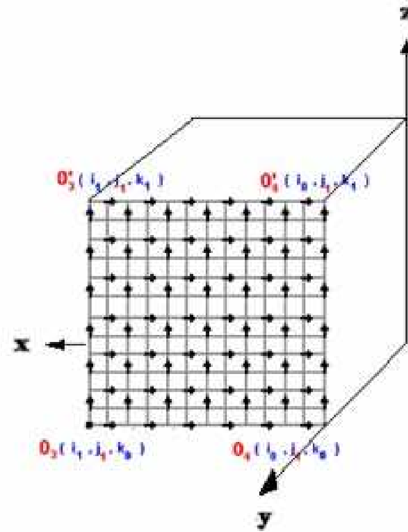

Figura 3.21: Componentes $Ex(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $j = j1$ no plano XZ. da figura 3.19

PUC-Rio - Certificação Digital Nº 0310507/CB

$$Para \quad Ez(i = i_0, \ldots, i_1; j = j_0; k = k_0 + 1/2, \ldots, k_1 - 1/2) :$$
$$Ez\Big|_{i,j_0,k}^{n+1} = \left\{ Ez\Big|_{i,j_0,k}^{n+1} \right\}_{Eq.(3-18)} + Cb(m)H_{x,inc}\Big|_{i,j_0-1/2,k}^{n+1/2} \tag{3-147}$$

Face j $= j_1$ As equações que garantem a consistência são:

$$Para \quad Ex(i = i_0 + 1/2, \ldots, i_1 - 1/2; j = j_1; k = k_0, \ldots, k_1);$$
$$Ex\Big|_{i,j_1,k}^{n+1} = \left\{ Ex\Big|_{i,j_1,k}^{n+1} \right\}_{Eq.(3-16)} + Cb(m)H_{z,inc}\Big|_{i,j1+1/2,k}^{n+1/2} \tag{3-148}$$

$$Para \quad Ez(i = i_0, \ldots, i_1; j = j_1; k = k_0 + 1/2, \ldots, k_1 - 1/2) :$$
$$Ez\Big|_{i,j_1,k}^{n+1} = \left\{ Ez\Big|_{i,j_1,k}^{n+1} \right\}_{Eq.(3-18)} - Cb(m)H_{x,inc}\Big|_{i,j1+1/2,k}^{n+1/2} \tag{3-149}$$

Face k $= k_0$ As equações que garantem a consistência são:

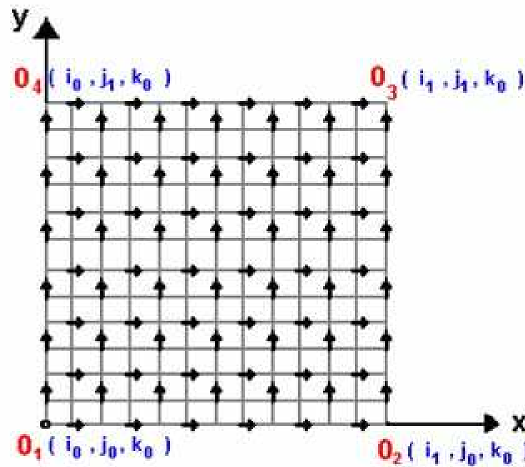

Figura 3.22: Componentes $Ex(\rightarrow)$ e $Ey(\uparrow)$ sôbre a face $k = k_0$ no plano XZ. da figura 3.19

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\begin{aligned} Para \quad & Ex(i = i_0 + 1/2, \ldots, i_1 - 1/2; j = j_0, \ldots, j_1; k = k_0); \\ & Ex\Big|_{i,j,k_0}^{n+1} = \left\{ Ex\Big|_{i,j,k_0}^{n+1} \right\}_{Eq.(3-16)} + Cb(m)H_{y,inc}\Big|_{i,j,k_0-1/2}^{n+1/2} \end{aligned} \tag{3-150}$$

$$\begin{aligned} Para \quad & Ey(i = i_0, \ldots, i_1; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_0) : \\ & Ey\Big|_{i,j,k_0}^{n+1} = \left\{ Ey\Big|_{i,j,k_0}^{n+1} \right\}_{Eq.(3-18)} - Cb(m)H_{z,inc}\Big|_{i,j,k-1/2}^{n+1/2} \end{aligned} \tag{3-151}$$

As equações que garantem a consistência são:

$$\begin{aligned} Para \quad & Ex(i = i_0 + 1/2, \ldots, i_1 - 1/2; j = j_0, \ldots, j_1; k = k_1); \\ & Ex\Big|_{i,j,k_1}^{n+1} = \left\{ Ex\Big|_{i,j,k_1}^{n+1} \right\}_{Eq.(3-16)} - Cb(m)H_{y,inc}\Big|_{i,j,k_1+1/2}^{n+1/2} \end{aligned} \tag{3-152}$$

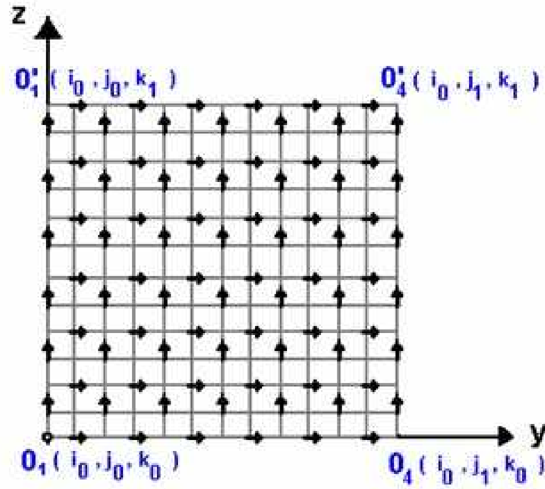

Figura 3.23: Componentes $Ey(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $k = i_0$ no plano YZ. da figura 3.19

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\begin{aligned} &Para \quad Ey(i = i_0, \ldots, i_1; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_1): \\ &Ey\Big|_{i,j,k_1}^{n+1} = \left\{ Ey\Big|_{i,j,k_1}^{n+1} \right\}_{Eq.(3-18)} + Cb(m) H_{x,inc}\Big|_{i,j,k+1/2}^{n+1/2} \end{aligned} \tag{3-153}$$

Face i $= i_0$ As equações que garantem a consistência são:

$$\begin{aligned} &Para \quad Ey(i = i_0; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_0, \ldots, k1); \\ &Ey\Big|_{i_0,j,k}^{n+1} = \left\{ Ey\Big|_{i_0,j,k}^{n+1} \right\}_{Eq.(3-17)} + Cb(m) H_{z,inc}\Big|_{i_0-1/2,j,k}^{n+1/2} \end{aligned} \tag{3-154}$$

$$\begin{aligned} &Para \quad Ez(i = i_0; j = j_0, \ldots, j_1; k = k_0 + 1/2, \ldots, k_1 - 1/2): \\ &Ez\Big|_{i_0,j,k}^{n+1} = \left\{ Ez\Big|_{i_0,j,k}^{n+1} \right\}_{Eq.(3-18)} - Cb(m) H_{y,inc}\Big|_{i_0-1/2,j,k}^{n+1/2} \end{aligned} \tag{3-155}$$

Face i $= i_1$ As equações que garantem a consistência são:

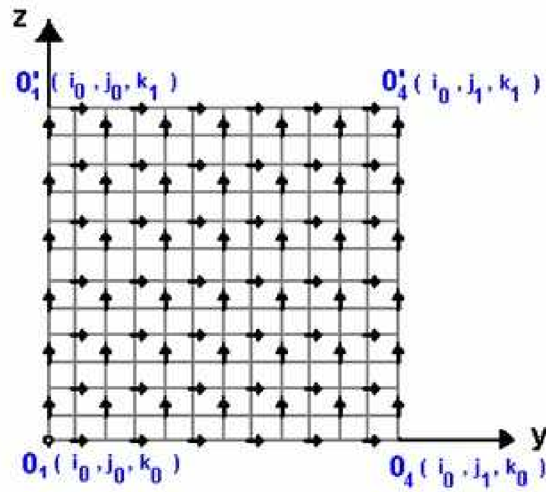

Figura 3.24: Componentes $Ey(\rightarrow)$ e $Ez(\uparrow)$ sôbre a face $i = i_1$ no plano YZ. da figura 3.19

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\begin{aligned} &Para \quad Ey(i = i_1; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_0, \ldots, k1); \\ &Ey\Big|_{i_1,j,k}^{n+1} = \left\{ Ey\Big|_{i_1,j,k}^{n+1} \right\}_{Eq.(3-17)} - Cb(m)H_{z,inc}\Big|_{i_1-1/2,j,k}^{n+1/2} \end{aligned} \tag{3-156}$$

$$\begin{aligned} &Para \quad Ez(i = i_1; j = j_0, \ldots, j_1; k = k_0 + 1/2, \ldots, k_1 - 1/2) : \\ &Ez\Big|_{i_1,j,k}^{n+1} = \left\{ Ez\Big|_{i_1,j,k}^{n+1} \right\}_{Eq.(3-18)} + Cb(m)H_{y,inc}\Big|_{i_1+1/2,j,k}^{n+1/2} \end{aligned} \tag{3-157}$$

As condições de conexão para as componentes de campo magnético localizadas a $1/2\Delta$ da interface Campo total/Campo espalhado, anàlogamente ao modo TM-2D, são dadas por:

$\mathbf{j} = j_0 - 1/2$ As equações que garantem a consistência são:

$$\begin{aligned} &Para \quad Hz(i = i_0 + 1/2; \ldots, i_1 - 1/2; j = j_1 + 1/2; k = k_0, \ldots, k1); \\ &Hz\Big|_{i,j_1+1/2,k}^{n+1/2} = \left\{ Hz\Big|_{i,j_1+1/2,k}^{n+1/2} \right\}_{Eq.(3-15)} + Db(m)E_{x,inc}\Big|_{i,j_1,k}^{n} \end{aligned} \tag{3-158}$$

$$Para \quad Hx(i=i_0;\ldots,i_1;j=j_1+1/2;k=k_0+1/2,\ldots,k_1-1/2):$$
$$Hx\Big|_{i,j_1+1/2,k}^{n+1/2} = \left\{ Hx\Big|_{i,j_1+1/2,k}^{n+1/2} \right\}_{Eq.(3-13)} - Db(m)E_{z,inc}\Big|_{i,j_1,k}^{n} \tag{3-159}$$

k $= k_0 - 1/2$ As equações que garantem a consistência são:

$$Para \quad Hy(i=i_0+1/2;\ldots,i_1-1/2;j=j_0,\ldots,j_1;k=k_0-1/2);$$
$$Hy\Big|_{i,j,k_0-1/2}^{n+1/2} = \left\{ Hz\Big|_{i,j,k_0-1/2}^{n+1/2} \right\}_{Eq.(3-14)} + Db(m)E_{x,inc}\Big|_{i,j,k_0}^{n} \tag{3-160}$$

$$Para \quad Hx(i=i_0;\ldots,i_1;j=j_0+1/2,\ldots,j_1-1/2;k=k_0-1/2):$$
$$Hx\Big|_{i,j,k_0-1/2}^{n+1/2} = \left\{ Hx\Big|_{i,j,k_0-1/2}^{n+1/2} \right\}_{Eq.(3-13)} - Db(m)E_{y,inc}\Big|_{i,j,k_0}^{n} \tag{3-161}$$

$= k_1 + 1/2$ As equações que garantem a consistência são:

$$Para \quad Hy(i=i_0+1/2;\ldots,i_1-1/2;j=j_0,\ldots,j_1;k=k_1-1/2);$$
$$Hy\Big|_{i,j,k_1+1/2}^{n+1/2} = \left\{ Hy\Big|_{i,j,k_1+1/2}^{n+1/2} \right\}_{Eq.(3-14)} - Db(m)E_{x,inc}\Big|_{i,j,k_1}^{n} \tag{3-162}$$

$$Para \quad Hx(i=i_0;\ldots,i_1;j=j_0+1/2,\ldots,j_1-1/2;k=k_1+1/2):$$
$$Hx\Big|_{i,j,k_1+1/2}^{n+1/2} = \left\{ Hx\Big|_{i,j,k_1+1/2}^{n+1/2} \right\}_{Eq.(3-13)} + Db(m)E_{y,inc}\Big|_{i,j,k_1}^{n} \tag{3-163}$$

i $= i_0 - 1/2$ As equações que garantem a consistência são:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$Para \quad Hz(i = i_0 - 1/2; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_0, \ldots, k_1);$$
$$Hz\Big|_{i_0-1/2,j,k}^{n+1/2} = \left\{ Hy\Big|_{i_0-1/2,j,k}^{n+1/2} \right\}_{Eq.(3-15)} + Db(m)E_{y,inc}\Big|_{i_0,j,k}^{n} \tag{3-164}$$

$$Para \quad Hy(i = i_0 - 1/2; j = j_0, \ldots, j_1; k = k_0 + 1/2, \ldots k_1 - 1/2):$$
$$Hy\Big|_{i_0-1/2,j,k}^{n+1/2} = \left\{ Hy\Big|_{i_0-1/2,j,k}^{n+1/2} \right\}_{Eq.(3-14)} - Db(m)E_{z,inc}\Big|_{i_0,j,k}^{n} \tag{3-165}$$

$\mathbf{i} = i_1 + 1/2$ As equações que garantem a consistência são:

$$Para \quad Hz(i = i_1 + 1/2; j = j_0 + 1/2, \ldots, j_1 - 1/2; k = k_0, \ldots, k_1);$$
$$Hz\Big|_{i_1+1/2,j,k}^{n+1/2} = \left\{ Hz\Big|_{i_1+1/2,j,k}^{n+1/2} \right\}_{Eq.(3-15)} - Db(m)E_{y,inc}\Big|_{i_1,j,k}^{n} \tag{3-166}$$

$$Para \quad Hy(i = i_1 + 1/2; j = j_0, \ldots, j_1; k = k_0 + 1/2, \ldots k_1 - 1/2):$$
$$Hy\Big|_{i_1+1/2,j,k}^{n+1/2} = \left\{ Hy\Big|_{i_1+1/2,j,k}^{n+1/2} \right\}_{Eq.(3-14)} + Db(m)E_{z,inc}\Big|_{i_1,j,k}^{n} \tag{3-167}$$

Origem de coordenadas

Observando a figura 3.19, a interface-3D entre as regiões de campo total /campo espalhado apresenta oito possíveis pontos de contato com as frentes de ondas incidentes para a faixa de variação dos ângulos θ e ϕ. Analogamente ao caso TM-2D, estes pontos de contato inicial são, na realidade origens de coordenadas para o cálculo da distancia de retardo “d” necessária para obter os campos da onda incidente para as condições de conexão em (3-146)-(3-167). Seguindo a notação estabelecida para o caso TM, a distancia “d” para o caso tridimensional é novamente dada por:

$$d = \hat{k}_{inc} \cdot \vec{r}_{comp} \tag{3-168}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

aqui $\hat{k}_{inc}$ é um vetor de onda unitário incidente dado por:

$$\hat{k}_{inc} = \hat{x}\ sin\ \theta cos\phi + \hat{y}\ sin\theta\ sin\phi + \hat{z}\ cos\theta \tag{3-169}$$

e $\vec{r}_{comp}$ é um vetor posição desde uma origem apropriada ao ponto onde se calcula o campo de interesse.

Para $0^o \leq \theta \leq 90^o$ Os pontos de contato inicial são: O_1, O_2, O_3 e O_4

$$O_1(i_0, j_0, k_0), 0^o \leq \phi \leq 90^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_0) + \hat{y}(j_{comp} - j_0) + \hat{z}(k_{comp} - k_0) \tag{3-170}$$

$$O_2(i_1, j_0, k_0), 90^o \leq \phi \leq 180^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_1) + \hat{y}(j_{comp} - j_0) + \hat{z}(k_{comp} - k_0) \tag{3-171}$$

$$O_3(i_1, j_1, k_0), 180^o \leq \phi \leq 270^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_1) + \hat{y}(j_{comp} - j_0) + \hat{z}(k_{comp} - k_0) \tag{3-172}$$

$$O_4(i_0, j_1, k_0), 270^o \leq \phi \leq 360^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_0) + \hat{y}(j_{comp} - j_1) + \hat{z}(k_{comp} - k_0) \tag{3-173}$$

Para $90^o \leq \theta \leq 180^o$ Os pontos de contato inicial são: O_1', O_2', O_3' e O_4'

$$O_1'(i_0, j_0, k_1), 0^o \leq \phi \leq 90^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_0) + \hat{y}(j_{comp} - j_0) + \hat{z}(k_{comp} - k_1) \tag{3-174}$$

$$O_2'(i_1, j_0, k_1), 90^o \leq \phi \leq 180^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_1) + \hat{y}(j_{comp} - j_0) + \hat{z}(k_{comp} - k_1) \tag{3-175}$$

$$O_3'(i_1, j_1, k_1), 180^o \leq \phi \leq 270^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_1) + \hat{y}(j_{comp} - j_1) + \hat{z}(k_{comp} - k_1) \tag{3-176}$$

$$O_4'(i_0, j_1, k_0), 270^o \leq \phi \leq 360^o$$

$$\vec{r}_{comp} = \hat{x}(i_{comp} - i_0) + \hat{y}(j_{comp} - j_1) + \hat{z}(k_{comp} - k_1) \tag{3-177}$$

Escolhido os ângulos, pode-se calcular a distancia “d” usando (3-168)

Geração e interpolação de dados de uma tabela look-up e cálculos finais das componentes de campo incidente

Como no caso bidimensional, um vetor unidimensional auxiliar pode ser usado para gerar a tabela “look-up”, para as variações espacio temporal dos

PUC-Rio - Certificação Digital Nº 0310507/CB

campos incidentes no caso 3D. A malha unidimensional é colocada ao longo do vetor de onda incidente de forma que uma origem apropriada ($O_1 - O_4, O'_1 - O'_4$) da interface entre as regiões de campo total / campo espalhado coincida com uma das componentes de campo da malha fonte $E_{inc}\Big|_{m_0}$. Tanto a malha 3D como a malha 1D usam o mesmo Δ e passo Δt no tempo. As equações em diferenças para a malha fonte são obtidas de (3-33)-(3-38) A única modificação é que o fator de equalização para a velocidade de fase numérica é escrita como $\nu_P(\theta = 0^o, \phi = 0^o)/\nu_P(\theta, \phi)$ refletindo a possibilidade angular adicional para a direção do vetor de onda em 3D. Dada a distancia de retardo "d" e os valores da malha 1D fonte, o método de interpolação linear deve ser usado para obter os valores de campo incidente $E_{inc}\Big|_d^n$ e $H_{inc}\Big|_d^n$. O último passo é calcular as componentes vetorais requeridas do campo incidente:

$$H_{x,inc}\Big|_d^{n+1/2} = H_{inc}\Big|_d^{n+1/2}(sen\psi\ sen\phi\ +\ cos\psi\ cos\theta\ cos\phi) \quad (3\text{-}178)$$

$$H_{y,inc}\Big|_d^{n+1/2} = H_{inc}\Big|_d^{n+1/2}(-sen\psi\ cos\phi\ +\ cos\psi\ cos\theta\ sen\phi) \quad (3\text{-}179)$$

$$H_{z,inc}\Big|_d^{n+1/2} = H_{inc}\Big|_d^{n+1/2}(-cos\psi\ sen\theta) \quad (3\text{-}180)$$

$$E_{x,inc}\Big|_d^{n} = E_{inc}\Big|_d^{n}(cos\psi\ sen\phi\ -\ sen\psi\ cos\theta\ cos\phi) \quad (3\text{-}181)$$

$$E_{y,inc}\Big|_d^{n} = E_{inc}\Big|_d^{n}(-cos\psi\ sen\phi\ -\ sen\psi\ cos\theta\ cos\phi) \quad (3\text{-}182)$$

$$E_{z,inc}\Big|_d^{n} = E_{inc}\Big|_d^{n}(sen\psi\ sen\theta) \quad (3\text{-}183)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

Saidas do programa TFSF

As figuras (3.25-3.32) ilustram vários casos de implementação da técnica TFSF, com uma onda plana propagando-se na região de campo total para vários ângulos de incidência (especificados nas legendas). Nas figuras (3.25) a (3.27), a região de campo total está livre de espalhadores e nas figuras (3.28) a (3.30) as ondas planas interagem com um cilindro PEC posicionado no centro da malha e com raio 5, $\epsilon = 50$, $\sigma = 1e7$, sendo mostrada a evolução das ondas plana de forma a destacar as interações com o espalhador. Nas figuras (3.31) a (3.32), a onda plana interage com uma lâmina quadrada PEC.

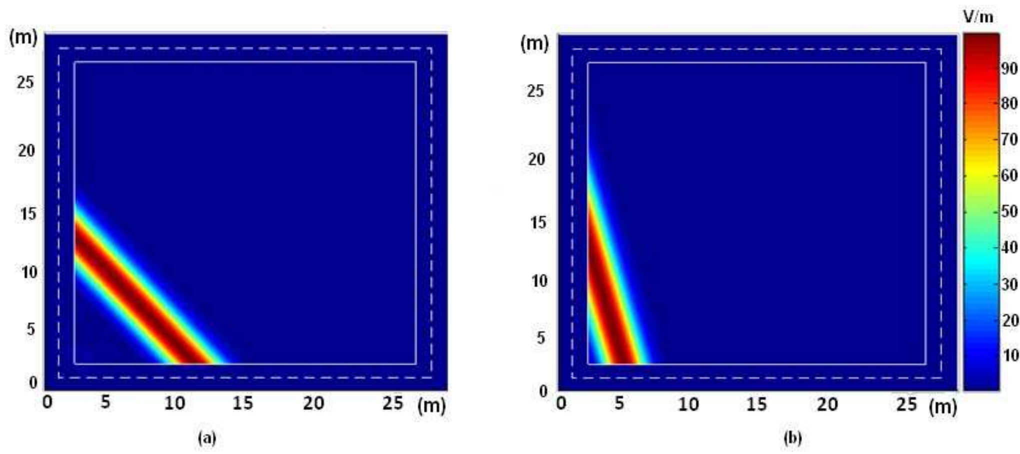

Figura 3.25: Onda plana propagando-se na região de campo total, ângulo de incidência: a) 50º e b)75º

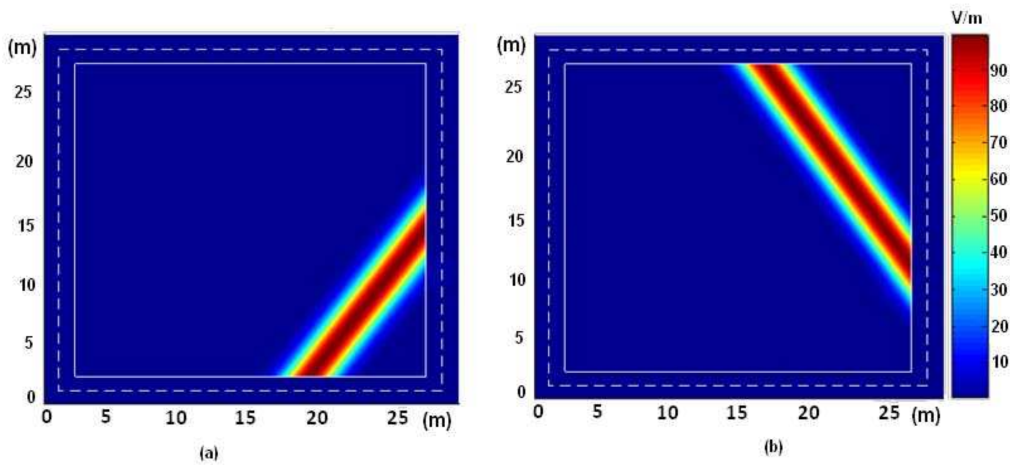

Figura 3.26: Onda plana propagando-se na região de campo total, ângulo de incidência a) 125º e b) 237º.

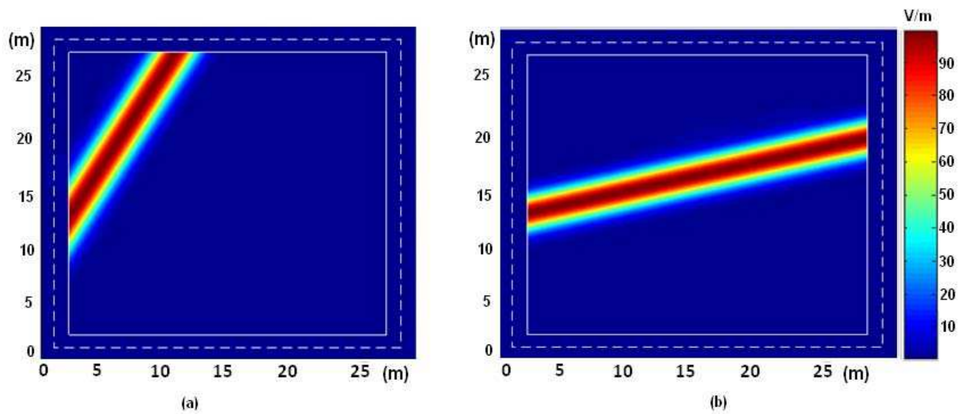

Figura 3.27: Onda plana propagando-se na região de campo total, ângulo de incidência a) 300º e b) 345º.

PUC-Rio - Certificação Digital Nº 0310507/CB

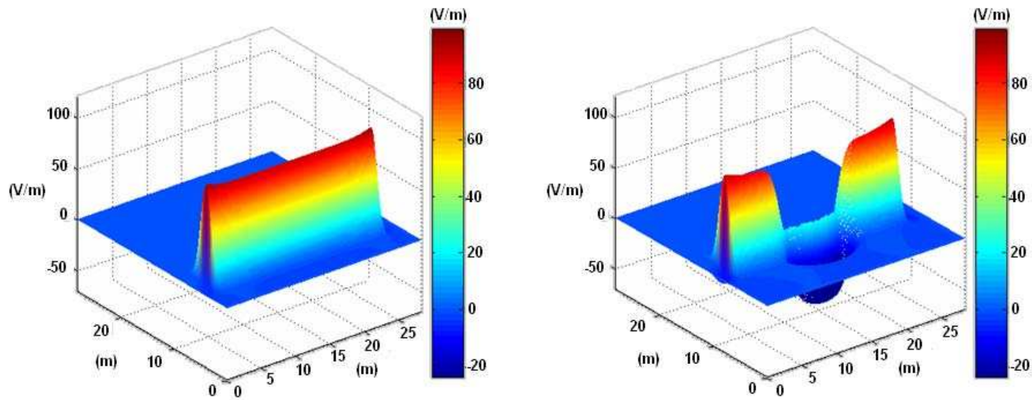

Figura 3.28: Onda plana propagando-se na região de campo total ângulo de incidência 0º . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$.

PUC-Rio - Certificação Digital Nº 0310507/CB

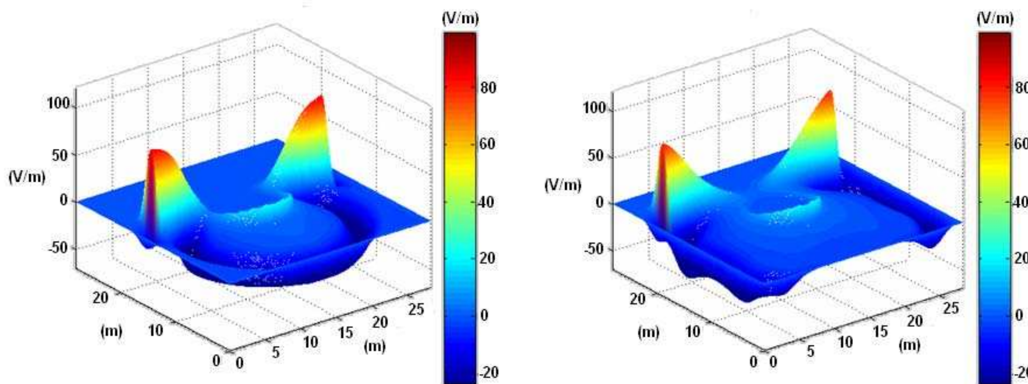

Figura 3.29: Onda plana propagando-se na região de campo total ângulo de incidência 0º . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$.

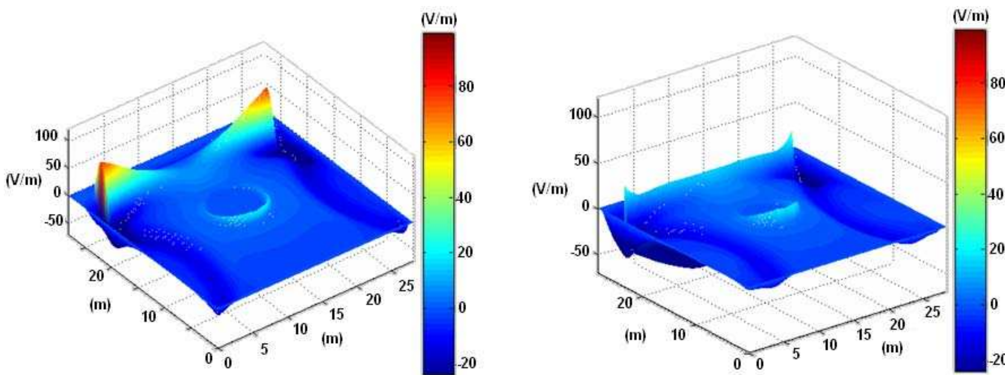

Figura 3.30: Onda plana propagando-se na região de campo total ângulo de incidência 0º . Interagindo com um cilindro de raio 5, $\epsilon = 50$,$\sigma = 1e7$.

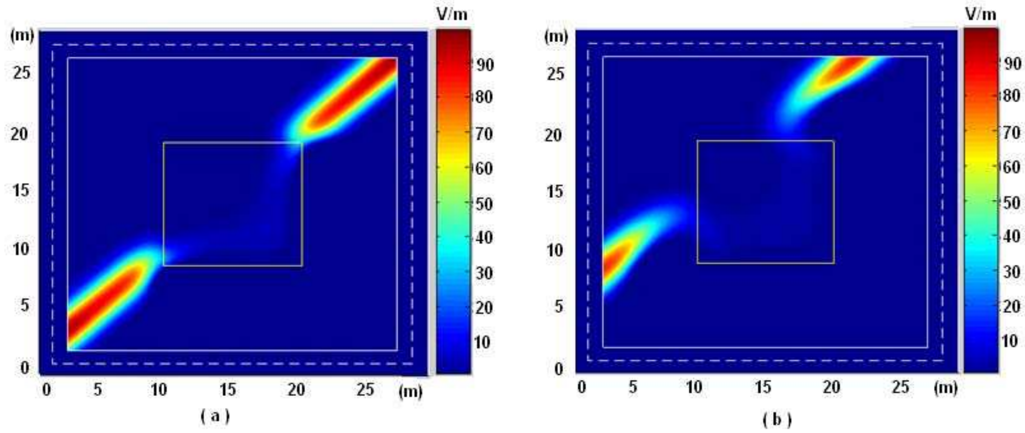

Figura 3.31: Onda plana, ângulos de incidência 130º e 0º, para diferentes passos. Interagindo com uma lámina quadrada.

PUC-Rio - Certificação Digital Nº 0310507/CB

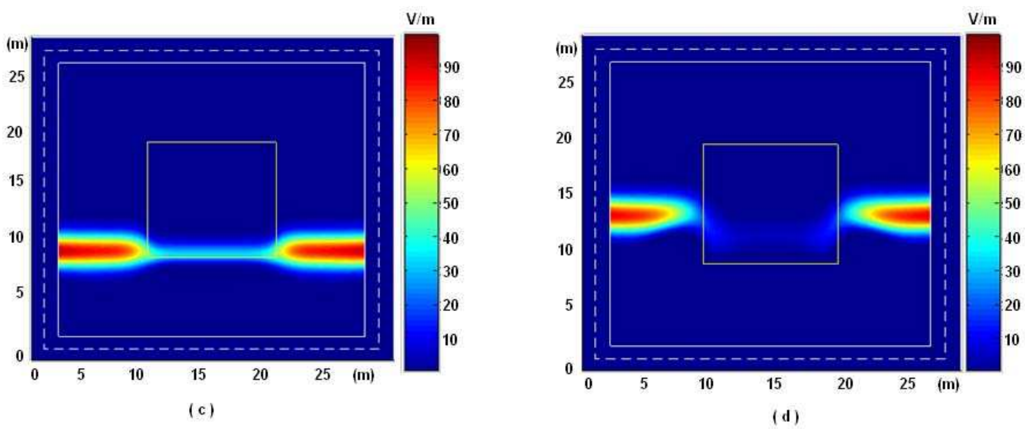

Figura 3.32: Onda plana, ângulos de incidência 130º e 0º, para diferentes passos. Interagindo com uma lámina quadrada.

4
Métodos Assintóticos

No regime de freqüências altas, quando as propriedades do meio não variam ao longo de um comprimento de onda e as dimensões dos obstaculos envolvidos são muito maiores que o mesmo, pode-se rastrear o campo eletromagnético ao longo de trajetorias (raios) ortogonais às fontes de onda e tratar os problemas de espalhamento localmente. Isto é feito a partir dos postulados o procedimentos estabelecidos pela Ótica Geométrica (GO) e outros métodos assintóticos como, por exemplo, a teoria (Geometrica) Uniforme da Difração (UTD), revistas a seguir.

4.1
A Ótica Geométrica

A Ótica Geométrica (GO) ou ótica de raios usa o conceito de raio para descrever os mecanismos de propagação da onda eletromagnética segundo o principio de Fermat (ou principio de mínima ação) que estabelece que o percurso descrito pelo raio corresponde à curva que minimiza o caminho ótico (retilíneo em meios homogêneos). Devido ao principio de conservação da energia, para o tubo de raios ilustrado na figura (4.1),

$$I_{\Omega} d\Omega = I_1 \, d\Sigma_1 = I_2 \, d\Sigma_2 \tag{4-1}$$

onde I_i denotam as densidades de energia através das seções retas de áreas Σ_i do tubo ao longo do raio, também chamadas de superficies eikonais e correspondentes às superficies equifasicas (ou "frentes de onda").

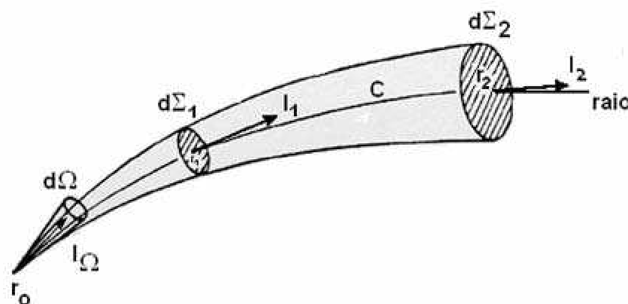

Figura 4.1: Tubo de raios em um meio não homogêneo.

PUC-Rio - Certificação Digital Nº 0310507/CB

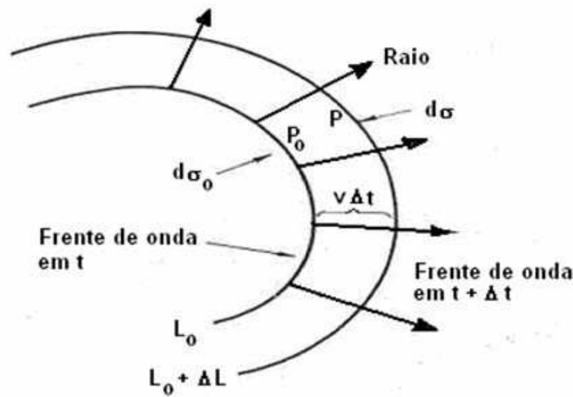

Figura 4.2: Relação entre os raios e as frentes de onda.

A variação da amplitude do campo da ótica geométrica ao longo de um tubo de raios é determinada pela lei da conservação da energia em (4-1) e considerando duas frentes de onda L_0 e $L_0 + \Delta L$, como mostrado na figura (4.2) pode-se construir um tubo de fluxo de energia constante usando raaios a eles ortogonais. O fluxo de energia através da seção transversal da seção transversal $d\sigma$ em P.
Resulta:

$$S_0 \; d\sigma_0 \;\; = \;\; S \; d\sigma \tag{4-2}$$

onde a densidade de potência média S corresponde à parte real do vetor de Poyting complexo

$$\left|\vec{S}\right| = \frac{1}{2}\sqrt{\frac{\epsilon}{\mu}} \;\; \left|\vec{E}\right|^2 \tag{4-3}$$

Substituindo 4-4 em 4-2 obtemos:

$$\left|\vec{E_0}\right|^2 \; d\sigma_0 = \left|\vec{E}\right|^2 \; d\sigma \tag{4-4}$$

e, em forma explicita, a variação da amplitude do campo em função da variação da área de seção reta de tubo de raios:

$$\left|\vec{E}\right| = \left|\vec{E_0}\right|\sqrt{\frac{d\sigma_0}{d\sigma}} \tag{4-5}$$

Considerando o tubo de raios astigmático ilustrado na figura (4.3), a expressão (4-4) pode ser reescrita em função dos raios principais de curvatura (ρ_1, ρ_2) e $(\rho_1 + l, \rho_2 + l)$ da frente de ondas, utilizando:

$$\frac{d\sigma_0}{d\sigma} = \frac{\rho_1 \; \rho_2}{(\rho_1 + \ell, \rho_2 + \ell)} \tag{4-6}$$

Substituindo (4-6) em (4-4)

PUC-Rio - Certificação Digital Nº 0310507/CB

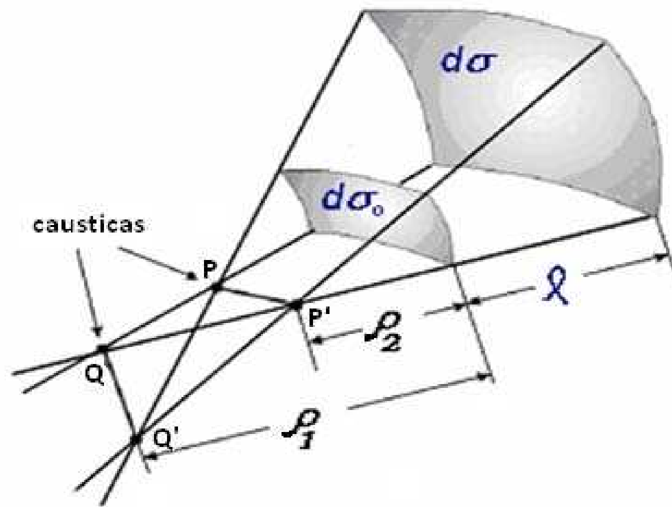

Figura 4.3: Tubo de raios astigmático.

$$\left|\vec{E}\right| = \left|\vec{E}_0\right| \sqrt{\frac{\rho_1\ \rho_2}{(\rho_1+\ell, \rho_2+\ell)}} \tag{4-7}$$

Observa-se que, quando o tubo de raios é focalizado, sua seção reta se anula e, consequentemente, a amplitude do campo em (4-7) tende a infinito, invalidando a aproximação da GO. Aos lugares geométricos desses pontos focais, dá-se o nome de (em geral de superfícies cáusticas)

A expressão completa para, o campo da GO é obtida pela inclusão do fator de fase, a partir do percurso ótico (l) desde o ponto de referência:

$$\left|\vec{E}\right| = \left|\vec{E}\right| \sqrt{\frac{\rho_1\ \rho_2}{(\rho_1+\ell, \rho_2+\ell)}}\ exp(-jk\ell) \tag{4-8}$$

onde o numero de onda $k = 2\pi/\lambda$, com λ denotando o comprimento de onda de trabalho.

4.1.1 Campo de onda direta

A figura (4.4) representa a vista superior de um cenário de 20x20x6 m. com os raios traçados em azul representando a radiação direta do transmissor em (-10,0,3) que são interceptados pelas paredes internas e os traçados em verde aqueles que atinguem, sem obstrução, as bordas do cenário. Outro exemplo é apresentando na figura (4.5)

Em campo distante, o campo elétrico (onda esférica) calculado a uma distância d de uma antena transmissora é dado por:

$$\vec{E}(d) = \vec{E}_A\ \frac{e^{-jkd}}{d} \tag{4-9}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

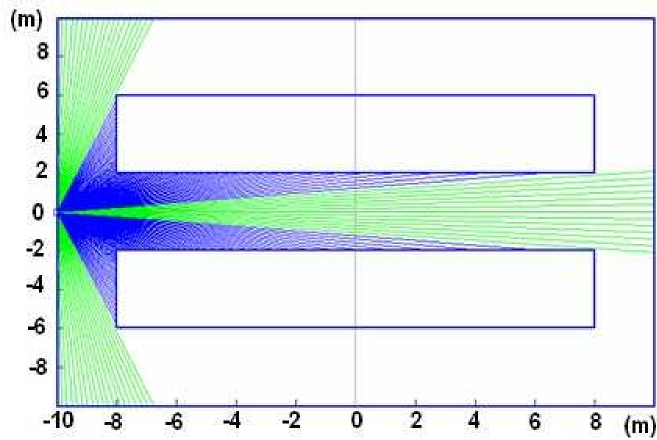

Figura 4.4: Vista 2D de cenário de 20x20x6 m, com um transmissor na posição (-10,0,3), mostrando os raios em visada direta (verde) sem atingir as paredes do corredor, os raios (azul) que originaram raios refletidos, transmitidos e difratados. .

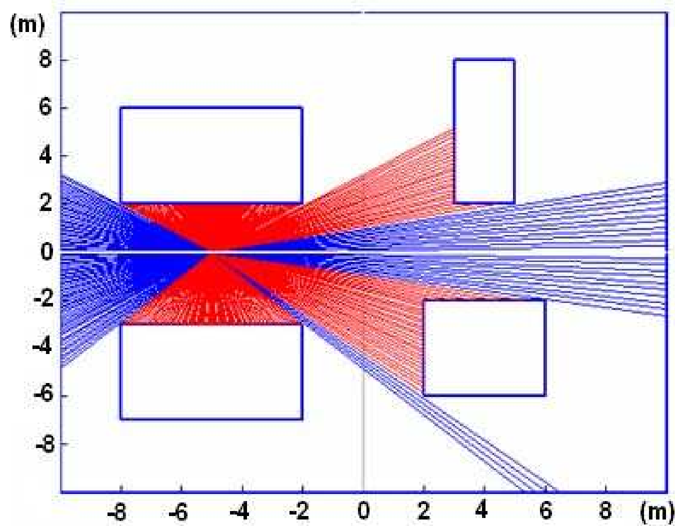

Figura 4.5: Vista 2D de cenário de 20x20x6 m, com um transmissor na posição (-5,0,2), mostrando os raios em visada direta (azul) sem atingir as paredes do corredor, os raios (vermelho) que originam raios refletidos, transmitidos e difratados.

PUC-Rio - Certificação Digital Nº 0310507/CB

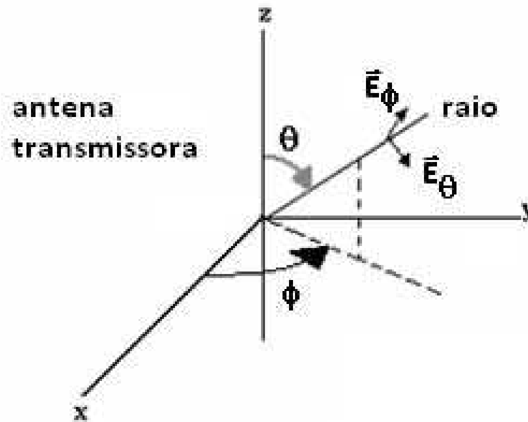

Figura 4.6: Sistema de coordenadas esféricas associado à antena transmissora.

onde:

$$\vec{E}_A(\theta,\psi) = \sqrt{\frac{\eta_0 P_t G_t}{2\pi}}\ \ \vec{E}_0(\theta,\psi) \quad [V/m] \tag{4-10}$$

com

- $\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} \cong 120\pi\Omega$ impedância intrínseca do espaço livre.
- $\epsilon_0 = 8,854 \times 10^{-12} F/m$ permissividade elétrica no espaço livre.
- $\mu_0 = 4\pi \times 10^{(} - 7)H/m$ permeabilidade magnética no espaço livre.
- Pt [W] potência de transmissâo.
- Gt ganho máximo da antena transmissora, em valor absoluto.
- $\vec{E}_0(\theta,\phi) = E_0(\theta,\phi)\ \hat{a}$, obtido a partir do diagrama de radiação da antena, com o par de coordenadas (θ,ϕ) relativo ao sistema de coordenadas esféricas centrado na antena, como ilustrado na figura (4.6).

4.1.2 Campo da onda refletida

O fenômeno da reflexão causa alteração no campo elétrico (amplitude e fase) e na direção de propagação da onda eletromagnética. Na figura (4.7) são representadas às reflexões especulares de primeira ordem dos raios que atingem os blocos do mesmo cenário apresentado na figura (4.5).

A direção da onda refletida é regida pela lei de Snell da Reflexão, segundo a qual:

$$\theta_r = \theta_i \tag{4-11}$$

onde os ângulos são conforme indicados na figura (4.8)

PUC-Rio - Certificação Digital Nº 0310507/CB

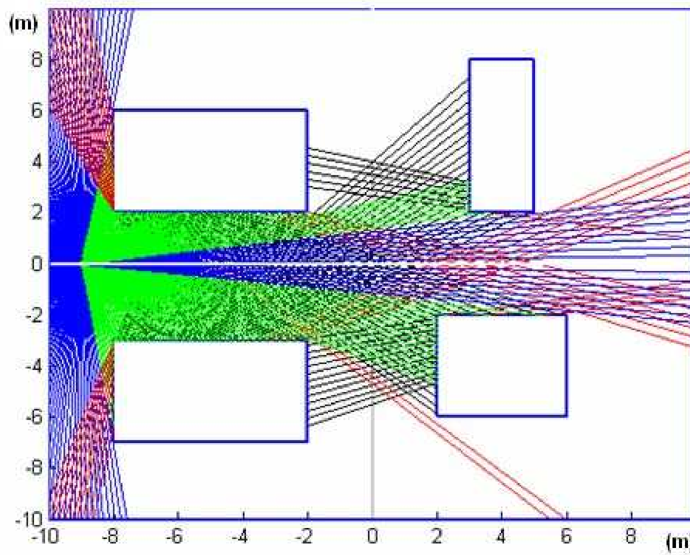

Figura 4.7: Vista 2D de cenário 20x20x6 m com antena na posição (-9, 0,2): primeira reflexão nos prédios representada pelos raios em preto, primeira reflexão nos prédios, mas saindo do cenário em vermelho; raios sem atingir os prédios, saindo do cenário, em azul.

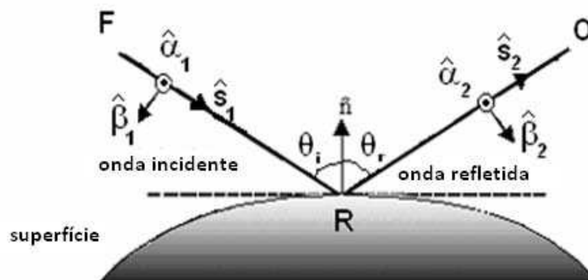

Figura 4.8: Sistemas de coordenadas fixos aos raios para a reflexão.

PUC-Rio - Certificação Digital Nº 0310507/CB

$\hat{n}$	vetor unitário normal à superficie refletora no ponto de reflexão R
θ_i	ângulo de incidência: ângulo agudo formado entre o raio incidente e o vetor normal à superficie $(0 \leq \theta_i \leq \pi/2)$ $\theta_i = acos(-\hat{n} \cdot \hat{s}_1)$
θ_r	ângulo de reflexão: ângulo agudo formado entre o raio refletido e o vetor normal $(0 \leq \theta_r \leq \pi/2)$
Plano de incidencia	plano que contem o raio incidente (direção de propagação da onda incidente, $\hat{s}_1$ e a normal $\hat{n}$
Plano de reflexão	plano que contem o raio refletido (direção de propagação da onda refletida, $\hat{s}_2$ e a normal $\hat{n}$
Os vetores $\hat{\alpha}_1, \hat{\beta}_1, \hat{s}_1$ e $\hat{\alpha}_2, \hat{\beta}_2, \hat{s}_2$	definem os sistemas de coordenadas fixos ao raio incidente e ao raio refletido, respectivamente, conforme explicado adiante.

O campo refletido relaciona-se ao incidente, no ponto de reflexão R da superficie, através de

$$\vec{E}^r(x_R, y_R, z_R) = \vec{E}^i(x_R, y_R, z_R)\bar{R} \tag{4-12}$$

onde:

$\vec{E}^r(x_R, y_R, z_R)$ campo refletido, calculado em R.

$\vec{E}^i(x_R, y_R, z_R)$ campo incidente, calculado em R.

$\bar{R}$ matriz de coeficentes de reflexão da superficie.

Sistemas de coordenadas fixos aos raios para a reflexão

Os sistemas fixos aos raios (incidente e refletido) são introduzidos para que $\bar{R}$ seja uma matriz 2×2 o que é feito através de um dois eixos do sistema ao longo do própio raio (incidente)$\hat{s}_1$ ou refletido $\hat{s}_2$, e os eixos restantes, perpendiculares ao raio, ao longo das direções transversais de polarização de campo, paralela ($\hat{\beta}_{1,2}$) e perpendicular ($\hat{\alpha}_{1,2}$) aos planos de incidencia e de reflexão conforme ilustrado na figura (4.8).

As componentes de campo perpendiculares ao plano de incidencia / reflexão são também denominadas componentes "soft". As componentes sobre o plano (componentes paralelas), são também conhecidas como componentes "hard". Assim:

$$\vec{E}^{i,r}_{\hat{\alpha}_{1,2}} = \vec{E}^{i,r} \cdot \hat{\alpha}_{1,2} \quad \text{componentes soft.}$$
$$\vec{E}^{i,r}_{\hat{\beta}_{1,2}} = \vec{E}^{i,r} \cdot \hat{\beta}_{1,2} \quad \text{componentes hard.}$$

Os vetores unitarios envolvidos são dados por :

$$\hat{s}_1 = \frac{\vec{s}_1}{|\vec{s}_1|} \qquad \hat{s}_2 = \frac{\vec{s}_2}{|\vec{s}_2|} \tag{4-13}$$

onde:

$$\hat{s}_2 = \hat{s}_1 - 2(\hat{n} \cdot \hat{s}_1)\hat{n} \tag{4-14}$$

e, a partir da figura (4.8)

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\hat{\alpha}_1 = \frac{\hat{s}_1 \times \hat{n}}{|\hat{s}_1 \times \hat{n}|} \qquad \hat{\beta}_1 = \hat{s}_1 \times \hat{\alpha}_1$$
$$\hat{\alpha}_2 = \frac{\hat{s}_2 \times \hat{n}}{|\hat{s}_2 \times \hat{n}|} \qquad \hat{\beta}_2 = \hat{s}_2 \times \hat{\alpha}_2 \tag{4-15}$$

Determinação do campo refletido

Através do uso dos sistemas fixos aos raios incidente e refletido, os campos podem ser descompostos nas direções dos vetores unitários em (4-15) e a expressão de campo refletido, calculado no ponto de observação 0, descomposto nas suas componentes soft e hard, se escreve:

$$\vec{E}^r(0) = \left[E^r_{\alpha_2}(\hat{\alpha}_2) + E^r_{\beta_2}(\hat{\beta}_2)\right] A^r\ e^{-jkd_2} \tag{4-16}$$

onde

$$\begin{pmatrix} E^r_{\alpha_2} \\ E^r_{\beta_2} \end{pmatrix} = \begin{pmatrix} \Gamma_s & 0 \\ 0 & \Gamma_h \end{pmatrix} \begin{pmatrix} E^i_{\alpha_1} \\ E^i_{\beta_1} \end{pmatrix} \tag{4-17}$$

determina as componentes soft ($E^r_{\alpha_2}$) e hard ($E^r_{\beta_2}$) do campo refletido, com:

$\begin{pmatrix} \Gamma_s & 0 \\ 0 & \Gamma_h \end{pmatrix}$ matriz $\bar{R}$ de coeficentes de reflexão, composta pelos coeficentes de reflexão de Fresnel Soft e Hard(Γ_s, Γ_h).

$\begin{pmatrix} E^{i,r}_{\alpha_{1,2}} \\ E^{i,r}_{\beta_{1,2}} \end{pmatrix} = \begin{pmatrix} \vec{E}^{i,r} \cdot \hat{\alpha}_{1,2} \\ \vec{E}^{i,r} \cdot \hat{\beta}_{1,2} \end{pmatrix}$ vetor campo incidente / refletido no ponto de reflexão, $\vec{E}^{i,r}(R)$, expresso em suas componentes soft e hard, os vetores unitarios $\hat{\alpha}_{1,2}$ e $\hat{\beta}_{1,2}$ definidos em (4-15).

$\hat{\alpha}_{1,2}, \hat{\beta}_{1,2}$ conforme definidos em (4-15).

A^r fator de divergencia do tubo de raios refletidos.

d_2 distancia entre o ponto de reflexão R e o ponto de observação O.

Fator de divergência do tubo de raios

O fator de divergencia do tubo de raios refletidos, em (4-16), é dado por:

$$A^r = \sqrt{\frac{\rho^r_1\ \rho^r_2}{(\rho^r_1 + d_2)(\rho^r_2 + d_2)}} \tag{4-18}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

PUC-Rio - Certificação Digital Nº 0310507/CB

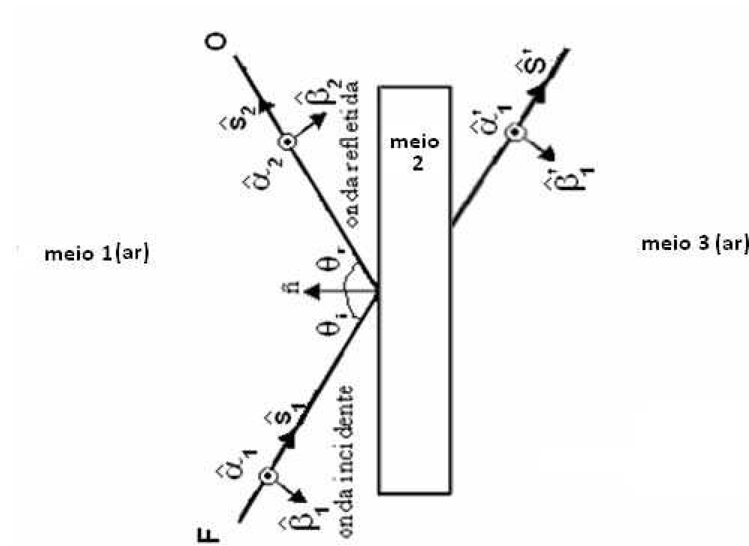

Figura 4.9: Esquema de refração: meios 1 e 3 iguais.

onde $\rho^r_{1,2}$ denotam os raios principais de curvatura da frente de onda refletida no ponto de reflexão R.

Coeficentes de reflexão de Fresnel para ambientes interiores

Em ambientes interiores, espera-se que a constituição de paredes internas e divisórias não seja muito distinta entre os ambientes e que a espessura seja mais bem definida. Os coeficentes de transmissão indoor foram obtidos através das seguintes considerações:

– são desprezadas múltiplas reflexões no interior da estrutura (reflexões entre as duas interfaces com o meio externo);

– a estrutura é considerada eletricamente distante de qualquer outra, de forma a no haver interação entre elas.

Os coeficentes apresentados a seguir consideram os meios 1 e 3 figura (4.9) como sendo os mesmos (vácuo), o meio 2 é homogêneo e isotrópico e as duas interfaces são planas nos pontos de refração.

Coeficentes de reflexão de Fresnel Soft

$$\Gamma_s(\theta_i) = \frac{Z_{2s} - (\eta_0/cos\ \theta_i)}{Z_{2s} + (\eta_0/cos\ \theta_i)} \tag{4-19}$$

com

$$Z_{2s} = (\eta_2/cos\ \theta_{r_1})\ \frac{(\eta_0/cos\ \theta_i) + (\eta_2/cos\ \theta_{r_1})\ tgh(\gamma_2\ d\ cos\ \theta_{r_1})}{(\eta_2/cos\ \theta_{r_1}) + (\eta_0/cos\ \theta_i)\ tgh(\gamma_2\ d\ cos\ \theta_{r_1})} \tag{4-20}$$

representando a impedancia [Ω] de entrada da estrutura vista da interface 1-2 figura (4.9) associada à polarização soft dos campos, onde

– a impedancia intrínseca do meio 2 [Ω]

$$\eta_2 = \frac{\eta_0}{\sqrt{\epsilon_{efr_2}}} \tag{4-21}$$

– a impedância intrinseca dos meios 1 e 3 [Ω]

$$\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} \tag{4-22}$$

– permissividade elétrica efetiva relativa do meio 2

$$\epsilon_{efr_2} = \frac{\epsilon_2 - j\frac{\sigma_2}{\omega}}{\epsilon_0} \tag{4-23}$$

. ϵ_2 permissividade elétrica do meio 2 [F/m].
. σ_2 condutividade elétrica do meio 2 [Siemens/m].
. $\omega = 2\pi f$ frequencia angular [rad/s], com f frequencia [Hz].

– Ainda a partir de (4-29) e (4-37),

$$cos\ \theta_{r1} = \sqrt{1 - \left(\frac{\gamma_0}{\gamma_2}\right)^2 sen^2\ \theta_i} \tag{4-24}$$

com

$$\gamma_2 = j\omega\sqrt{\mu_0\ \epsilon_0\ \epsilon_{erf2}} \tag{4-25}$$

$$\gamma_0 = j\omega\sqrt{\mu_0\ \epsilon_0} \tag{4-26}$$

– d [m] espessura da estrutura.

PUC-Rio - Certificação Digital Nº 0310507/CB

Coeficente de reflexão de Fresnel Hard

$$\Gamma_h(\theta_i) = \frac{Z_{2h} - (\eta_0/cos\ \theta_i)}{Z_{2h} + (\eta_0/cos\ \theta_i)} \tag{4-27}$$

com

$$Z_{2h} = (\eta_2/cos\ \theta_{r_1})\ \frac{(\eta_0/cos\ \theta_i) + (\eta_2/cos\ \theta_{r_1})\ tgh(\gamma_2\ d\ cos\ \theta_{r_1})}{(\eta_2/cos\ \theta_{r_1}) + (\eta_0/cos\ \theta_i)\ tgh(\gamma_2\ d\ cos\ \theta_{r_1})} \tag{4-28}$$

representando a impedancia [Ω] de entrada da estrutura vista da interface 1-2 figura associada à polarização hard dos campos, com os demais parametros conforme Z_{2s} em (4-20).

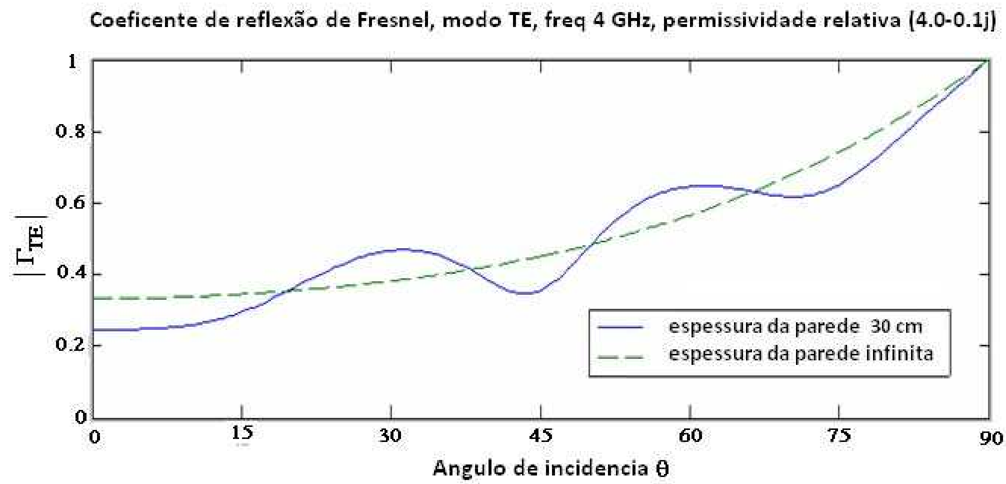

Figura 4.10: Coeficente de reflexão de Fresnel soft para paredes com permissividade relativa (4.0-0.1j) e espessuras 30 cm e infinito, em função do angulo de incidencia e para a frequencia de 4 GHz.

PUC-Rio - Certificação Digital Nº 0310507/CB

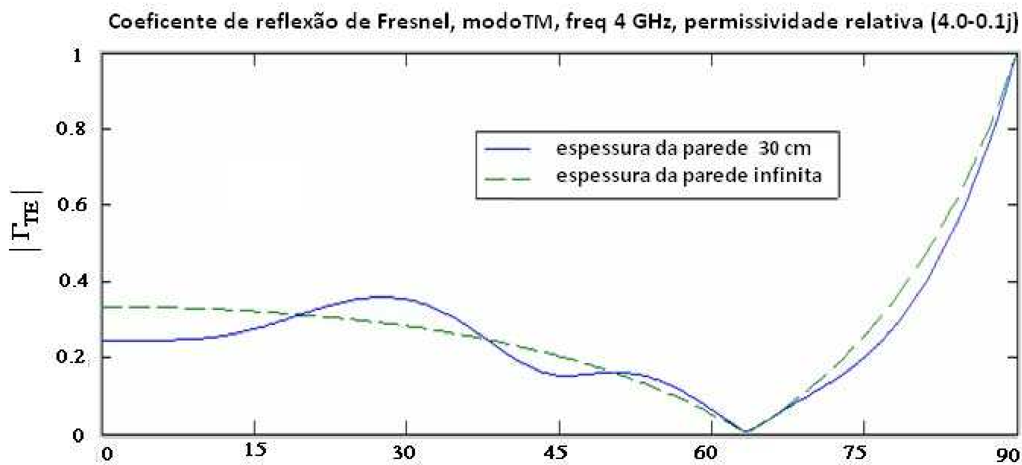

Figura 4.11: Coeficente de reflexão de Fresnel hard para paredes com permissividade relativa (4.0-0.1j) e espessuras 30 cm e infinito, em função do angulo de incidencia e para a frequencia de 4 GHz.

4.2 Refração

A onda eletromagnética, ao incidir sobre a superfície de separação entre dois meios, além de gerar a onda refletida, gera também uma onda refratada (transmitida), conforme ilustrado na figura (4.10). Também causando alterações na amplitude e fase do campo, propagante. A direção da onda refratada é regida pela Lei de Snell da Refração, dada por :

$$\gamma_1 \; sen \; \theta_i = \gamma_2 \; sen \; \theta_r \tag{4-29}$$

onde:

γ_1 constante de propagação da onda no meio 1

γ_2 constante de propagação da onda no meio 2

com:

$$\gamma_{1,2} = \alpha_{1,2} + j \; \beta_{1,2} \tag{4-30}$$

$\alpha_{1,2}$ = constantes de atenuação nos meios 1 e 2 [Np/m]

$$\alpha_{1,2} = \omega \; \sqrt{\mu_0 \epsilon_{1,2}} \; \left\{ \frac{1}{2} \left[\sqrt{1 + \left(\frac{\sigma_{1,2}}{\omega \; \epsilon_{1,2}} \right)^2} - 1 \right] \right\}^{1/2} \tag{4-31}$$

– para bons dielétricos: $\left(\frac{\sigma}{\omega \epsilon} \right)^2 << 1$

$$\alpha_{1,2} \cong \frac{\sigma_{1,2}}{2} \sqrt{\frac{\mu_0}{\epsilon_{1,2}}} \tag{4-32}$$

– para bons condutores: $\left(\frac{\sigma}{\omega \epsilon} \right)^2 >> 1$

$$\alpha_{1,2} \cong \sqrt{\frac{\omega \mu_0 \sigma_{1,2}}{2}} \tag{4-33}$$

$\beta_{1,2}$ = constate de fase nos meios 1 e 2 [rad/m]

$$\beta_{1,2} = \omega \; \sqrt{\mu_0 \epsilon_{1,2}} \; \left\{ \frac{1}{2} \left[\sqrt{1 + \left(\frac{\sigma_{1,2}}{\omega \; \epsilon_{1,2}} \right)^2} + 1 \right] \right\}^{1/2} \tag{4-34}$$

– para bons dielétricos: $\left(\frac{\sigma}{\omega \epsilon} \right)^2 << 1$

$$\beta_{1,2} \cong \omega \sqrt{\mu_0 \epsilon_{1,2}} \tag{4-35}$$

– para bons condutores: $\left(\frac{\sigma}{\omega \epsilon} \right)^2 >> 1$

$$\beta_{1,2} \cong \sqrt{\frac{\omega \mu_0 \sigma_{1,2}}{2}} \tag{4-36}$$

As constantes de propagação γ também podem ser expressas da forma:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\gamma_{1,2} = j\omega\sqrt{\mu_0\ \epsilon_{efr_{1,2}}} \tag{4-37}$$

onde, a partir das permissividades (reais) eletricas e condutividades dos meios,

$$\epsilon_{efr_{1,2}} = \epsilon_{1,2} - j\frac{\sigma_{1,2}}{\omega} \tag{4-38}$$

Na figura (4.12) de acordo com a lei de Snell em (4-29):

$$sen\ \theta_r = \frac{\gamma_1}{\gamma_2}\ sen\ \theta_i \tag{4-39}$$

As situações de interesse envolverão ambientes nos quais γ_1 será real, pois o meio 1 é o ar ($\alpha = 0$) e, em geral, γ_2 será complexo. Fica claro então, pela expressão (4-39), que o angulo θ_r calculado através da lei de Snell será um ângulo complexo e, por tanto, não tem significado físico, sendo necessário outro procedimento de cálculo para que se determine o ângulo real de refração (trajetoria real do raio refratado). O procedimento detalhado de determinação do ângulo de refração real é apresentado em [2] e, a seguir, são listados os resultados finais.

Denominando θ_r, na figura (4.12) de θ_r, ângulo real de refração, tem-se:

$$cos\ \theta_z = \frac{q}{\sqrt{(\beta_1\ sen\ \theta_i)^2 + q^2}} \qquad e \qquad sen\ \theta_z = \frac{\beta_1\ sen\ \theta_i}{\sqrt{(\beta_1\ sen\ \theta_i)^2 + q^2}} \tag{4-40}$$

onde:

$$q = s\ \left\{\frac{1}{2}\left[\alpha_2^2\left(1 - \frac{1-(a^2-b^2)\ sen^2\ \theta_i}{s^2}\right) + \beta_2^2\left(1 + \frac{1-(a^2-b^2)\ sen^2\ \theta_i}{s^2}\right)\right] + \alpha_2\ \beta_2\ sen(2\nu)\right\}^{1/2} \tag{4-41}$$

com:

$$s = \left[\frac{2\ ab\ sen^2\ \theta_i}{\sqrt{\frac{A^2}{A^2+1}}}\right]^{1/2} \tag{4-42}$$

$$a = \frac{\beta_1\ \beta_2}{\alpha_2^2 + \beta_2^2} \qquad b = \frac{\beta_1\ \alpha_2}{\alpha_2^2 + \beta_2^2} \tag{4-43}$$

$$A^2 = \left[\frac{-2\ ab\ sen^2\ \theta_i}{1-(a^2-b^2)\ sen^2\ \theta_i}\right]^2 \tag{4-44}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

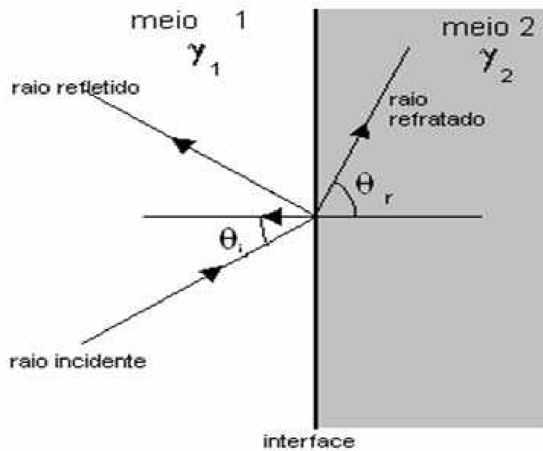

Figura 4.12: Esquema de reflexão e refração dos raios na interface entre dois meios diferentes.

$$sen\ (2\nu) = -\sqrt{\frac{A^2}{A^2+1}} \tag{4-45}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

4.2.1 Determinação do campo transmitido

Considerando o esquema da figura (4.9) (meios 1 e 3 iguais), o campo transmitido pela estrutura relaciona-se ao incidente através da expressão:

$$\vec{E}^t\ (0) = \left[E^t_{\alpha_1}\ \hat{\alpha}_1 + E^t_{\beta_1}\ \hat{\beta}_1\right] A^t\ e^{-jkd_2} \tag{4-46}$$

onde:

$\vec{E}^t(0)$: campo transmitido calculado no ponto de observação O, descomposto em suas componentes soft e hard.

$\begin{bmatrix} \vec{E}^t_{\alpha_i} \\ \vec{E}^t_{\beta_i} \end{bmatrix} = \begin{bmatrix} T_s & 0 \\ 0 & T_h \end{bmatrix} \begin{bmatrix} \vec{E}^i_{\alpha_i} \\ \vec{E}^i_{\beta_i} \end{bmatrix}$: componentes soft (E_{α_t}) e (E_{β_t}) do campo transmitido com:

$\begin{bmatrix} T_s & 0 \\ 0 & T_h \end{bmatrix}$: matriz $\bar{T}$ de coeficentes de transmissão, composta pelos coeficentes de transmissão de Fresnel soft e hard, asociados aos sistemas de coordenadas fixos aos raios.

$\begin{bmatrix} \vec{E}^i_{\alpha_i} \\ \vec{E}^i_{\beta_i} \end{bmatrix} = \begin{bmatrix} \vec{E}^i\ (I)\cdot\hat{\alpha}_1 \\ \vec{E}^i\ (I)\cdot\hat{\beta}_1 \end{bmatrix}$: Componente soft e hard do campo incidente na estrutura, $\vec{E}^i(I)$, com os vetores unitarios $\hat{\alpha}_1$ e $\hat{\beta}_1$ definidos em (4-15)

e observando-se que os vetores do sistema fixo ao raio incidente já que os meios 1 e 3 são iguais.

A^t fator de divergencia do tubo de raios transmitido, dado para superficies planas por (4-18), substituendo $\rho^i_{1,2}$ pelos raios principais de curvatura da frente de onda incidente, $\rho^i_{1,2}$. Nesse caso, a distancia d_2 na equação (4-18) é aproximadamente a distancia entre o ponto de incidencia I e o observador O. A transmissão por superficies planas não altera a forma da onda incidente no obstaculo.

Coeficentes de transmissão de Fresnel

Assim como no casao da reflexão, os coeficentes de transmissão apresentados a seguir consideram os meios 1 e 3 figura (4.9) iguais (vácuo), o meio 2 homogêneo e isotrópico e as duas interfcaes planas nos pontos de refração.

Coeficente de transmissão de Fresnel soft

$$T_s\ (\ \theta_i\ ) = T_{1s} \cdot T_{2s} \tag{4-47}$$

$$T_{1s} = \frac{1 + \Gamma_{s1}}{e^{\gamma_2\ d\ cos\ \theta_{r1}} + \Gamma_{s2} e^{-\gamma_2\ d\ cos\ \theta_{r1}}} \tag{4-48}$$

$\Gamma_{s1} = \Gamma_s(\theta_i)$ para ambientes interiores, dado por (4-19)

$$\Gamma_{s2} = \frac{(\eta_0/cos\ \theta_i) - (\eta_2/cos\ \theta_{r1})}{(\eta_0/cos\ \theta_i) + (\eta_2/cos\ \theta_{r1})} \tag{4-49}$$

$$T_{2s} = \frac{2(\eta_0/cos\ \theta_i)}{(\eta_2/cos\ \theta_{r1}) + (\eta_0/cos\ \theta_i)} \tag{4-50}$$

com os parametros definidos como na determinação do campo refletido

PUC-Rio - Certificação Digital Nº 0310507/CB

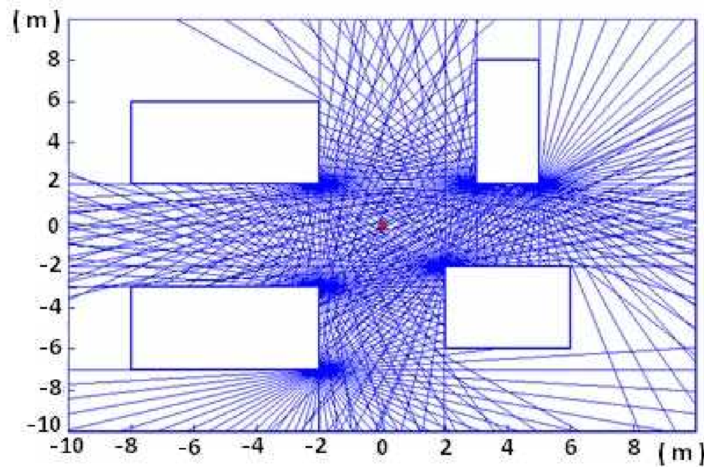

Figura 4.13: Raios difratados nas quinas dos predios com Tx em (0,0,2).

Coeficente de transmissão de Fresnel hard

$$T_h\ (\ \theta_i\ ) = T_{1h} \cdot T_{2h} \tag{4-51}$$

$$T_{1h} \;=\; \frac{1+\Gamma_{h1}}{e^{\gamma_2\ d\ cos\ \theta_{r1}} + \Gamma_{h2} e^{-\gamma_2\ d\ cos\ \theta_{r1}}} \tag{4-52}$$

$\Gamma_{h1} = \Gamma_h(\theta_i)$ para ambientes interiores, dado por (4-27)

$$\Gamma_{h2} = \frac{(\eta_0/cos\ \theta_i) - (\eta_2/cos\ \theta_{r1})}{(\eta_0/cos\ \theta_i) + (\eta_2/cos\ \theta_{r1})} \tag{4-53}$$

$$T_{2h} \;=\; \frac{2(\eta_0/cos\ \theta_i)}{(\eta_2/cos\ \theta_{r1}) + (\eta_0/cos\ \theta_i)} \tag{4-54}$$

com os parametros definidos como na determinação

do campo refletido

4.3
Campo Difratado

A difração é, em geral, o mecanismo de espalhamento eletromagnético na borda de uma superficie, na aresta formada pela junção (aresta de uma quina) de duas superficies, no vértice de um sólido ou, ainda, devido à incideência rasante sobre uma superficie [2], A figura (4.13) ilustra, para o cenário da figura (4.5), o traçado de raios que sofrem difração de primeira ordem.

PUC-Rio - Certificação Digital Nº 0310507/CB

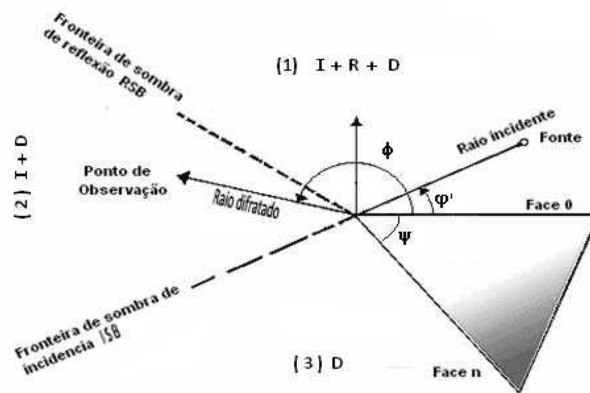

Figura 4.14: Raio incidente sôbre uma aresta e divisão da área ao redor em três regiões distintas.

Se um raio incide sobre uma aresta a área ao, seu redor, no plano perpendicular à mesma, em três regiões distintas, conforme ilustrado na figura (4.14)

– Na região (1) estão presentes o raio incidente, os raios refletidos na face "0" e o campo difratado na aresta. A fronteira desta região é definida pelo raio refletido na borda da aresta e é conhecida como fronteira de sombra de reflexão RSB (Reflection Shadow Boundary).

– A região (2) iluminada por raios diretos e difratados, é limitada pela fronteira de sombra de incidencia - ISB (Incidence Shadow Boundary) correspondente ao prolongamento do raio incidente sobre a aresta, e pela fronteira RSB.

– Na região (3), iluminada pela fronteira ISB e pela face "n" não iluminada do obstaculo, apenas os raios difratados estão presentes e é conhecida como região de sombra.

A tabela 4.1 resume as caracteristicas das regiões acima descritas. A teoria Uniforme da Difração (UTD), corresponde à extensão da Teoria Geometrica da Difração (GTD) para tornar o campo total, difratado e da Ótica Geométrica (incidente ou refletido), finito e continuo através das fronteiras de sombra, é resumida a seguir:

	Região 1	Região 2	Região 3
Espaço azimutal	$0 < \phi < \pi - \phi'$	$\pi - \phi' < \phi < \pi + \phi'$	$\pi + \phi' < \phi < 2\pi - \psi$
Constitução do campo	incidente, refletido,	incidente, difratado	difratado difratado

Tabela 4.1: Regiões definidas pelas fronteiras de sombra

PUC-Rio - Certificação Digital Nº 0310507/CB

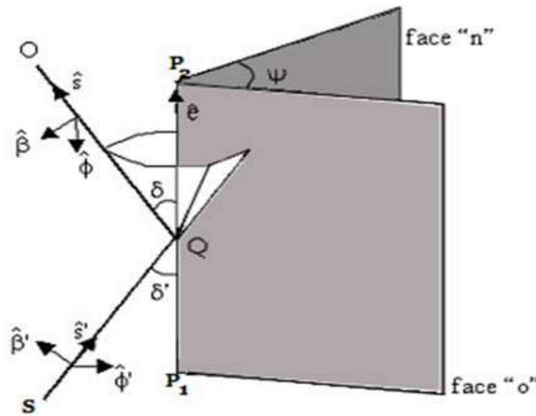

Figura 4.15: Cone de difração e sistemas de coordenadas fixos aos raios para a difração.

Teoria Uniforme da Difração (UTD)

A figura (4.15) apresenta o detalhamento dos parâmetros e sistemas de coordenadas envolvidos na aplicação da UTD os cálculos de campo difratado por arestas retas em cunhas de cones correspondendo à situação de interesse, onde os obstaculos serão representados por facetas planas tangentes às superficies.

A incideência de um raio em uma aresta gera, alem do(s) raio(s) refletido(s) nas faces “0” e “n” um cone de raios difratados (cone de Keller) com semi-ângulos de abertura (δ) igual ao ângulo (δ') segundo o qual o raio incidente atinge a aresta e vertice concidente com o ponto de difração Q, resultando:

$$\hat{s}' \cdot \hat{e} \;=\; \hat{s} \cdot \hat{e} \tag{4-55}$$

Na figura 4.15 é possível identificar :

Plano de incidencia fixo à aresta : plano que contem o raio incidente (direção de propagação da onda incidente, $\hat{s}'$ é o vetor $\hat{e}$.

Plano de difração fixo à aresta : plano que contem um raio difratado (direção de propagação da onda difratada, $\hat{s}$ e o vetor $\hat{e}$.

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\hat{e} \ : \ \text{vetor unitário tangente à aresta no ponto de difração}$$
$$\delta' = acos(\hat{s}' \cdot \hat{e}) \ : \ \text{ângulo agudo entre o raio incidente } \hat{s}' \text{ e o vetor } \hat{e} \text{ tangente à aresta.}$$
$$\delta = \delta' \ : \ \text{angulo agudo entre o raio difratado } (\hat{s}) \text{ e o vetor } \hat{e} \text{ tangente à aresta.}$$
$$\Psi \ : \ \text{ângulo de abertura da cunha.}$$

Os vetores $\hat{s}', \hat{\beta}', \hat{\phi}'$ e $\hat{s}, \hat{\beta}, \hat{\phi}$ constituem os sistemas de coordenadas fixos aos raios incidente e difratados, respectivamente, conforme especificados a seguir.

Sistemas de coordenadas fixos aos raios para a difração

Os sistemas fixos aos raios para a difração são sistemas de três eixos no qual:

- um eixo está ao longo do raio e, na figura (4.15), corresponde aos unitarios $\hat{s}'$ e $\hat{s}$ ao longo dos raios incidente e difratado, respetivamente;
- um eixo é perpendicular aos planos de incidência / difração fixos à aresta e, na figura (4.15) corresponde aos unitarios $\hat{\phi}'$ e $\hat{\phi}$, respectivamente;
- um terceiro eixo esta sobre os planos de incidência / difração fixos à aresta e na figura (4.15), corresponde aos unitários $\hat{\beta}'$ e $\hat{\beta}$, respectivamente.

As componentes perpendiculares aos planos de incidência / difração fixos à aresta são denominadas componentes hard, enquanto aquelas sobre os planos (componentes paralelas), são conhecidas por componentes soft. Assim, sejam $\vec{E}^i$ o campo elétrico incidente e $\vec{E}^d$ o campo elétrico difratado:

$$\vec{E}^i \cdot \hat{\beta}' \quad e \quad \vec{E}^d \cdot \hat{\beta} \ : \ \text{componentes soft}$$
$$\vec{E}^i \cdot \hat{\phi}' \quad e \quad \vec{E}^d \cdot \hat{\phi} \ : \ \text{componentes hard}$$

Os vetores unitários envolvidos $\hat{s}, \hat{s}', \hat{e}$ são deduzidos apartir da figura (4.15)

$$\hat{s}' = \frac{\vec{Q} - \vec{S}}{\left|\vec{Q} - \vec{S}\right|} \qquad \hat{e} = \frac{\vec{P}_2 - \vec{P}_1}{\left|\vec{P}_2 - \vec{P}_1\right|} \tag{4-56}$$

onde :

$$\hat{s} = (cos\ \delta')\hat{e} \ + \ sen\ \delta'[(cos\ \phi)\ \hat{f} + (sen\ \phi)\ \hat{n}_0] \tag{4-57}$$

PUC-Rio - Certificação Digital Nº 0310507/CB

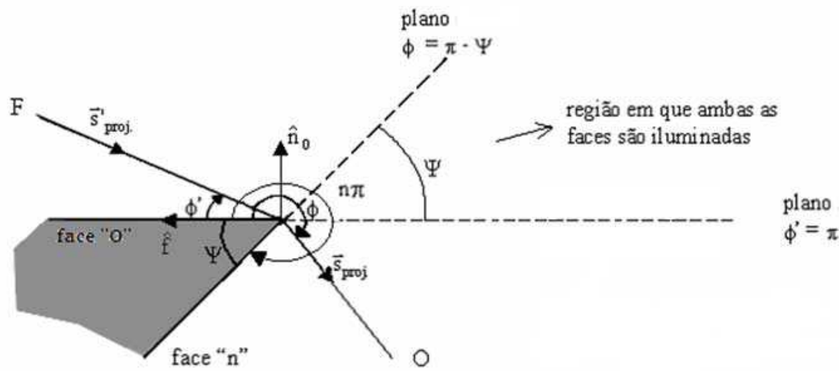

Figura 4.16: Vista do plano normal à aresta; indice "proj" refere-se à projeção no plano da vista.

com

$$\begin{aligned} cos\ \delta' &= \hat{s} \cdot \hat{e} \\ sen\ \delta' &= \hat{s}' \times \hat{e} \end{aligned}$$

$\hat{f}$: vetor unitário normal a $\hat{e}$, sobre o plano tangente à face "O" no ponto de difração Q figura (4.16).

$\hat{n}_0$: vetor unitário normal à face "0" no ponto de difração Q figura (4.16).

ϕ : ângulo entre a projeção do raio difratado sobre o plano normal à aresta no ponto de difração e o plano tangente à face "0" no ponto de difração figura (4.16).

e, a partir da figura (4.15)

$$\hat{\phi}' = -\frac{\hat{e} \times \hat{s}'}{|\hat{e} \times \hat{s}'|} \qquad \hat{\beta}' = \hat{\phi}' \times \hat{s}' \tag{4-58}$$

$$\hat{\phi} = \frac{\hat{e} \times \hat{s}}{|\hat{e} \times \hat{s}|} \qquad \hat{\beta} = \hat{\phi} \times \hat{s} \tag{4-59}$$

Determinação do campo difratado

Será aqui descrita a difração de primeira ordem que é a mais relevante e corresponde, na maioria dos casos, à principal contribuição ao campo total difratado. A contribuição (segundo a UTD) de segunda ordem, ou "slope diffraction", não será aqui considerada.

Difração de primeira ordem

O campo difratado relaciona-se ao campo incidente através da expressão:

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\vec{E}^d(0) = \left[E^d_\beta\ \hat{\beta} + E^d_\phi\ \hat{\phi}\right]\ A^d\ exp(-j\ k_0\ d_2) \tag{4-60}$$

onde:

$\vec{E}^d(0)$: campo elétrico difratado calculado no ponto de observação 0, descomposto nas suas componentes *soft* e *hard*

$$\begin{pmatrix} E^d_\beta \\ E^d_\phi \end{pmatrix} = \begin{pmatrix} -D_s & 0 \\ 0 & -D_h \end{pmatrix} \begin{pmatrix} E^i_{\beta'} \\ E^i_{\phi'} \end{pmatrix}$$

com

$\begin{pmatrix} -D_s & 0 \\ 0 & -D_h \end{pmatrix}$ matriz $\bar{D}$ de coeficentes de difração, composta pelos coeficentes de difração *soft* e *hard* definidos em (**??**).

$\begin{pmatrix} E^i_{\beta'} \\ E^i_{\psi'} \end{pmatrix} = \begin{pmatrix} \vec{E}^i(Q) \cdot \hat{\beta}' \\ \vec{E}^i(Q) \cdot \hat{\phi}' \end{pmatrix}$ componentes *soft* e *hard* do campo elétrico incidente na aresta $\vec{E}^i(Q)$ onde os vetores unitários $\hat{\beta}'$ e $\hat{\phi}'$ são definidos em (4-60).

$\hat{\beta}$: conforme definidos em (4-60).

A^d : fator de divergência do tubo de raios difratados.

d_2 [m]: distância entre o ponto Q e o ponto de observação O.

Coeficente de difração

No caso de arestas em obstáculos formados por material condutor perfeito, os coeficentes de difração são obtidos da expressão assintótica para a solução exata do campo espalhado.

Para o caso de arestas em cunhas formadas por semi-planos de condutividade finita, pode-se utilizar soluções aproximadas (semi-heuristicas) como em [51] e em [2]. Os coeficêntes de difração $D_{s,h}$ *soft* e *hard* respectivamente, são, então, definidos por:

$$D_{s,h}(L,\ \psi,\ \psi',\ \delta,\ n) = G_{0_{s,h}}\left(D_1 + \Gamma_{0_{s,h}}\ D_2\right) + G_{n_{s,h}}\left(D_3 + \Gamma_{n_{s,h}}\ D_4\right) \tag{4-61}$$

onde

PUC-Rio - Certificação Digital Nº 0310507/CB

$$D_1 = \frac{-e^{-j\pi/4}}{2n\sqrt{2k\ \pi}\ sen\ \delta}\ cot\left[\frac{\pi-(\psi-\psi')}{2n}\right] F\left[\ k\ L^{i}\ a^{-}(\psi-\psi')\right] \quad (4\text{-}62)$$

$$D_2 = \frac{-e^{-j\pi/4}}{2n\sqrt{2k\ \pi}\ sen\ \delta}\ cot\left[\frac{\pi-(\psi+\psi')}{2n}\right] F\left[\ k\ L^{r_0}\ a^{-}(\psi+\psi')\right] \quad (4\text{-}63)$$

$$D_3 = \frac{-e^{-j\pi/4}}{2n\sqrt{2k\ \pi}\ sen\ \delta}\ cot\left[\frac{\pi+(\psi-\psi')}{2n}\right] F\left[\ k\ L^{i}\ a^{+}(\psi-\psi')\right] \quad (4\text{-}64)$$

$$D_4 = \frac{-e^{-j\pi/4}}{2n\sqrt{2k\ \pi}\ sen\ \delta}\ cot\left[\frac{\pi+(\psi+\psi')}{2n}\right] F\left[\ k\ L^{r_n}\ a^{+}(\psi+\psi')\right] \quad (4\text{-}65)$$

com

ψ, ψ' e δ : definidos anteriormente (figura 4.16)

$n = \frac{2\pi-\psi}{\pi}$: fator de abertura da cunha, onde $\psi = \pi - acos(\hat{n}_1 \cdot \hat{n_2})$ representa o ângulo interior da cunha e $\hat{n}_{1,2}$ os vetores unitarios normais a cada face da cunha. O ângulo externo 4.16 da cunha é dado a partir de (4-66) por $n\pi = 2\pi - \psi$.

$F(x)$: A função de transição de Fresnel em (4-61), responsavel pela continuidade do campo total no entorno das fronteiras de sombra, é dado por:

$$F(x) = 2j\ \sqrt{x}\ exp(jx) \int_{\sqrt{x}}^{\inf} exp(-j\ \tau^2)\ d\tau \quad (4\text{-}66)$$

$L^{i,r_{0,n}}$: parâmetros de distancia, referentes às faces "0" e "n"

$$L^{i,r_{0,n}} = \frac{d_2\ (\rho_e^{i,r_{0,n}} + d_2)\ \rho_1^{i,r_{0,n}}\ \rho_2^{i,r_{0,n}}\ sen^2\ \delta}{\rho_e^{i,r_{0,n}}\ (\rho_1^{i,r_{0,n}} + d_2)(\rho_2^{i,r_{0,n}} + d_2)} \quad (4\text{-}67)$$

$\rho_e^{i,r_{0,n}}$: raio de curvatura da frente de onda incidente (refletida) no plano contendo o raio incidente (refletido) e o vetor $\hat{e}$ da aresta;

$\rho_{1,2}^{i,r_{0,n}}$: raios principais de curvatura das frentes de onda incidente e refletida.

$a^{\pm}$: representa uma medida da separação angular entre o ponto de observação e uma fronteira de incidencia ou reflexão.

$$a^{\pm}(\beta) = 2\ cos^2\left(\frac{2n\ \pi\ N^{\pm} - \beta}{2}\right) \quad (4\text{-}68)$$

onde $\beta = \phi \pm \phi'[rd]$ cuja representação usual é da forma $\beta^{\pm} = \phi \pm \phi'$ para denotar a soma e a substração envolvendo os ângulos ϕ e ϕ'. Na expressão (4-68), entretanto β é apresentado sem o sobreescrito $\pm$ apenas para ressaltar a inexistência de relação com os sobrescritos de $a(\beta)$ e de

PUC-Rio - Certificação Digital Nº 0310507/CB

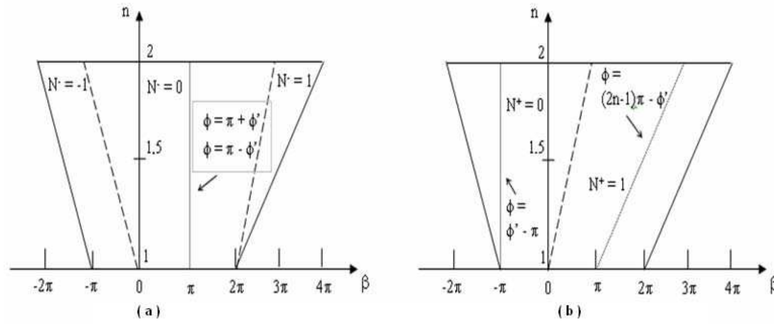

Figura 4.17: Variação de $N^{\pm}$ com linhas partilhadas às fronteiras de transição

N, estes sim, relacionados conforme segue: $N^{\pm}$ são os inteiros que mais fielmente satisfazem às seguintes equações:

$$2\ \pi\ n\ N^{+}\ -\ (\beta^{\pm})\ =\ \pi \tag{4-69}$$

$$2\ \pi\ n\ N^{-}\ -\ (\beta^{\pm})\ =\ -\pi \tag{4-70}$$

e para difração exterior ($1 < n \leq 2$), os valores são: $N^{-} = -1; 0; 1$ e $N^{+} = 0; 1$ conforme ilustrado na figura (4.17). O sobreescrito de $a(\beta)$ esta diretamente relacionado ao de N, ou seja, $a^{+} \leftrightarrow N^{+}$ e $a^{-} \leftrightarrow N^{-}$

Os ângulos ϕ e ϕ' são sempre medidos a partir da face "0" e são determinados, conforme a figura (4.16) por

$$cos\ \phi' \ = \ -\frac{\vec{s}_{proj} \cdot \hat{f}}{|\vec{s}_{proj}|} \quad \Rightarrow \quad \phi' = acos\left(-\frac{\vec{s}_{proj} \cdot \hat{f}}{|\vec{s}_{proj}|}\right) \tag{4-71}$$

$$\vec{s'}_{proj.} = \vec{s'} - (\vec{s'} \cdot \hat{e})\ \hat{e} \tag{4-72}$$

$$cos\ \phi \ = \ \frac{\vec{s}_{proj} \cdot \hat{f}}{|\vec{s}_{proj}|} \quad \Rightarrow \quad \phi = acos\left(\frac{\vec{s}_{proj} \cdot \hat{f}}{|\vec{s}_{proj}|}\right) \tag{4-73}$$

$$\vec{s}_{proj.} = \vec{s} - (\vec{s} \cdot \hat{e})\ \hat{e} \tag{4-74}$$

Um estudo detalhado da implementação da UTD para obstaculos de condutividade finita é apresentado em [2], onde são listados e comentados diversos aspectos complementares.

PUC-Rio - Certificação Digital Nº 0310507/CB

5
Método Híbrido

Existem dois enfoques do método híbrido de Traçado de Raios (RT), acoplado à UTD, combinado com o método de Diferenças Finitas no Dominio do Tempo (FDTD) e aplicado ao cálculo de cobertura em ambientes interiores, o primeiro formulado por Ying Wang [5] e o segundo por Sebastien Reynaud [4]. Neste capitulo são apresentados estes enfoques aplicados a problemas bi-dimensionais de forma a servir de fundamentação teórica para uma futura implementação de um novo modelo. Na parte final deste capitulo são propostas algumas idéias para esta implementação, de forma a proveitar as melhores características dos modelos existentes.

5.1
Modelo de Ying Wang

O cenário bi-dimensional a ser modelado é subdividido em duas regiões de forma que cada sub-região seja tratada com o método apropiado para essa região. Áreas preenchidas com objetos de tamanho elétrico maior que o comprimento de onda são tratadas a partir de RT/UTD e áreas com objetos heterogêneos e de tamanho elétrico menor ou igual que o comprimento de onda são tratadas a partir da FDTD.

5.1.1
Descrição do traçado de raios

O processo se inicia com o traçado de raios, o transmissor é modelado por uma fonte pontual, irradiando na direção de vértices obtidos de sub-divisões das faces de um icosaedro centrado na fonte, conforme descrito no apêndice A. Os objetos interiores ao domínio são modelados como blocos dielétricos ou condutores com parámetros definidos: espessura, fronteiras e parâmetros constitutivos, desta forma, paredes, teto, chão, janelas e portas são também caracterizados facilmente.

O progresso de cada raio emanado do transmissor é rastreado através do cenário e, detectada a interseção com o objeto mais próximo, o algoritmo gera gera um raio transmitido e um raio refletido. As informações do raio refletido

PUC-Rio - Certificação Digital Nº 0310507/CB

são salvas e o raio transmitido é traçado de forma similar aos raios provenientes da fonte. Este processo continua até que a intensidade do raio atija um limiar ou o raio deixe o cenário. No próximo passo, o raio refletido, cujas informações foram guardadas, é traçado de forma semelhante. O processo continua até esgotar as informações de raios refletidos salvas.

Para determinar o campo total em um ponto de observação, uma esfera de recepção (Apêndice A) é construída ao redor do ponto de recepção e, se o raio intercepta esta esfera o campo ao longo deste raio contribui para o campo total neste ponto.

Difração é outro mecanismo de propagação incluído no rastreamento de campo ao longo de raios em cenários interiores. A maioria das estruturas causando difração pode ser categorizada como: semiplano grosso (portas, janelas e divisórias) e cunha em ângulo reto como uma quina de parede. Para um tratamento adequado da difração por uma cunha dielétrica com perdas, é usada a UTD modificada [2, 51] para as duas estruturas consideradas. Quando detectada que uma quina do cenário é interceptada por um tubo de raios, o algoritmo de traçado de raios procura por pontos de recepção que estejam em linha de visada direta segundo o cone de difração. Os campos difratados são então calculados e passam a contribuír para o campo total nos pontos de recepção. A partir dai os raios difratados não são mais rastreados, isto é, não se consideram espalhamentos de segunda ordem envolvendo difrações.

5.1.2
Descrição da hibridização

Para a combinação com o método FDTD a região de interesse é fechada com uma caixa virtual, sobre esta caixa, a partir do campo externo rastreado a lo longo de raios incidentes, são establecidas fontes superficiais virtuais que excitarão os campos em seu interior.

Sempre que um raio intersepta a caixa, a posição de interseção, a direção do raio e o campo elétrico são salvos em um arquivo de dados, o qual é subseqüentemente usado como uma fonte de excitação, para aplicação do método FDTD no interior. Múltiplas caixas virtuais podem ser definidas de forma que a fonte de excitação em cada região fechada possa ser obtida com cada região sendo tratada separadamente por FDTD. A figura (5.1) ilustra um cenário 2D, com um transmissor Tx, onde uma área de interesse, com um objeto heterogeneo no interior é circundada por linhas tracejadas $A_1B_1C_1D_1$. Os quatro lados deste retângulo, representando os planos de incidência dos raios externos, correspondem também à interface entre as regiões de aplicação do método ótico e FDTD. Alguns raios típicos que colidem com a caixa virtual

PUC-Rio - Certificação Digital Nº 0310507/CB

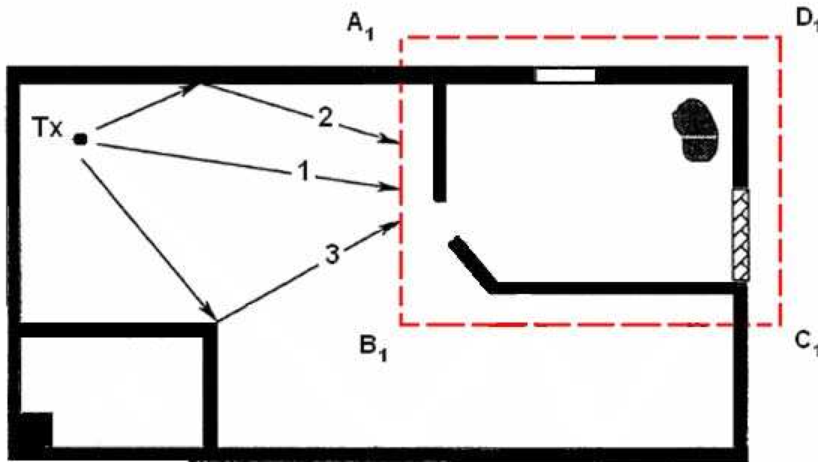

Figura 5.1: Cenário interior com áreas de interesse fechadas por caixas virtuais $A_1B_1C_1D_1$ e $A_2B_2C_2D_2$. Diferentes sombras e texturas são usadas para representar diferentes materiais [34]

PUC-Rio - Certificação Digital Nº 0310507/CB

são mostrados na figura (5.1)

O raio 1 representa um raio direto, o raio 2 um raio refletido e o raio 3 um raio difratado. O método FDTD será usado para calcular todos os efeitos de propagação e espalhamento dentro da caixa virtual.

A figura (5.2) correspondente ao dimensionamento da figura (5.1), ilustra uma onda de polarização TMz incidente segundo o raio 1 sobre o nó (i,j) no plano (A_1B_1). A distancia do nó a cada ponto de interseção de raios vizinhos é analisada e a metodologia da esfera de recepção (apêndice A) é utilizada para coletar os raios incidentes de interesse e calcular os campos correspondentes que são convertidos para o dominio do tempo após uma inversão adequada de Fourier [52].

O campo elétrico total em (i,j) pode, então ser calculado por:

$$\vec{E}(i,j) = \sum_{n=1}^{k} \vec{E}_n \tag{5-1}$$

onde $\vec{e}$ é o campo elétrico dependente do tempo correspondente ao enésimo raio no nó (i,j). Como mostrado na figura (5.2) o dominio atual $(A_1B_1C_1D_1)$ é redimensionado para $(A'B'C'D')$, o qual corresponde á técnica de inserção de uma onda plana (TFSF), vista no capitulo (3). Pode-se escrever:

$$\begin{aligned} \vec{E}_{total} &= \vec{E}_{inc} + \vec{E}_{esplh} \\ \vec{H}_{total} &= \vec{H}_{inc} + \vec{H}_{esplh} \end{aligned} \tag{5-2}$$

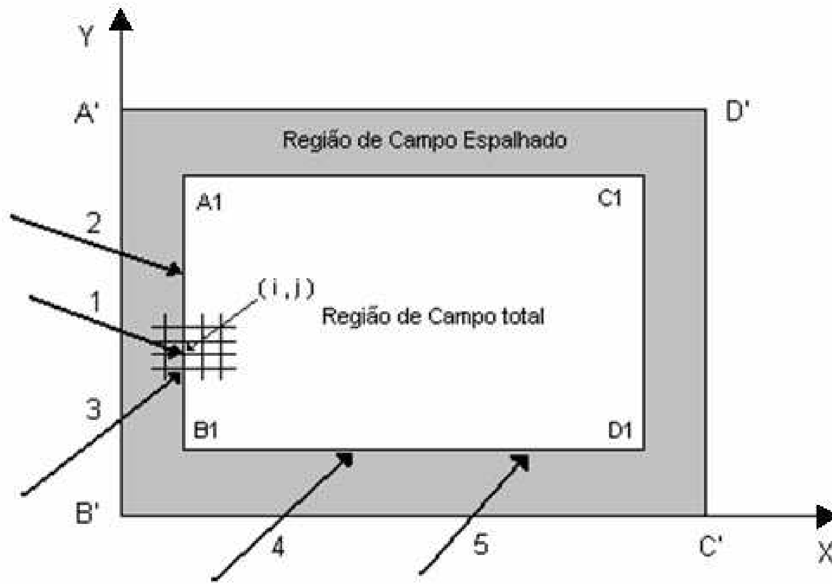

Figura 5.2: Redimensionamento do retângulo $A_1B_1C_1D_1$ da figura (5.1), ilustrando a interface entre os métodos RT/FDTD [34]

PUC-Rio - Certificação Digital Nº 0310507/CB

onde $\vec{E}_{total}(\vec{H}_{total})$ é o campo elétrico (magnético) total, $\vec{E}_{inc}(\vec{H}_{inc})$ é o campo elétrico (magnético) incidente, $\vec{E}_{esplh}(\vec{H}_{esplh})$ é o campo elétrico (magnético) espalhado e os valores dos campos incidentes são introduzidos ao longo da interface entre estas duas regiões.

A figura (5.3) ilustra o esquema da fonte de excitação, observando-se as componentes de campo ao redor do nó (i,j) ao longo do plano incidente (A_1B_1)

Segundo as equações de atualização do método FDTD, o cálculo da componente $E_z(i,j)$ no paso $(n+1)$, denotado por $E_z^{n+1}(i,j)$, requer o conhecimento do valor das componentes $H_y(i-1/2,j)$, $H_y(i+1/2,j)$, $H_x(i,j-1/2)$, e $H_x(i,j+1/2)$. Observe-se na figura (5.3) que $E_z(i,j)$, $H_y(i+1/2,j)$, $H_x(i,j-1/2)$ e $H_x(i,j+1/2)$ estão na região de campo total enquanto $H_y(i-1/2,j)$ está na região de campo espalhado. Usando o principio de linearidade das equações de Maxwell, estas são reescritas na forma [37]:

$$
\begin{aligned}
E_{z,tot}^{n+1}(i,j) \;=\; & E_{z,tot}^{n}(i,j) + \frac{\Delta t}{\epsilon_0\,\Delta} \times \Big[H_{y,tot}^{n+1/2}(i+1/2,j) - H_{y,esplh}^{n+1/2}(i-1/2,j) + \\
& H_{z,tot}^{n+1/2}(i,j-1/2) - H_{z,tot}^{n+1/2}(i,j+1/2) \Big] - \frac{\Delta t}{\epsilon_0 \Delta} H_{y,inc}^{n+1/2}(i-1/2,j)
\end{aligned} \tag{5-3}
$$

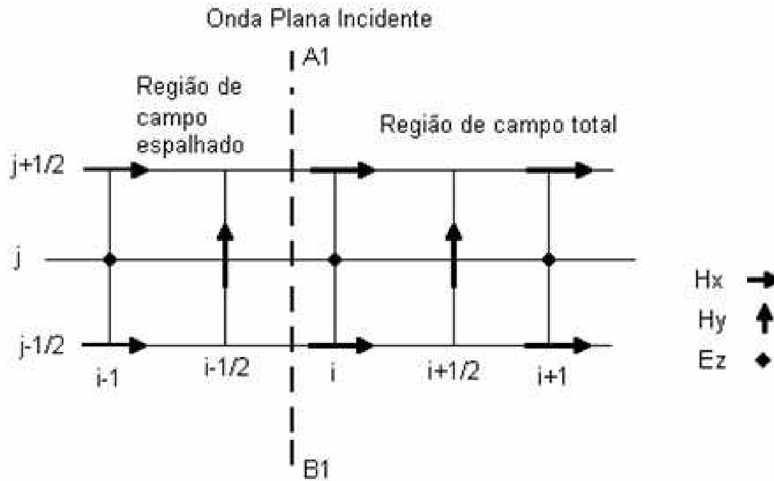

Figura 5.3: Esquema da fonte de excitação (adaptada de [34])

$$\begin{aligned} H^{n+1/2}_{y,esplh}(i-1/2,j) &= H^{n}_{y,esplh}(i-1/2,j) + \frac{\Delta t}{\mu_0\ \Delta} \times \left[E^{n}_{z,tot}(i,j) - E^{n}_{z,esplh}(i-1,j) + \right. \\ &\left. H^{n+1/2}_{z,tot}(i,j-1/2) - H^{n+1/2}_{z,tot}(i,j+1/2)\right] - \frac{\Delta t}{\mu_0 \Delta x} E^{n}_{z,inc}(i,j) \end{aligned} \quad (5\text{-}4)$$

Observe que as equações (5-3) e (5-4), correspondente ao termo de correção do campo incidente. Com a informação (conhecida) da direção do raio e do campo elétrico incidentes, o campo magnético incidente pode ser outro calculado. As outras fontes de excitação ao longo de B_1C_1, C_1D_1 e D_1A_1, podem ser calculadas de maneira similar.

Depois de especificar a distribuição de campo incidente sobre os planos de incidência, o campo que se propaga dentro das regiões fechadas é então calculado via FDTD. Wang [34] usa condições de fronteira absorventes de MUR, que são aplicadas às equações de atualização sobre as quatro fronteiras: $A'B'$, $B'C'$, $C'D'$ e $D'A'$. Assim a distribuição de campo dentro da região é obtida.

5.2 Modelo de Sebastien Reynaud

Neste modelo híbrido RT/FDTD análogamente á técnica de Wang [34], o cenário é dividido em duas sub-regiões e a técnica de traçado de raios, aliada à UTD, é usada para modelar a propagação da onda nos quartos, corredores e areas com espalhadores de dimensões grandes quando comparadas com o comprimento de onda, enquanto a FDTD é usada para calcular a matriz de

PUC-Rio - Certificação Digital Nº 0310507/CB

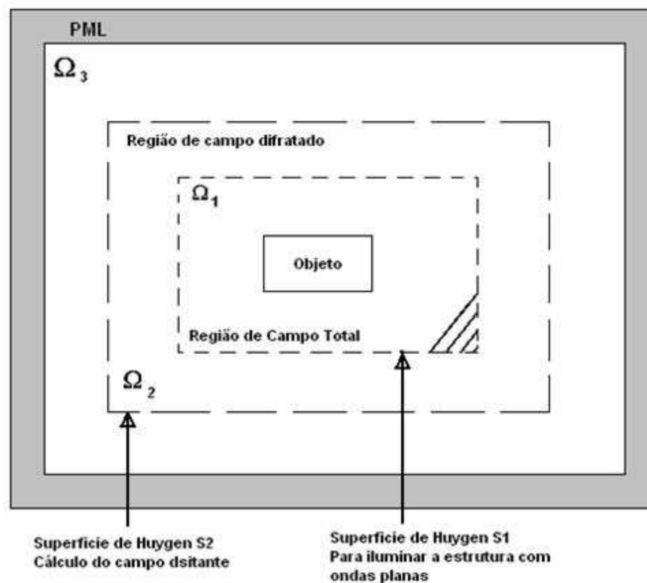

Figura 5.4: Divisão do dominio computacional em regiões para aplicação do método

espalhamento de estruturas pequenas e mais complexas.

5.2.1
Descrição do método hibrido

Reynaud na descrição de seu método enumera dois passos:

- O método FDTD é usado para a determinação da matriz de espalhamento da estrutura estudada.
- A matriz de espalhamento é incluída na aproximação assintótica como uma generalização do coeficente de difração.

Determinação da matriz de espalhamento

A determinação da matriz de espalhamento é calculada usando FDTD em conjunto à camada absorvente PML, a técnica campo total / campo espalhado (TFSF) e a técnica Campo Próximo / Campo Distante (NFFF) revistas no capitulo 3. A Figura (5.4) ilustra as diferentes regiões dentro do dominio computacional.

Primeramente é iluminado o objeto de interesse por uma onda plana pulsada e, seguindo o principio de Huygens as correntes equivalentes existentes sobre a superficie que circunda o objeto espalhador geram um campo incidente dentro da região ω_1. Depois de interagir a onda com o objeto, os campos calculados na região 2 , fora de S_1, correspondem aos campos espalhados pela estrutura. Através de uma transformação de Fourier, se obtem o campo

PUC-Rio - Certificação Digital Nº 0310507/CB

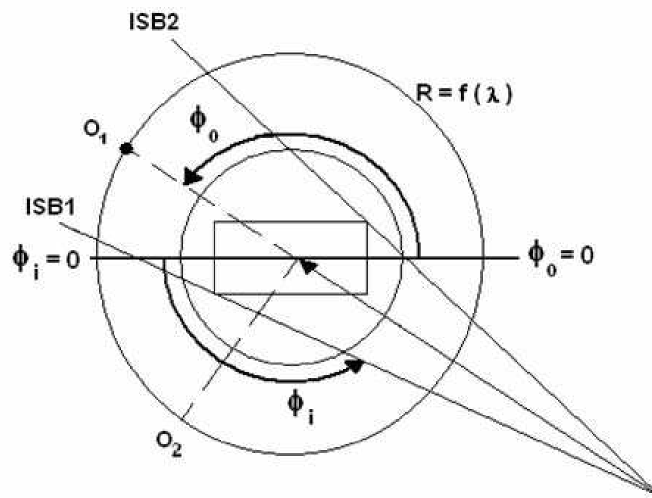

Figura 5.5: Principio de cálculo da matriz de espalhamento [4]

PUC-Rio - Certificação Digital Nº 0310507/CB

espalhado na região campo próximo, no domínio da freqüência, e, para obter o campo distante, usa-se outra caixa de Huygen S_2 ao redor da primeira. As correntes equivalentes elétrica e magnética irradiam para fora desta segunda superfície e, devido à transformação campo próximo / campo distante, obtem-se o campo distante espalhado. Conhecendo o campo incidente no centro de fase P escolhido para a transformação campo próximo / campo distante, deduz-se uma matriz de espalhamento no domínio da freqüência, para uma freqüência dada e um ângulo de iluminação,reunindo os coeficientes de difração para vários ângulos de observação.

$$D\ (\phi_i, \phi_0, f) = \frac{E_{distante}(\phi_i, \phi_0, f)}{E_{inc}^{onda\ plana}(P)} \tag{5-5}$$

onde:

ϕ_i, e ϕ_0 ângulos de incidência e observação calculados em relação às linhas $\phi_i = 0$ e $\phi_0 = 0$, respectivamente, na figura 5.5, onde ISB1 e ISB2 representam as fronteiras de sombra.

$E_{distante}(\phi_i, \phi_0, f)$ campo elétrico distante, obtido na direção ϕ_0.

$E_{inc}^{onda\ plana}(P)$ campo incidente de ondas planas a partir do centro de fase P da estrutura em estudo.

θ_1, θ_2 Pontos de observação.

$f(\lambda)$

Correções da matriz de espalhamento

Em concordancia com o principio de Huygens o campo incidente está confinado na região ω_1 figura (5.4) e na região ω_2 o campo corresponde ao campo total menos o campo incidente. O campo físico irá consequentemente ser falso quando o punto de observação estiver localizado dentro da região de sombra. De fato, pode-se compensar este erro adicionando um componente corretor ao coeficente de difração para direções de interesse como a região de sombra. Obviamente, este fator irá depender do ponto de emissão já que as regiões de sombra mudam com a posição da fonte. Mais precisamente, o fator de correção é especificado por:

$$E_{corr}(r, s') = E_{inc}^{onda\ plana}(P)\frac{\varphi(r)}{\varphi(s')} \tag{5-6}$$

onde r, s' são as distancias do transmissor ao receptor e ao centro de fase, respectivamente. $\varphi(r)$ e $\varphi(s')$ são funções de Green $\varphi(R) = \frac{e^{-jkR}}{R}$ onde k é o vetor de onda.

Finalmente no código RT, o objeto ira a ser substituído por uma matriz de espalhamento e vista como um simples ponto de difração.

O maior problema com o modelo de Reynaud [4] está relacionado com o tamanho das matrizes de espalhamento, pois ocupam muita memória. Reynaud aborda este problema, já critico em dois dimensões, usando técnicas de compressão baseadas em Wavelets [4].

5.3
Métodologia proposta para um novo modelo

A metodologia proposta a seguir está baseada principalmente nos dois modelos anteriores, em especial no modelo de Wang [34]. O cenário escolhido é estudado de forma a identificar regiões propicias a serem tratadas com os métodos RT e FDTD figura (5.6).

Identificam-se estas regiões através do uso da técnica de discretização espacial Quadtree [3] permitindo a determinação dos domínios mínimos exatos que circundam os espalhadores cujos tamanhos sejam menores ou iguais que o comprimento da onda . Consequentemente, estes dominios deverão ser tratados por FDTD, permitiendo, assim, um uso eficiente da memoria. Identificados os domínios FDTD, por descarte as demais regiões corresponderão ao tratamento por RT.

As superficies que servem de fronteira entre estes dois domínios formam a caixa de Huygens que servirá como interface entre os dois métodos. As regiões com espalhadores maiores que o comprimento de onda serão tratados por RT. Na escolha da camada absorvente que circunda o dominio computacional,

PUC-Rio - Certificação Digital Nº 0310507/CB

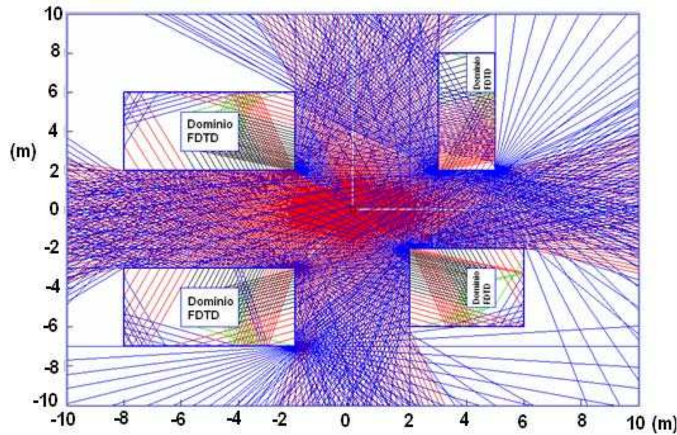

Figura 5.6: Subdomínios RT/FDTD

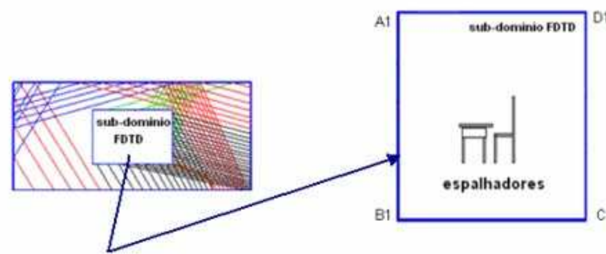

Figura 5.7: Detalhe do subdomínio FDTD - Caixa $A_1B_1C_1D_1$

PUC-Rio - Certificação Digital Nº 0310507/CB

recomenda-se usar a UPML [43] de forma a otimizar o usa da memória e, no caso de limitações severas de memória, recomenda-se usar a camada absorvente de Mur [40] de segunda ordem, com a qual se obtém resultados aceitáveis.

Descrição do método híbrido

Raios são emitidos do transmissor com igual espaãmento angular seguindo o modelo descrito no Apêndice A. Os objetos no interior do cenário são modelados e descritos por seus parâmetros constitutivos. Para determinar o campo total (somatório das contribuções de todos os tipos de raios) em um ponto, uma esfera de recepção é criada na posição do receptor conforme descrito no aoeênci A.

As fontes que contribuem para o campo no subdomío FDTD são externas ao próprio subdominio são levadas em conta usando o principio de equivalencia com a formulação campo total / campo espalhado (TFSF) vista no capitulo 3,com as correntes superficiais calculadas apartir do campo eletromagnético incidente (obtido via RT) por:

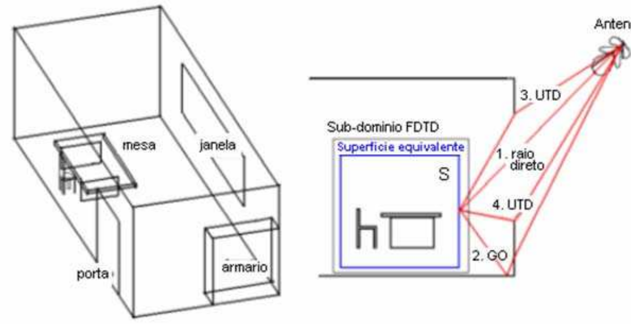

Figura 5.8: Fontes externas que contribuem para o campo no subdomínio FDTD

$$\vec{J}_e = -\hat{n} \times \vec{H}_1$$
$$\vec{J}_m = \hat{n} \times \vec{E}_1 \quad (5\text{-}7)$$

PUC-Rio - Certificação Digital Nº 0310507/CB

A região de intéresse é enclausurada por uma caixa virtual 3D (caixa de Huygens) composta por superficies e passa a ser considerada como um objeto dentro do algoritmo de RT.

Na formulação numérica de RT/FDTD, cada superficie equivalente é posicionada de forma a coincidir com os planos da malha onde as componentes de campo elétrico são definidas.

Cada uma destas superficies é subdividida em pequenas faces quadradas de forma que, tôda vez que um raio intersepta a caixa de Huygens, são salvos os dados da posição de interseção, da direção do raio e o campo incidente é avaliado em cada uma destas faces. A fig.(5.9) ilustra este processo, para uma caixa de huygens usada como exemplo. O campo incidente $(\vec{E}_i, \vec{H}_i)$, é então transformado para o dominio do tempo visando seu aproveitamento pelo método FDTD

Depois de especificar a distribução dos campos incidentes, sobre os planos de incidência, as ondas se propagam na região de campo total e a tecnica de campo total / campo espalhado é utilizada. Os campos calculados por FDTD produzirão informação sobre o campo próximo ao objeto e, utilizando-se o principio de equivalencia (figura 5.10), pode-se calcular o campo distante.

No processo, aplicando-se a transformada de Fourier discreta (DFT) às correntes superficiais no domínio do tempo, obtem-se as correntes superficiais no dominio da frequencia. A seguir, calcula-se o par de vetores auxiliares:

$$\vec{N} = \int_{S'} \vec{J}_s \; e^{jkr' \; cos\varphi dS'} \quad (5\text{-}8)$$

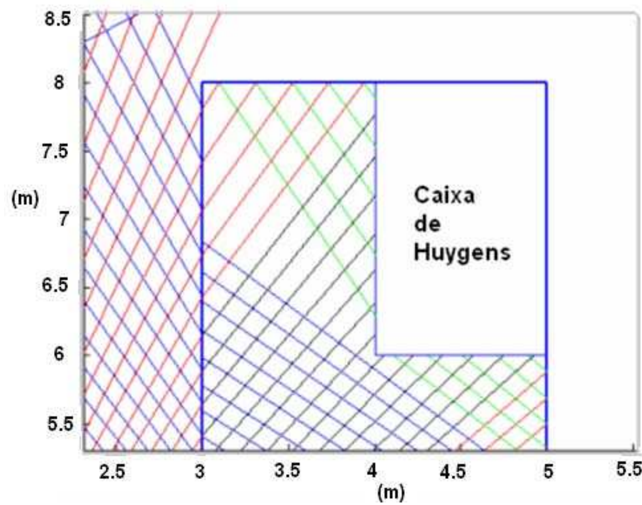

Figura 5.9: Fontes externas que contribuem para o campo no subdomínio FDTD

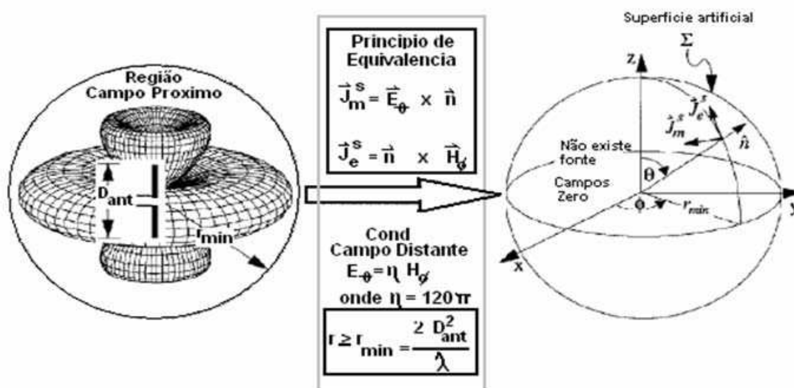

Figura 5.10: Principio de Equivalência [53]

PUC-Rio - Certificação Digital Nº 0310507/CB

$$\vec{L} = \int_{S'} \vec{M}_s \; e^{jkr' \; cos\varphi dS'} \tag{5-9}$$

e definem-se os potenciais vetores:

$$\vec{A} = \mu \frac{e^{-jkr}}{4\pi \; r} \; \vec{N} \tag{5-10}$$

$$\vec{F} = \epsilon \frac{e^{-jkr}}{4\pi \; r} \; \vec{L} \tag{5-11}$$

Finalmente poderemos determinar a intensidade de radiação em campo distante através de [37]:

$$\vec{E} = -j\omega\vec{A} - \frac{j\omega}{k^2} \; \nabla \; (\nabla \cdot \vec{A}) - \frac{1}{\epsilon} \; \nabla \times \vec{F} \tag{5-12}$$

$$\vec{H} = -j\omega\vec{F} - \frac{j\omega}{k^2} \; \nabla \; (\nabla \cdot \vec{F}) - \frac{1}{\epsilon} \; \nabla \times \vec{A} \tag{5-13}$$

6
Resultados

Este trabalho analisou o método FDTD a ser aplicado conjuntamente com o método RT/UTD em uma abordagem híbrida para o cálculo de cobertura em ambientes interiores. Apos uma revisão dos conceitos correspondentes aos métodos com ênfase no método FDTD e técnicas complementares (PML, UPML, TFSF, NFFF), foram implementados exemplos que validem os códigos desenvolvidos. Estes e aqueles asociados ao traçado de raios, fizeram uso otimizado da memória, dividiendo a implementação em três fases: pré-processamento, núcleo e pós-processamento. Em cada fase são salvos em um arquivo somente os dados necessários para a próxima fase, sendo liberada a memória alocada a variáveis supérfluas.

6.1
Cálculo da Cobertura numa WLAN usando FDTD [54]

Ambientes servidos por “redes locais sem fio” (WLAN) são em geral, preenchidos por um número grande de objetos que podem degradar a propagação do sinal de forma variável em curtos períodos de tempo. É tambem comum que os objetos presentes nestes ambientes tenham formas e sejam constituídos por materiais heterogêneos, alguns dos quais de dimensões menores ou iguais ao comprimento de onda.

Neste trabalho utilizou-se o método FDTD-2D conjuntamente com a técnica de camada absorvente PML [42] para calcular a cobertura de um ambiente interior. Os resultados obtidos permitem identificar os fenômenos físicos de reflexão, difração e transmissão, assim como a identificação de áreas de atenuação usando um mapa de cores, de forma a permitir uma melhor tomada de decisões na instalação de uma rede sem fio.

6.1.1
Modelo da fonte

A fonte (pontual) considerada é excitada por um pulso gaussiano conforme a densidade de corrente em (6-1), e posicionada em $(7,5;7,5)$), posição que não corresponde exatamente ao centro do cruzamento na figura (6.1):

PUC-Rio - Certificação Digital Nº 0310507/CB

$$J_z(t) = J_{max}\ exp(-\alpha[t - 6\ \eta\Delta t]^2) \tag{6-1}$$

onde J_{max} amplitude máxima é fixada em $500A/m^2$, $\alpha = (0,5/(\eta\Delta t))^2$ Δt valor do passo no tempo é igual a $28,32x10^{-12}$s e η é constante que define a característica de decaimento do pulso, fixada em 128.

6.1.2 Modelo do cenário

De forma a observar os mecanismos de propagação das ondas eletromagnéticas considerou-se um cenário (figura (6.1)) de $15x15m$ onde as paredes são representadas por segmentos de linha pretos e as portas por segmentos verdes, caracterizadas por parâmetros especificados na tabela 6.1 As portas apresentam o mesmo tamanho $(1x0,25)$m também é uniforme.

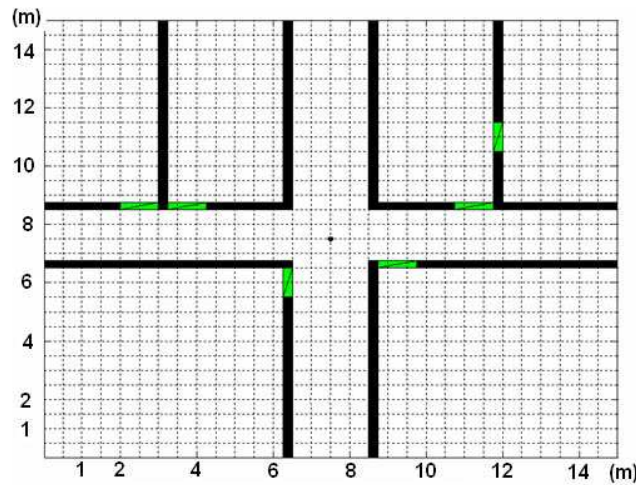

Figura 6.1: Cenário em ambiente interno (15x15m), com paredes divisórias em preto e portas em verde.

	madeira	concreto	PEC	Gesso
Permissividade relativa	4,0	7,85	1,0	9,0
Condutividade $[(\Omega - m)^{-1}]$	$0,333x10^{-7}$	$0,0$	$1,0x10^{-7}$	$0,0$

Tabela 6.1: Parâmetros característicos dos materiais considerados nas simulações [57]

6.1.3 Propagação do pulso na presença de paredes de diferentes materiais

(a.) *Paredes e portas condutoras elétricas perfeitas (PEC)*

Considerando paredes e portas PEC no ambiente da figura (6.1) e situando a antena transmissora no centro do cruzamento, espera-se que o

PUC-Rio - Certificação Digital Nº 0310507/CB

sinal se propague somente no corredor. Este comportamento é verificado nos diagramas de cobertura corresponde à propagação da componente de campo Ez nos instantes de $15ns$, $19ns$, e $25ns$ ilustrados nas figuras (6.2) Observe-se que os corredores da figura (6.1) não apresentam a mesma largura, o que explica a pequena perda de simetria observada na propagação da energia.

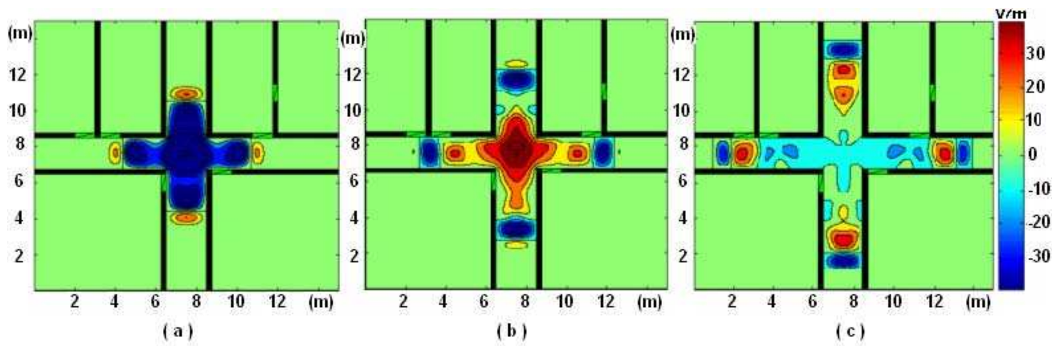

Figura 6.2: Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes e portas PEC, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns$.

(b.) *Paredes PEC em ausência de portas*

Para esta segunda configuração, consideram-se paredes PEC na ausência de portas, como esperado a energia é transmitida ao interior dos cômodos a través das portas, estes resultados são obtidos para os mesmo instantes de tempo considerados no caso (a), vale dizer (15 ns, 19 ns e 25 ns) e apresentados nas figuras (6.3)

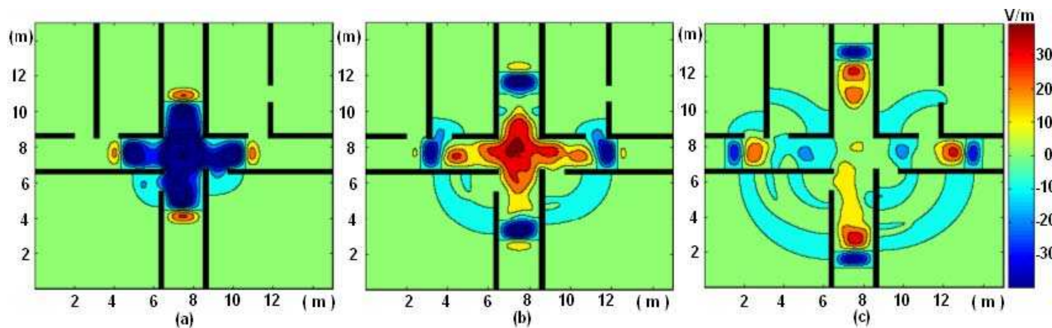

Figura 6.3: Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes PEC e sem portas, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns$.

(c.) *Paredes de concreto com portas de madeira*

Esta configuração corresponde a um caso mais realista e se observa a transmissão da energia através de paredes e portas, sendo maior a transmissão pelas portas. Os resultados apresentados na figura (6.4) consideram os mesmos instantes de tempo das figuras anteriores.

PUC-Rio - Certificação Digital Nº 0310507/CB

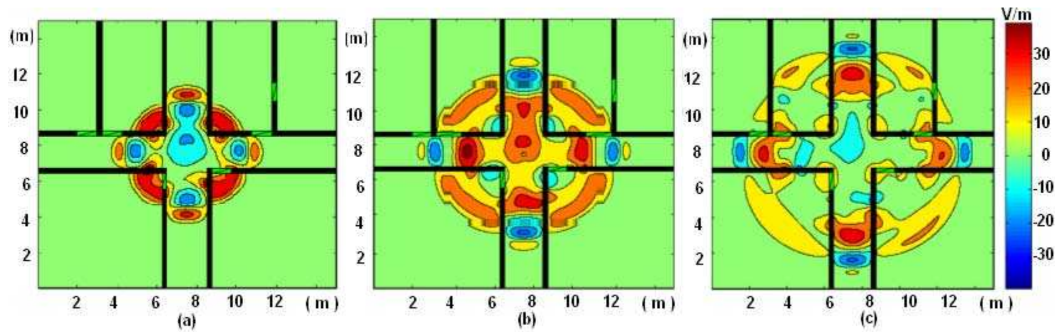

Figura 6.4: Cobertura pela componente de campo Ez do cenário da figura (6.1) constituído por paredes PEC e sem portas, para os instantes de tempo: $(a)15ns, (b)19nse(c)25ns$.

6.1.4
Cálculo da potencia - sistema de antenas SIMO.

Na figura (6.5) apresentam-se um sistema de antenas SIMO (Simple Input - Multiple Output) distribuído no cenário da figura (6.1). com as paredes modeladas como sendo de concreto e as portas de madeira. A posição da antena transmissora corresponde às coordenadas $Tx(7,5;7,5)$ e as posições das antenas receptoras correspondem às coordenadas :$Rx1(7,5;5.0)$, $Rx2(7,5;3.5)$, $Rx3(7,5;2.0)$; $Rx4(5.5;13,0)$. As figuras (6.6.a -6.6.d) ilustram o comportamento, em função do tempo decorrido, da potência normalizada recebida em cada uma das antenas.

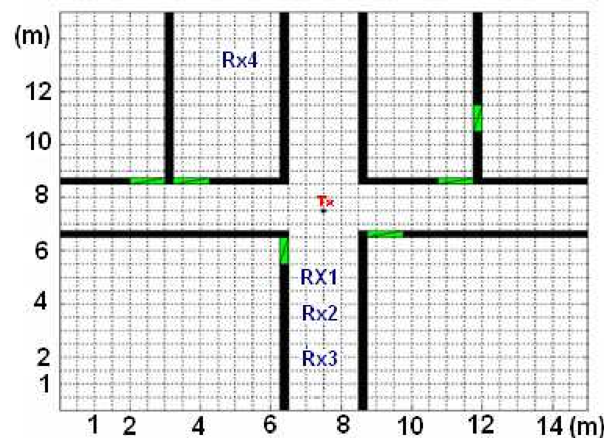

Figura 6.5: Distribução das antenas no cenário considerado

O gráfico da figura (6.6.a) revela que o primeiro sinal em $Rx1$ é recebido aproximadamente aos $5,5ns$ e logo se observa uma primeira inflexão, correspondente ás contribuções provenientes das primeiras reflexões nas paredes e, com as primeiras difrações nas quinas, a curva vai mudando conforme um diagrama de interferência. Aproximadamente aos 14 ns, observa-se um desvacimento produzido pela propagação em maior proporção através das portas

PUC-Rio - Certificação Digital Nº 0310507/CB

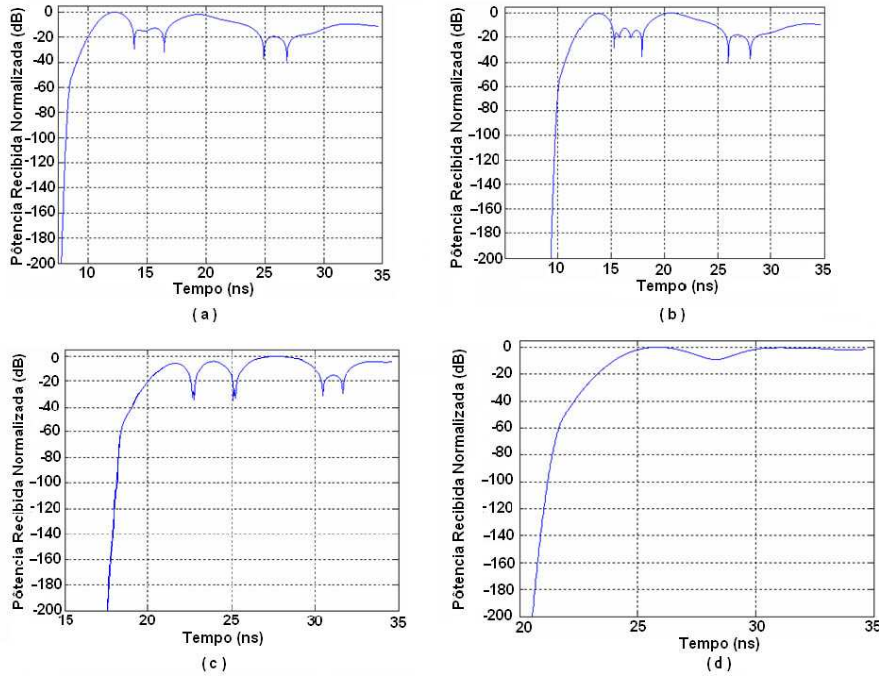

Figura 6.6: Pôtencia normalizada recebida em (a)Rx1, (b)Rx2, (c)Rx3, (d)Rx4

(figura (6.4).) decorridos alguns ns, observa-se novo aumento na potência recebida por causa das contribuções construtivas provenientes de reflexões. A próxima queda na pôtencia recebida é observada aos $24,7ns$, quando a energia novamente se propaga em maior proporção pelas portas.

A figura (6.6.b) apresenta a potência recebida na antena $Rx2$ distante $1,5m$ da antena $Rx1$ revelando que o primeiro sinal é recebido aproximadamente $3,5ns$ depois que em Rx1 e, em geral, o comportamento ao longo do tempo é semelhante da figura (6.6.a) devido á proximidade das antenas.

A figura (6.6.c) ilustra o comportamento da potência recebida na antena receptora $Rx3$, separada $3,0m$ da antena $Rx1$. O primeiro sinal é recebido em Rx3 aproximadamente $12,2ns$ depóis que em $Rx1$ e $7,5ns$. depois de de $Rx2$. Este sinal apresentamaior número de contribuções mas, em geral também, apresenta comportamento similar aos anteriores devido á proximidade daquelas antenas.

A figura (6.6.d) apresenta o comportamento da potência recebida na antena $Rx4$, mais distante das demais e em uma posição de obstrução com relação á antena transmissora. A energia que atinge esta antena é menor e mais estável devido á redução das contribuções interferentes.

PUC-Rio - Certificação Digital Nº 0310507/CB

6.2
Emprego do método FDTD para análise de cobertura em ambientes interiores [55]

Desde sua introdução, no início dos anos 80, o número de aplicações para WLAN´s vem crescendo. Estas aplicações usam as faixas de freqüências ISM (Industrial, Scientific and Medical) de $2,45GHz$ e $5,8GHz$, a faixa licenciada operando em $18-19GHz$ [58]-[59] e a faixa de freqüência das ondas milimetricas [60].

Metodologias usuais de cálculos de cobertura em ambientes interiores compreendem o rastreamento ótico do campo eletromagnético que fornece resultados satisfatórios no regime de freqüências altas. Para sistemas de faixa muito larga como, por exemplo, UWB [32], técnicas distintas precisariam ser utilizadas para freqüências baixas e intermediárias, tornando atraente a utilização de técnicas de cálculo do campo diretamente no domínio do tempo e sua posterior inversão, por Fourier, para as freqüências desejadas [61].

Um método alternativo que nos permite obter uma boa analise de cobertura em ambientes interiores é o método de diferenças finitas no domínio do tempo, FDTD o qual foi usado neste trabalho, conjuntamente com a camada uniaxial perfeitamente casada, UPML (Uniaxial Perfect Matched Layer)[42].

6.2.1
Modelo do cenário

O cenário a ser considerado é apresentado na figura (6.7). É um cenário de $15x15x3$ m típico (embora sem mobilia), representando a distribução de cômodos em um ambiente interior, com paredes e portas que podem ser de concreto e madeira, escolhidas através de seus parámetros característicos, especificados na tabela 6.1. As portas apresentam o mesmo tamanho $(1x2x0,25m)$, assim também como a espessura das paredes $(0,25\ m)$ é uniforme.

Na implementação do método FDTD o domínio computacional é discretizado através de uma malha uniforme com tamanho do passo espacial (Δx) igual a $8x10^{-2}m$. e o tamanho do passo temporal (Δt) foi calculado usando o critério de Courant, passos esses, utilizados nas aproximações por diferencias centrais das equações de Maxwell. Para a trucagem do domínio computacional, se escolheu trabalhar com a UPML conforme descrito no capitulo 2.

6.2.2
Modelagem da fonte

Uma simples antena dipolo é usada como fonte, ilustrada na Figura (6.8). Esta consiste de dois braços de metal pelos quais flui corrente gerando

PUC-Rio - Certificação Digital Nº 0310507/CB

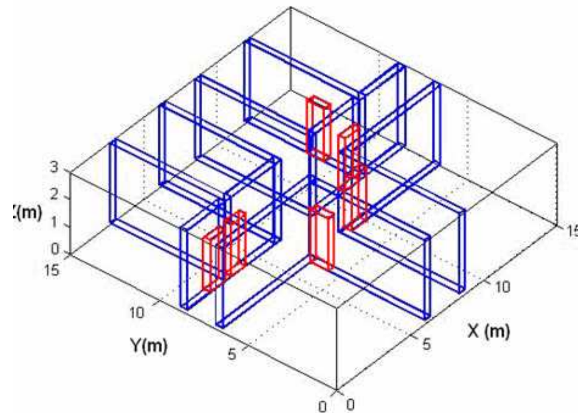

Figura 6.7: cenário em ambiente interior ($15x15x3\ m$), com paredes divisórias em azul e portas em vermelho

radiação (Figura 6.8.a) e para simular os braços de metal fixou-se os valores da componente de campo Ez igual a zero nos pontos correspondentes à posição dos braços dentro da malha. A excitação, um pulso gaussiano, conforme equação (6-2), é inserida no "gap" e a componente Ez nesta posição vai assumindo valores ao longo do tempo. A Figura (6.8.b). mostra a radiação do dipolo no plano vertical para instantes iniciais e a Figura (6.8.c). mostra a radiação do dipolo nos planos horizontal e vertical, para os mesmos instantes iniciais da Figura (6.8.b).

PUC-Rio - Certificação Digital Nº 0310507/CB

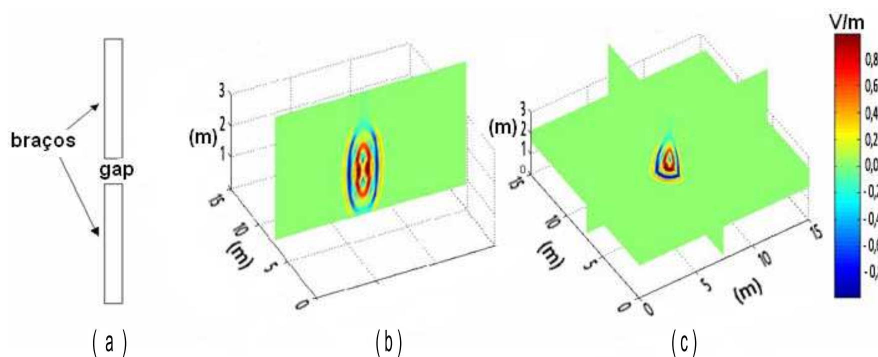

Figura 6.8: Dipolo usado no programa (a) modelagem, (b) radiação no plano vertical, (c) radiação espacial.

$$Ez(t) = 100\ exp\left[-18\ \left(\frac{t-D}{pW}\right)^2\right] \tag{6-2}$$

onde $D = 30\Delta$ é o deslocamento da origem do pulso gaussiano, $pW = 10\Delta t$ é a variação do tempo utilizada pertence ao intervalo $0 \leq t \leq 6000\ \Delta t$.

A figura (6.9) ilustra o pulso gaussiano utilizado no gap (a) e seu espectro de frequencia (b).

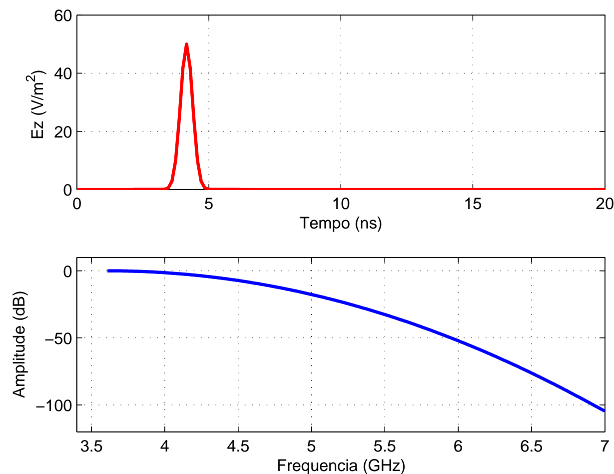

Figura 6.9: Pulso gaussiano utilizado no gap (a) dominio do tempo (b) espectro de freqüência.

6.2.3 Simulações e resultados

Foram consideradas duas configurações, a primeira constituída por paredes e portas PEC com o objetivo de validar a implementação do método e uma segunda configuração constituída por paredes de concreto e portas de madeira, caso de interesse. Estas configurações foram implementadas usando os valores dos parâmetros característicos da tabela 6.1

Observa-se na Figura (6.10.a) que o sinal sofre interferências provenientes das difrações nas quinas e das reflexões nas paredes e portas, representadas por cores próximas do vermelho para interferências construtivas e em azul intenso para interferências destrutivas.

a. *Cenário constituído por paredes e portas PEC*

   Considerando paredes e portas perfeitamente condutoras (PEC), os cômodos se constituem em ambientes fechados onde a energia não penetra. A energia irá, então, se propagar apenas nos corredores, conforme coberturas ilustradas nas Figuras 6.10, à medida que o tempo evolui.

   Na Figura (6.10.b) é possível observar que o sinal interage com as camadas absorventes sem produzir reflexões espúrias. Como a fonte é constituída de um único pulso, observa-se (código de cores), que ao passar o tempo, a intensidade de campo recebido vai decrescendo isto é, correspondendo apenas ao campo espalhado na falta de nova excitação.

b. *Cenário constituído por paredes de concreto e portas de madeira*

   Esta segunda configuração corresponde a uma situação realista, onde se consideram paredes de concreto com portas de madeira, dispostas con-

PUC-Rio - Certificação Digital Nº 0310507/CB

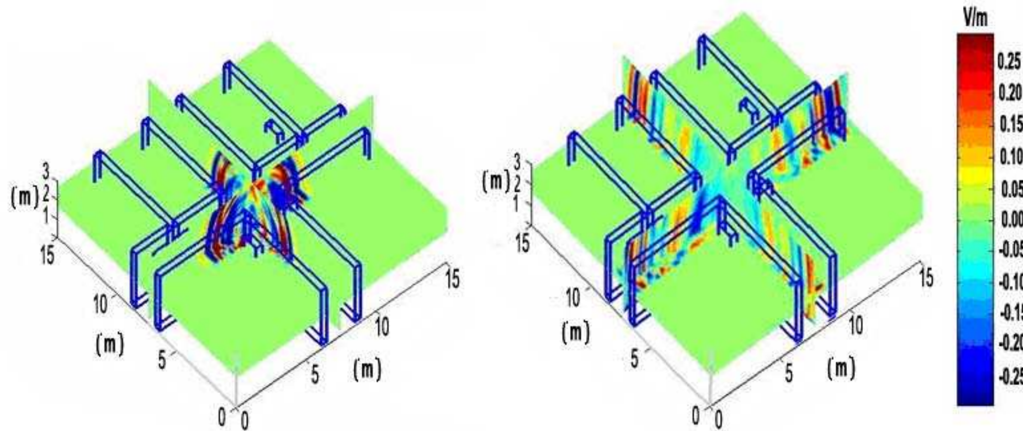

Figura 6.10: Cobertura pela componente Ez de ambiente com paredes e portas PEC: (a) primeiros instantes $13,85ns$ da propagação da energia, (b) aos $27,13ns$ a energia atinge as camadas UPML sendo absorvidas.

forme ilustrado na figura (6.7); o transmissor, como nos casos anteriores, está situado no centro do cenário.

PUC-Rio - Certificação Digital Nº 0310507/CB

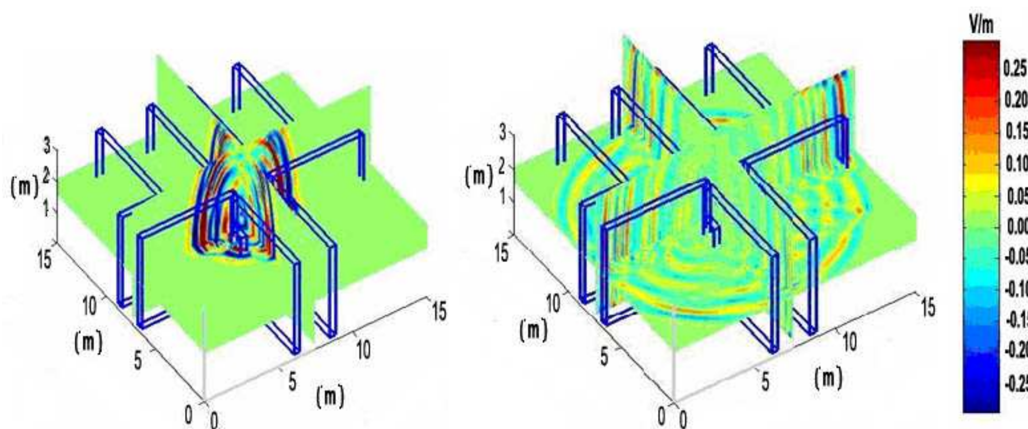

Figura 6.11: Cobertura pela componente Ez de ambiente com paredes de concreto e portas de madeira: (a) decorridos $13,85ns$, (b) decorridos $27,13ns$.

Na figura (6.11.a) é possível observar a transmissão da energia através da porta próxima à quina (ponto de observação de coordenadas (5; 5; 1,5)) representada pela cor vermelha intensa situada exatamente dentro da porta. A maior transmissão de energia pela porta deve-se à condutividade da madeira ser menor que a do concreto. A quantidade de energia passando pela porta é pouca devido a sua largura reduzida, mas de magnitude suficiente para produzir perturbações significativas na simetria do diagrama de cobertura como se observa na figura (6.11.a). Na figura (6.11.b), especificamente na posição $(15; 7,5; 2)$ é possível observar maior energia com relação ao canto $(0; 7,5; 2)$ devido à não existencia de uma porta simétrica à quina $(5; 5; 1,5)$ que absorva a energia.

6.2.4
Cálculo da potência para uma distribução de antenas SIMO

No ambiente da figura (6.12) o qual é constituído por paredes de concreto e portas de madeira, apresenta-se a distribuição das antenas que pertencem a um sistema SIMO (Simple Input Multiple Output). A posição (x, y, z) das antenas receptoras corresponde aos pontos $Rx1(7,5; 5,0; 1,0)$, $Rx2(7,5; 3,5; 1,8)$, $Rx3(5,5; 13,0,8)$, $Rx4(14,0; 14,0; 2,0)$ e a antena transmissora está localizada em $Tx(7,5; 7,5; 1,5)$. Pode-se observar que a altura das antenas é diferente para cada uma delas, de forma a evidenciar a aplicabilidade de uma análise $3D$.

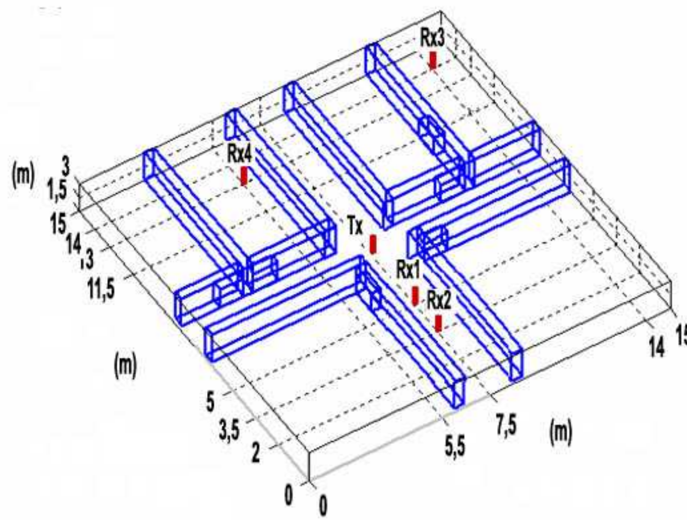

Figura 6.12: Distribução das antenas no ambiente considerado.

PUC-Rio - Certificação Digital Nº 0310507/CB

As figuras (6.13-6.16) apresentam os diagramas de potência, normalizada em relação ao máximo, recebida nos diversos receptores, à medida que o tempo evolui; estes gráficos são conhecidos também como perfis de retardo (power delay profile) [62].

Do cenário na figura (6.12), observa-se que existem antenas em situação de linha de visada direta (LOS) do transmissor, como as antenas $Rx1$ e $Rx2$, e outras não (NLOS), como as antenas $Rx3$ e $Rx4$. No caso das primeiras, elas recebem um sinal forte no primeiro instante de tempo, aos $22,5ns$ e $35,0ns$, respectivamente. O nivel do sinal chegando em Rx1(figura 6.13) apresenta sua primeira queda decorridos, aproximadamente $24,0ns$, devido à maior transmissão de energia através da porta nas imediações, energia que chegaria diretamente a $Rx1$ se aquela não existisse. Contribuições construtivas provenientes das difrações nas quinas e reflexões no corredor geram um aumento na energia recebida. A seguir, o nível do sinal vai decaindo devido a transmissões através de outras portas existentes no ambiente.

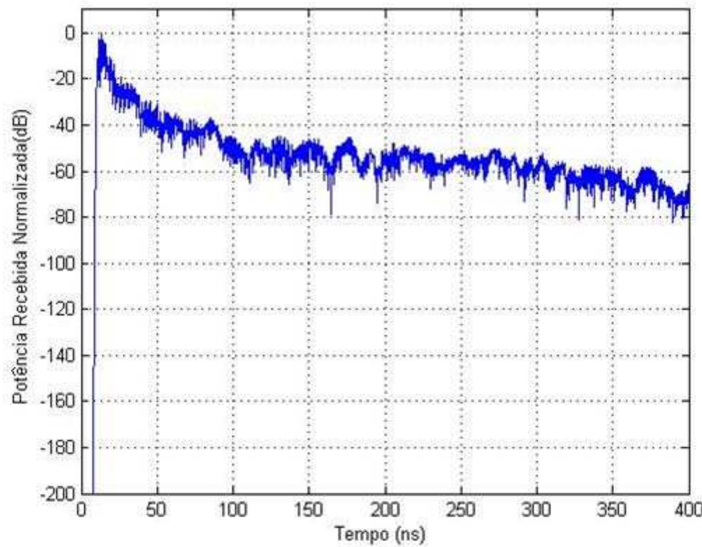

Figura 6.13: Potência recebida normalizada em Rx1.

PUC-Rio - Certificação Digital Nº 0310507/CB

Os resultados nas figuras (6.13) e (6.14) apresentam semelhança devido à proximidade e localização das antenas $Rx1$ e $Rx2$, separadas de apenas $1,5m$ com relação ao eixo y, ocupando mesma posição com relação ao eixo x e com uma diferença pequena $(0,8m)$ com relação ao eixo z.

O sinal chegando em $Rx3$ (figura (6.15) apresenta quedas menores devido às contribuições principais serem provenientes das transmissões e reflexões do sinal no corredor.

O sinal chegando em Rx4 (figura (6.16), devido a sua posição dentro do cenário (NLOS), apresenta um comportamento semelhante ao da antena $Rx3$, com a diferença que, aos $97ns$, apresenta uma queda significativa devido à parte da energia que é transmitida pela porta mais próxima à antena $Rx4$ antes de atingi-la.

Na figura (6.17) se compara os diagramas de potência recebida pela antena $Rx1$ e pela antena $Rx4$, ambos normalizados com relação à intensidade máxima em $Rx1$. É de se esperar que a intensidade do sinal inicial para antenas em situação NLOS seja menor se comparado com a intensidade do sinal para antenas em situação LOS, comportamento que é verificado na figura (6.17).

6.3 Modelagem de redes sem fio em ambientes interiores [56]

Na evolução de sistemas celulares, observamos que, para os sistemas de primeira geração, foi suficiente, na modelagem do canal de banda estreita, calcular a atenuação entre o transmissor e receptor. Para os sistemas de segunda geração, a descrição da seletividade em freqüência ou seu correspondente no domínio do tempo (a dispersão dos retardos) torna-se necessária, dando uma

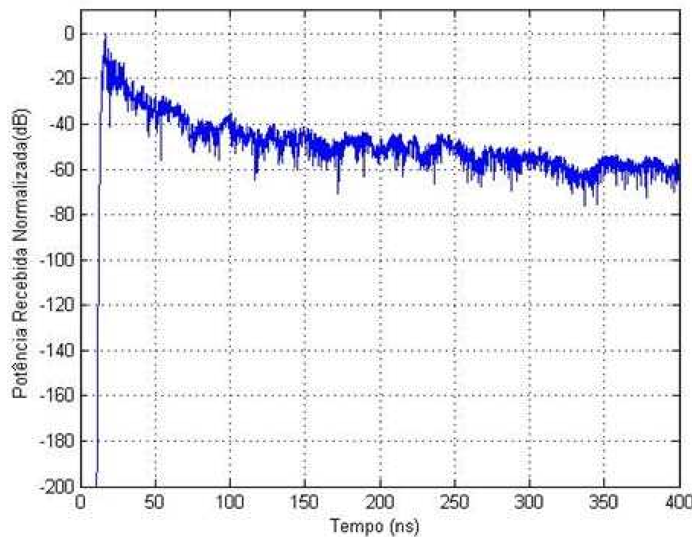

Figura 6.14: Potência recebida normalizada em Rx2.

PUC-Rio - Certificação Digital Nº 0310507/CB

descrição da variação da atenuação sobre a banda do sistema. Ainda, para sistemas de terceira geração recentes, são utilizadas múltiplas antenas, como no sistema SIMO (Simple Input Multiple Ouput).

Neste trabalho [56], trabalha-se com um sistema SIMO dentro de um ambiente interno, se faz uma análise comparativa da aplicabilidade dos métodos deterministicos RT/UTD e FDTD para o cálculo da cobertura. Para tal, será calculado o "Perfil de Retardo de Potência"(PDP), dado que este valor é usado para calcular outros parâmetros relacionados com a caracterização do canal.

6.3.1 Modelo do cenário

O ambiente considerado é o mesmo apresentado na figura (6.12) e ás paredes e portas podem ser caracterizados pelos parâmetros característicos da tabela (6.1.) Os valores da impedância intrínseca dos diferentes meios podem ser calculados por $\eta_i = \eta_0 \ / \ \sqrt{\epsilon_{efi}}$, onde $\eta_0 = 120\ \pi$ corresponde á impedância do espaço livre, $\epsilon_{efi} = (\epsilon_i - j\sigma_i \ / \ \omega)/\epsilon_0$ com ϵ_i, σ_i representando, respectivamente, sua permissividade elétrica e conductividade e ω é a freqüência angular.

6.3.2 Modelo da fonte

É utilizada uma antena dipolo, modelada para o método FDTD como descrito na seção (6.2.2), com a excitação no gap sendo dada por:

$$Ez(t) = E_0\ exp\left[-\left(2\pi f_b(t-t_0)\right)^2\right]\ cos[2\pi f_0(t-t_0)] \tag{6-3}$$

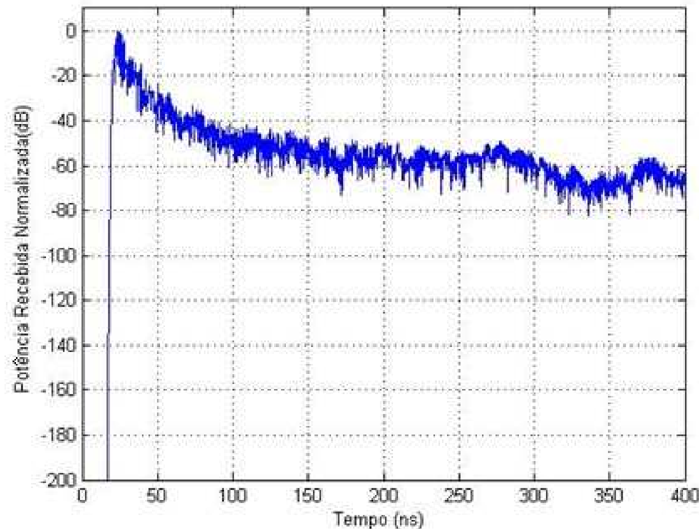

Figura 6.15: Potência recebida normalizada em Rx3.

PUC-Rio - Certificação Digital Nº 0310507/CB

onde $E_0 = 30V/m$ é a amplitude do sinal, $t_0 = 15ns$ é o tempo no qual o pulso atinge seu valor máximo, $f_b = 400MHz$ é a largura do pulso, $f_0 = 2,4GHz$ é a frequencia central e $0 \leq t \leq 6000\Delta t$ representa o tempo de propagação do sinal, com $\Delta t = 0,05ns$ representando o valor do passo no tempo. Observe-se que o sinal modelado desta forma permite escolher entre trabalhar em banda estreita ou banda larga, já que um pulso estreito no domínio do tempo proporciona um amplo conteúdo espectral no dominio da freqüência. A função (6-3) permite também a escolha de uma freqüência central (figura 6.18)

6.3.3
Simulações e resultados

A configuração do cenário (figura 6.12) corresponde a paredes de concreto e portas de madeira, todas elas fechadas e caracterizadas segundo os valores apresentados na tabela 6.1. A posição da antena transmissora corresponde ás coordenadas $(7,5; 7,5; 1,5)$ e a posição das antenas receptoras a $Rx1(7,5; 5,0; 1,0)$, $Rx2(7,5; 3,5; 1,8)$, $Rx3(5,5; 13,0; 0,8)$ e $Rx4(14,0; 14,0; 2,0)$.

A. **Cálculo do perfil de retardo de potência**

O PDP pode ser obtido através dos seguintes procedimentos, para o método RT/UTD: a) Calcular a potência recebida no receptor, usando os valores dos campos elétricos calculados pelas equações (4-10), (4-12), (4-46) e (4-60); a transformada de Fourier inversa da resposta em freqüência, fornece a resposta ao impulso limitada em banda do canal. b) ordenar cada raio recebido segundo seu tempo de chegada no receptor c)

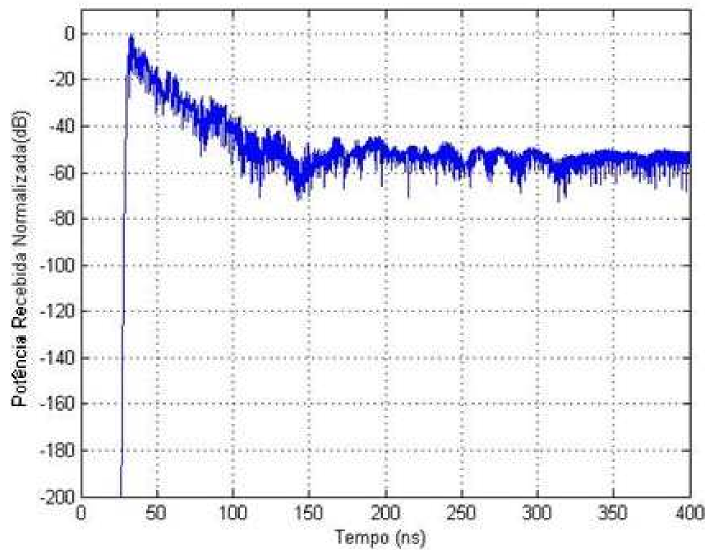

Figura 6.16: Potência recebida normalizada em Rx4.

PUC-Rio - Certificação Digital Nº 0310507/CB

normalizar os valores das potências com relação ao valor de potência máxima recebida. Para o FDTD, calcula-se os campos recebidos no receptor segundo as expressões (3-13)- (3-20).

As figuras (6.19-6.22) correspondem aos diagramas de potência, normalizadas com relação ao máximo recebido, nos diversos receptores, à medida que o tempo evolui. Quando se usa a transformada inversa de fourier, para obtenção da resposta no dominio do tempo segundo o método RT/UTD, são inseridos erros de aproximação, um fenômeno conhecido como alising, e para corrigi-eos é necessário usar uma transformada inversa de Fourier com "janelamento" [52] Observa-se nas figuras um alto grau de concordância entre os resultados dos dois métodos, o que confirma as informações a este respeito presentes na bibliografia [63]-[65].

Nas figuras (6.19) e (6.22) pode-se perceber que o maior contribução para o valor da potência do sinal vem do raio direto, comportamento esperado devido à situação LOS das duas antenas receptoras ($Rx1$ e $Rx2$). O primeiro sinal recebido em $Rx3$ e $Rx4$ é aproximadamente aos 5 e $6ns$, respetivamente, depois do primeiro sinal recebido em Rx1 e Rx2, devido à maior distancia que são percorridas pelos raios em face das condições NLOS em que se encontram estas antenas em relação à antena transmissora. Observam-s,e nos gráficos (6.21) e (6.22) desvanecimentos mais profundos que se comparados com os gráficos (6.19) e (6.20) pois o sinal sofrer várias transmissões até atingir os receptores, o que não acontece quando em situação LOS.

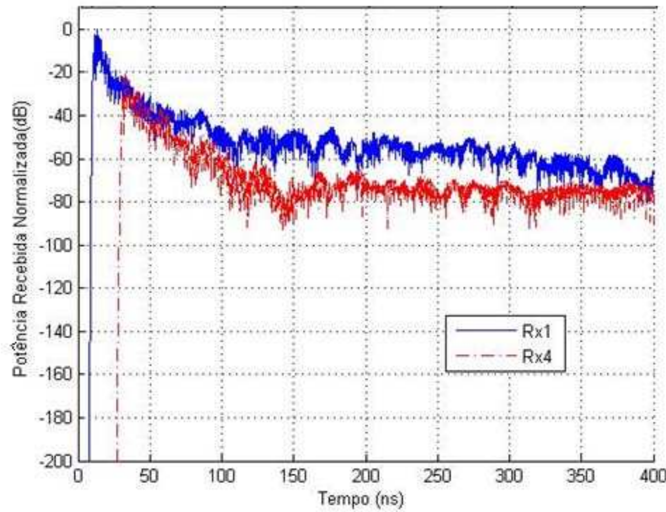

Figura 6.17: Comparação das potências recebidas em $Rx1$ e $Rx4$, normalizadas com relação ao máximo em Rx1.

B. **Transformação domínio do tempo - domínio da freqüência**

Os valores do campo elétrico no domínio do tempo obtidos em cada receptor, usando FDTD, são transformados para o domínio da freqüência a través da transformada rápida de Fourier [52], com a finalidade de comparar com os resultados obtidos via RT/UTD, conforme apresentado nas figuras (6.23-6.26). Observa-se uma melhor concordância dos resultados à medida em que a freqüência aumenta, comportamento condizente com a maior aplicabilidade do método para altas freqüência.

PUC-Rio - Certificação Digital Nº 0310507/CB

6.4
Resultados com inserção de mobiliário simples

Com a finalidade de ilustrar a adequação do método FDTD ao se levar em conta a presença de mobiliário na análise de cobertura de ambientes interiores, é inserido no interior de um dos ambientes uma mesa de madeira ($2x2x1$ m), representado na figura (6.27) por um bloco cinza.

As posições das antenas correspondem a: $Rx1(10,5;7,5;1,0)$, $Rx2(10,0;5,5;1,8)$, $Rx3(14,0;4,5;2,0)$; $Rx4(12,0;1,0;1,60)$ e a antena transmissora tem as mesmas características apresentadas para as simulações anteriores e descritas no capítulo 6. As figuras (6.28-6.31) ilustram os efeitos dos diferentes mecanismos de propagação através da comparação do sinal que atingem as antenas na presença e na ausência da mesa e da evolução do digrama de cobertura do ambiente.

Na figura (6.28.a) observa-se conforme esperado o sinal correspondente à situação sem a presença da mesa atinge o receptor antes daquele correspondentemente à situação que considera a presença da mesa. Já na figura (6.28.b),

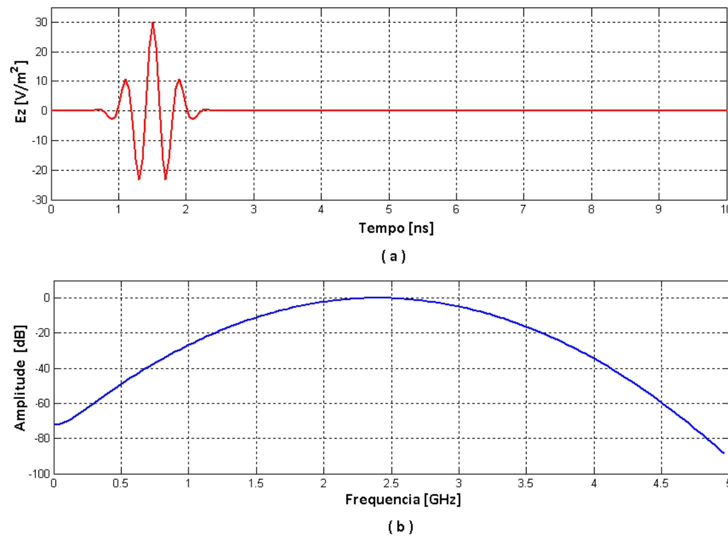

Figura 6.18: Pulso gausiano (6-3) para excitação de uma antena dipolo, (a) no dominio do tempo, (b) espectro de freqüência.

correspondente à cobertura do ambiente nos instantes iniciais, percebe-se a perturbação ainda mínima pela presença da mesa.

Na figura (6.29.a), com a antena receptora $Rx2$ se encontra próxima da porta do ambiente com a mesa a diferença entre os tempos de chegada do sinal nas duas situações é menor que para a antena $Rx1$. Observe-se, tanto na figura (6.29.a) como na figura (6.29.b), interferenças construtivas devido às reflexões e difrações, consequentemente incremento do nível do sinal. Observa-se também, na figura (6.29.b), a cobertura nos instantes iniciais em que o sinal interage com a mesa.

Na figura (6.30.a), devido á posição da antena receptora $Rx3$, observa-se que os tempos de chegada dos sinais, considerando a presença e a ausência da mesa, são quase iguais e as interferências construtivas são menos pronunciadas se comparadas com as da figura (6.29). Observam-se, também, desvanecimentos mais profundos na situação que considera a presença da mesa. A cobertura do ambiente corresponde à interação do sinal com toda a mesa de madeira.

Na figura (6.31.a), devido à posição da antena receptora $Rx4$, observa-se uma maior influencia dos mecanismos de propagação e consequentemente, as interfêrencias são mais pronunciadas comparadas com as situações anteriores. Da mesma forma que na figura (6.30.a), observa-se desvanecimentos mais profundos, como consequência da presença da mesa.

PUC-Rio - Certificação Digital Nº 0310507/CB

PUC-Rio - Certificação Digital Nº 0310507/CB

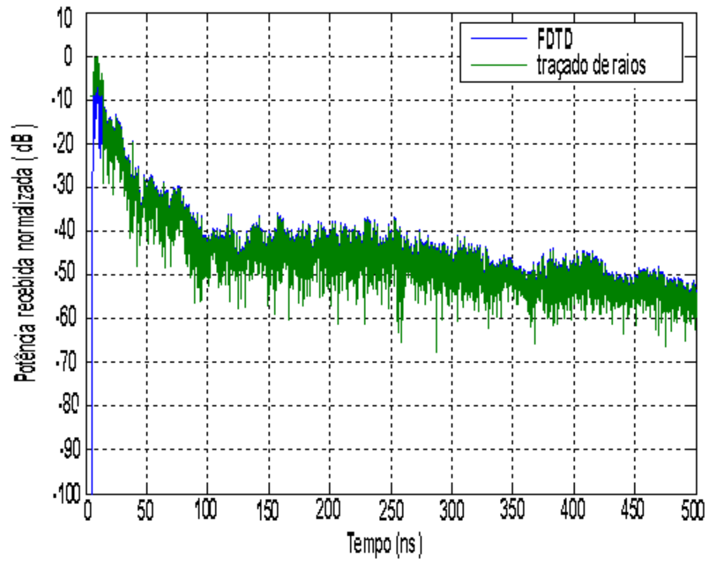

Figura 6.19: Comparação de potência recebida em Rx1, normalizada com relação ao máximo, calculada por RT e FDTD.

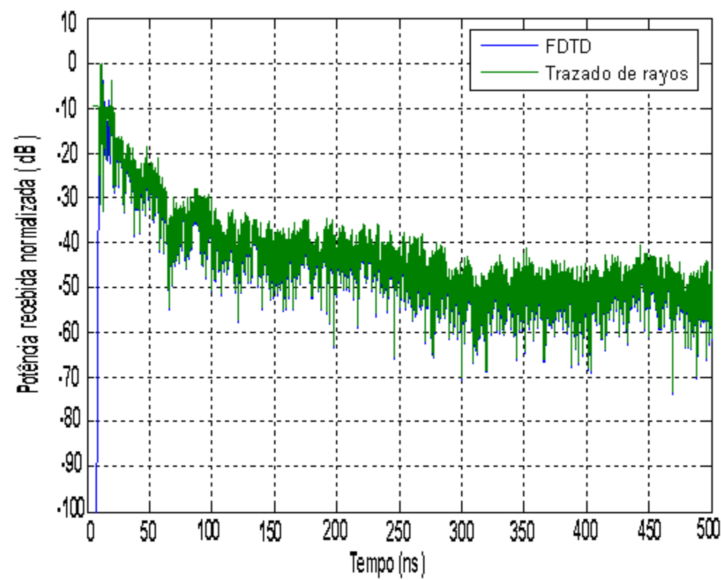

Figura 6.20: Comparação de potência recebida em Rx2, normalizada com relação ao máximo, calculada por RT e FDTD.

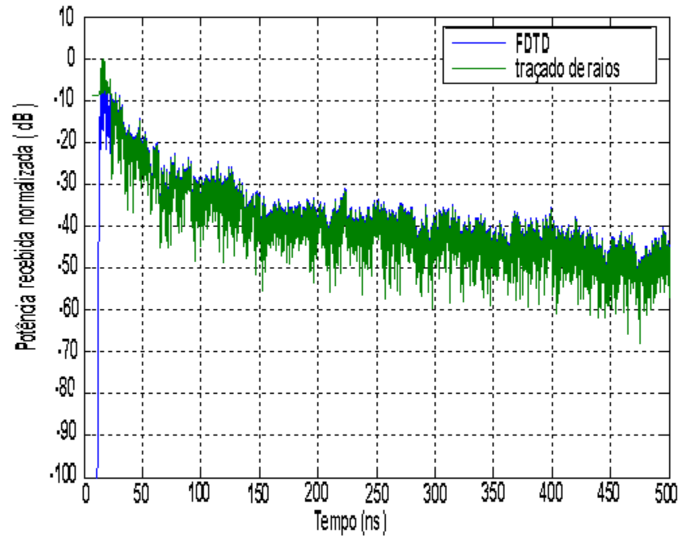

Figura 6.21: Comparação de potência recebida em Rx3, normalizada com relação ao máximo, calculada por RT e FDTD.

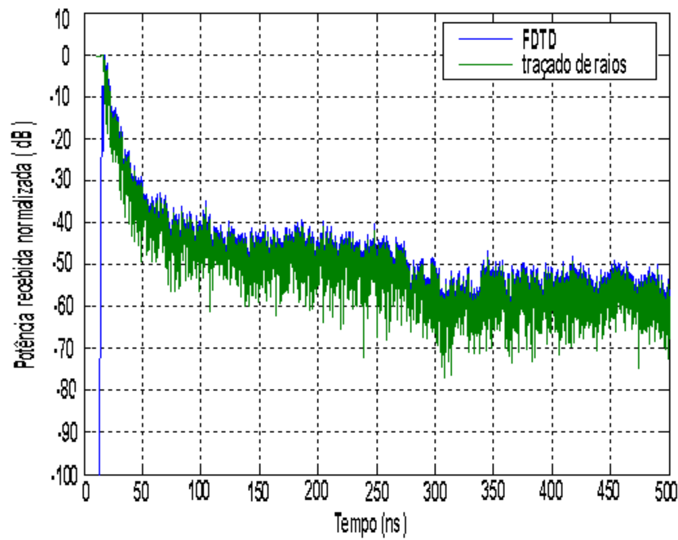

Figura 6.22: Comparação de potência recebida em Rx4, normalizada com relação ao máximo, calculada por RT e FDTD.

PUC-Rio - Certificação Digital Nº 0310507/CB

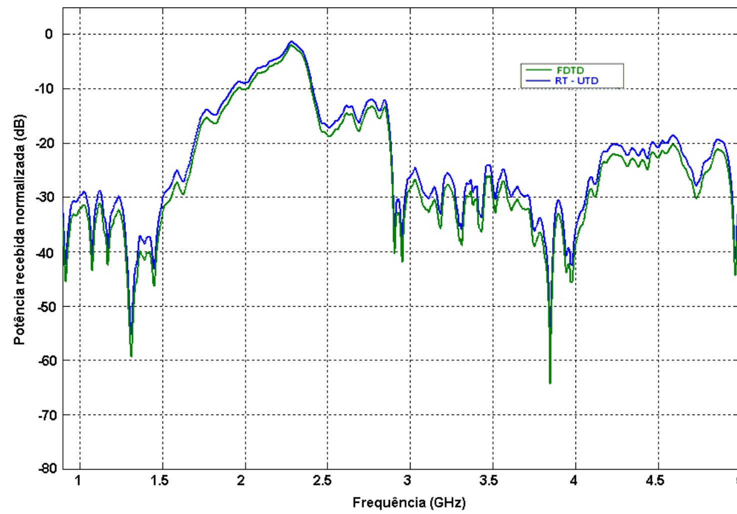

Figura 6.23: Comparação da variação com a freqüência da potência recebida em $Rx1$ e calculada usando RT/UTD e FDTD.

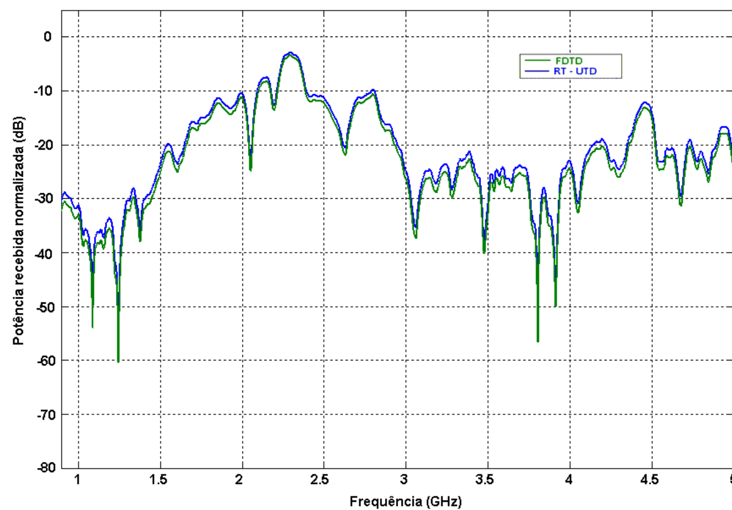

Figura 6.24: Comparação da variação com a freqüência da potência recebida em $Rx2$ e calculada usando RT/UTD e FDTD.

PUC-Rio - Certificação Digital Nº 0310507/CB

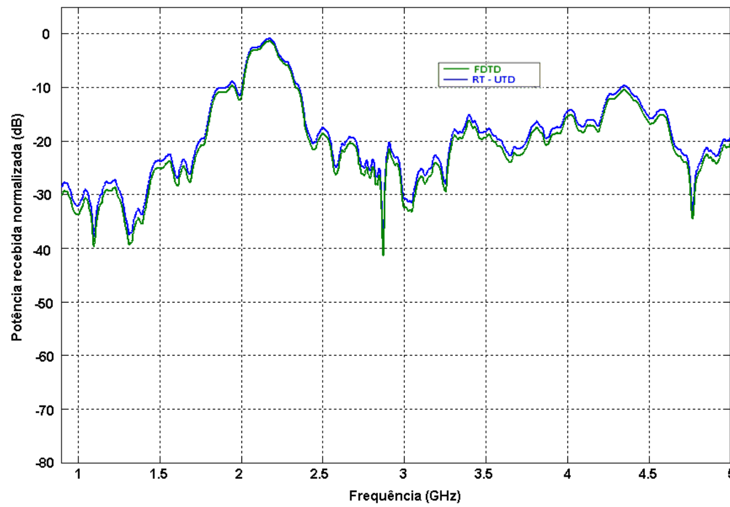

Figura 6.25: Comparação da variação com a freqüência da potência recebida em $Rx3$ e calculada usando RT/UTD e FDTD.

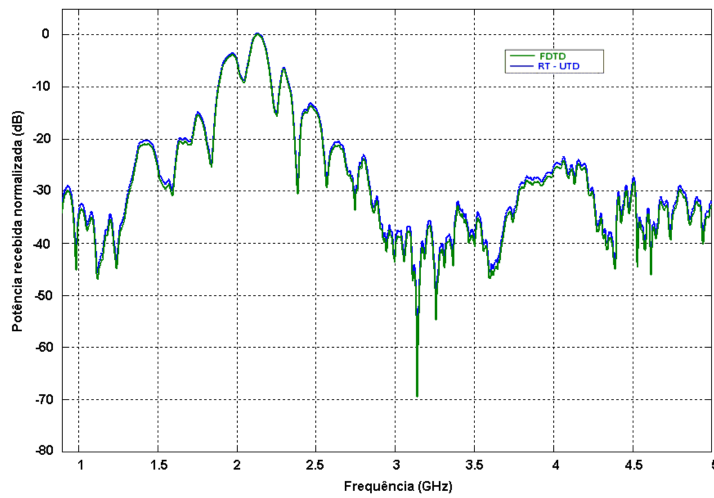

Figura 6.26: Comparação da variação com a freqüência da potência recebida em $Rx4$ e calculada usando RT/UTD e FDTD.

PUC-Rio - Certificação Digital Nº 0310507/CB

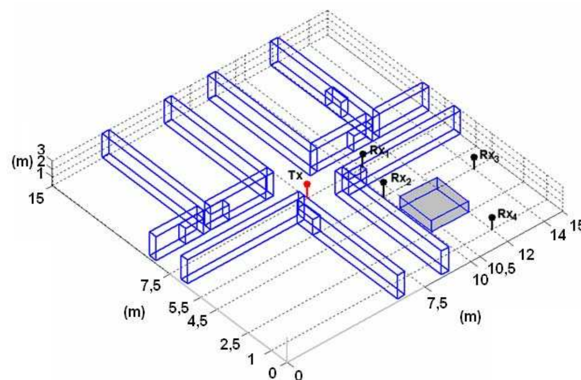

Figura 6.27: Cenário com uma distribução de antenas e um bloco de madeira de $(2x2x1\ m)$ no interior

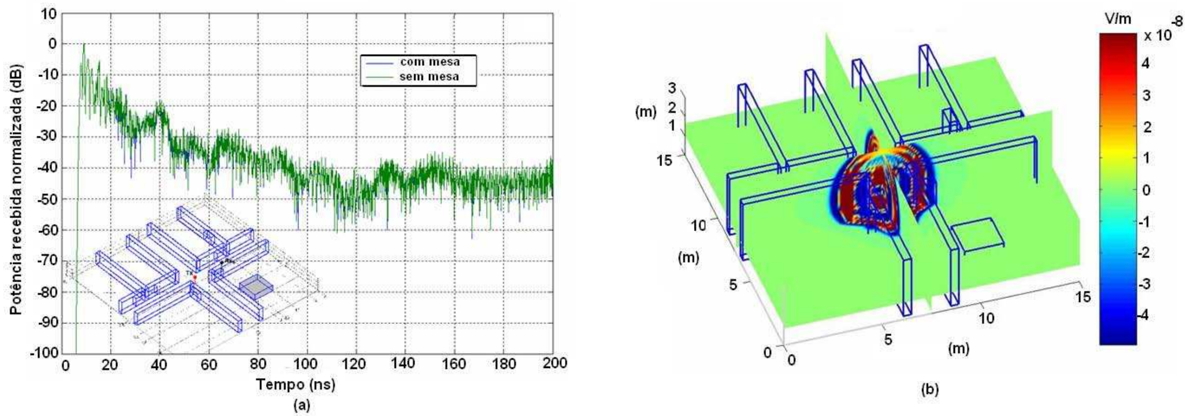

Figura 6.28: (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx1$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $6,5$ ns.

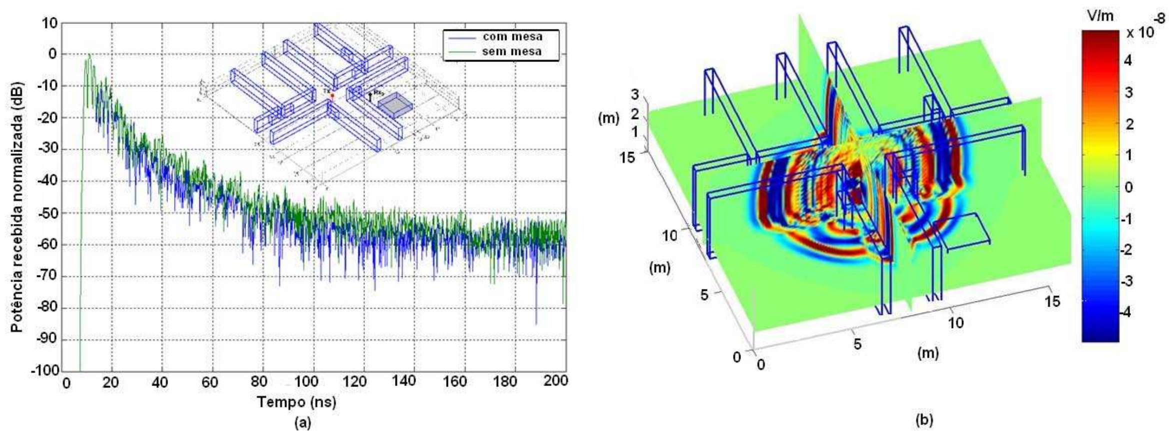

Figura 6.29: (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx2$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $13,2$ ns.

PUC-Rio - Certificação Digital Nº 0310507/CB

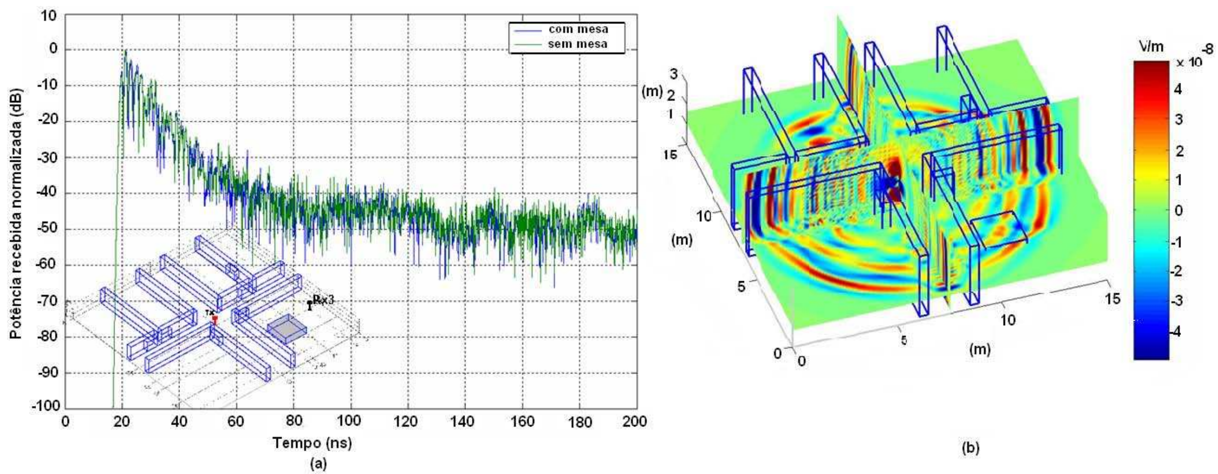

Figura 6.30: (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx3$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $19,5\ ns$.

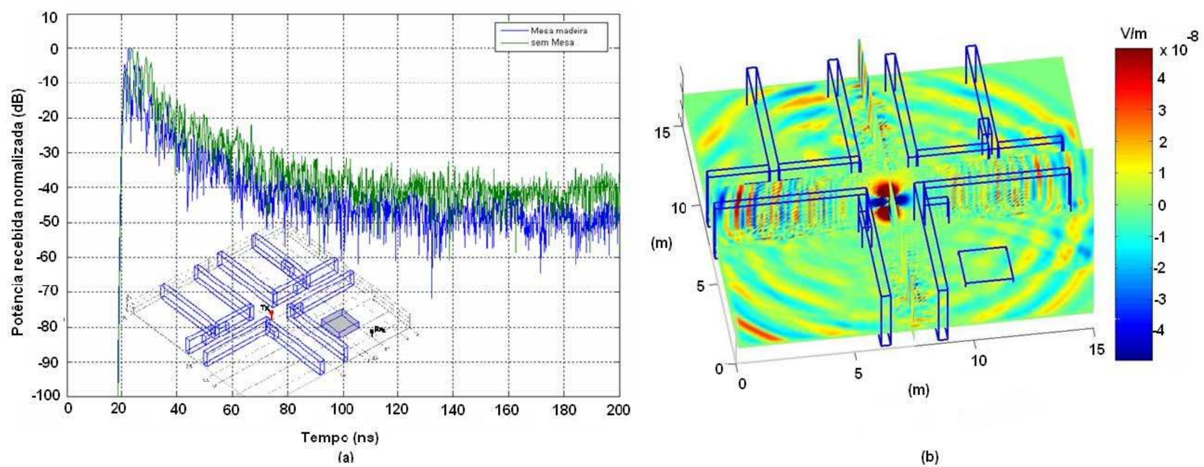

Figura 6.31: (a)comparação da potencia, normalizada com relação ao máximo, recebida na antena $Rx4$ com e sem a presença de uma mesa de madeira, (b) Cobertura pela componente de campo Ez do cenario da figura (6.27), constituido por paredes de concreto e portas de madeira, decorridos $26,2\ ns$.

PUC-Rio - Certificação Digital Nº 0310507/CB

7
Conclusoes e Sugestões

Este trabalho foi dedicado ao estudo do método das Diferenças Finitas no Domínio do Tempo (FDTD) para a análise da cobertura eletromagnética em ambientes interiores, aí considerada a existência de objetos de dimensões menores ou iguais ao comprimento de onda e de materiais e formas diversas. Estas características costumam tornar a o emprego de métodos analíticos, como o rastreamento do campo eletromagnético ao longo de raios óticos, de difícil implementação, recomendando a adoção de métodos numéricos, entre eles FDTD, conforme aqui investigado.

Inicialmente, no capitulo 2, foi apresentada uma revisão bibliográfica dos diferentes métodos, estatísticos e determinísticos, existentes para o cálculo de cobertura em ambientes interiores, ressaltando-se sua gama mais ou menos restrita de aplicações. Quanto a utilizações anteriores do método FDTD, estas se deram mais recentemente, e na maioria dos casos, para aplicações bidimensionais, salvo algumas versões simplificadas dos casos 3D [28, 29]. Esta limitação, conforme explicitado na seção 3.3, está relacionada diretamente a critérios de estabilidade do método, sendo indicada a representação, ao longo de um comprimento de onda, por pelo menos 10 incrementos espaciais dentro do esquema FDTD. Foram também apreciadas as referências bibliográficas que introduzem métodos híbridos, acomodando o traçado de raios aliado ao cálculo do campo espalhado pela Teoria Uniforme da Difração (RT/UTD) em ambientes vazios, como corredores, à implementação do FDTD em ambientes contíguos, tipicamente mobiliados, embora as versões existentes relatadas tenham se restringido a configurações bidimensionais [4, 5, 9,10]. Optou-se, no presente trabalho, por explorar, em situações mais próximas das realistas, ou seja, abordando configurações tridimensionais, a aplicabilidade do método FDTD como ferramenta única de cálculo de cobertura em ambientes interiores, o que poderá subsidiar, inclusive sua eventual utilização em métodos híbridos para cenários complexos.

No capitulo 3 procedeu-se a uma exaustiva revisão do método FDTD e, na seção 3.4, equacionou-se a quantidade de memória e tempo de CPU necessários para um dado problema especifico, usando como variáveis independentes a

PUC-Rio - Certificação Digital Nº 0310507/CB

velocidade de operação do computador e o número de operações de ponto flutuante. Foram também descritas técnicas complementares que permitem abordar problemas em espaços abertos, conhecidas como camadas absorventes, tendo sido implementadas as duas mais conhecidas, PML e UPML, conforme apresentadas nas seções 3.6 e 3.7. Concluiu-se que o FDTD consume uma quantidade significativa de memória e, do exemplo na seção 3.4.2, observou-se que, para um cenário típico de $6x15x3m^3$ e uma freqüência de trabalho de $1,2GHz$, a quantidade de memória RAM necessária par o cálculo da cobertura foi de $0.972GB$. Observou-se também, das implementações, que a escolha entre as duas camadas absorventes deve recair na UPML por utilizar menos memória. Ainda no capitulo 3, foram revistas as técnicas existentes para gerar fontes de excitação, passando por pulsos gaussiano, duplo gaussiano, gaussiano diferencial, acopladas à incidência de ondas planas em obstáculos canônicos.

No capitulo 4 foi apresentada uma revisão do método assintótico para altas freqüências, aqui denominado RT/UTD, ressaltadas suas características e propriedades principais e listadas as expressões utilizadas em sua implementação nos exemplos apresentados no capítulo 6. As duas versões (bidimensionais) existentes do método híbrido RT/UTD - FDTD, a primeira de Wang [5] e a segunda de Reynaud [4], são resumidas no capítulo 5, evidenciando as principais características de cada implementação.

No capitulo 6 foram apresentados resultados obtidos nas diferentes fases de implementação, tanto do núcleo FDTD como de outras técnicas necessárias para a adoção de um método hibrido 3D, testados em casos canônicos ou comparados com resultados apresentados na literatura. Assim, a implantação do programa FDTD-PML bidimensional desenvolvido foi validada com os resultados presentes em [67, 38] e, conforme verificado na seção 6.1, pode-se concluir que a versão PML das camadas absorventes é altamente efetiva permitindo simular a propagação em espaços abertos, além de permitir simular com facilidade uma distribuição de múltiplas antenas. No caso específico de um sistema SIMO, também foi observado que a quantidade de memória requerida pela PML foi excessiva, mesmo na simulação bidimensional. Tendo isso em conta e com o objetivo de analisar configurações tridimensionais, implementou-se a UPML que requer menos memória e tem a mesma eficácia, conforme evidenciado pelos resultados apresentados na seção 3.7 para a aplicação do método FDTD tridimensional no cálculo de cobertura de um sistema SIMO, onde as antenas de diferentes alturas foram distribuídas em zonas singulares, de forma a permitir a identificação dos diversos mecanismos de propagação, observando-se um comportamento similar aos resultados publicados em [62, 25], que foram obtidos através de medidas em ambientes semelhantes. A melhor adequação

PUC-Rio - Certificação Digital Nº 0310507/CB

de uma modelagem 3D foi ainda constatada, na seção 6.4, ao se inserir no ambiente uma mesa de madeira e executar a simulação nas mesmas condições da seção 6.3. Na seção 6.3.3.B é apresentada uma análise comparativa do cálculo de cobertura em ambientes interiores usando os métodos RT/UTD e FDTD, com a transposição entre os domínios temporal e da freqüência sendo implementada através de inversões de Fourier. A concordância sempre observada entre os resultados das coberturas, para ambientes com diferentes materiais, preenchidos ou não, permitiu concluir pela adequação - e complementaridade - dos métodos, em especial do FDTD, objeto deste trabalho.

7.1 Sugestões para trabalhos futuros

Em desenvolvimentos futuros, sugere-se a extensão e implementação do modelo híbrido para ambientes tridimensionais complexos e, em especial, em uma abordagem da configuração de transmissão outdoor e recepção indoor, de interesse em sistemas ora em implantação, como os de televisão digital. Ainda, a utilização de técnicas assintóticas no domínio do tempo como, por exemplo, discutidas recentemente em [69], em esquemas híbridos (FDTD - RT/TD-UTD), otimizaria conversões entre os domínios do tempo e da freqüência e seria de grande aplicação na análise de sistemas de banda ultra larga (UWB). Sempre que possível, nas diversas situações mencionadas, deve-se procurar consubstanciar as implementações através de uma comparação de seus resultados com aqueles provenientes de campanhas de medidas.

PUC-Rio - Certificação Digital Nº 0310507/CB

Referências Bibliográficas

[1] OHMORI, S.; YAMAO, Y. and NAKAJIMA, N., "The future generations of mobile communications based on broadband access technologies",*IEEE Communications Magazine*, pp. 134–142, December 2000.

[2] RODRIGUES, M.E.C., "Técnicas de traçado de raios em três dimensões para cálculo de campos em ambientes interiores e exteriores". *Dissertação de Mestrado-Pontificia Universidade Católica do Rio de Janeiro*, Abril 2000.

[3] RAMIREZ, L.A.R., "Técnica de lançamento de raios em trés dimensões para a previsão da cobertura em ambientes micro-celulares".*Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro*, Novembro 2002.

[4] REYNAUD, S., "Modélisation hybride du canal radiomobile en environnement indoor complexe. application aux systémes sans fil". *Thése Docteur De L'université De Limoges, Université de Limoges*, 2006.

[5] WANG, Y.; CHAUDHURI, S. and SAFAVI-NAEINI, S., "An FDTD/RAY-TRACING analysis method for wave penetration through inhomogeneous walls",*IEEE Transactions on Antennas and Propagation, vol. 50*, pp.1598–1604, November 2002.

[6] KEPAL, M.; PESCH, D. and HRADECKY, Z., "Optimising motif models for indoor radio propagation prediction using evolutionary computation",*Irish Signal and System Conference - ISSC 2002, Ireland, Cork*, pp.345–351, June 2002.

[7] WÖLFLE, G.; WAHL, R.; WILDBOLZ, P. and WERTZ, P., "Dominant path prediction model for indoor and urban scenarios",*11th COST 273 MCM, Duisburg (Germany)*, pp.176–179, September 2004.

[8] REYNAUD, S.; VAUZELLE, R.; REINEIX, A. and GUIFFAUT, C., "A hybrid FDTD/UTD radiowave propagation modeling: application to

PUC-Rio - Certificação Digital Nº 0310507/CB

indoor channel simulations",*Microwave And Optical Technology Letters, vol. 49, no. 6*, pp.1312–1320, June 2007.

[9] YUN, Z.; ISKANDER, M.F.; ZHANG Z. and SORENSEN, R.K., "Outdoor/indoor propagation prediction for complex wall and window structures using ray-tracing models", *Vehicular Technology Conference, IEEE 58th*, pp.154–157, 2003.

[10] REYNAUD, S.; GUIFFAUT, C.; REINEIX, A. and VAUZELLE, R., "Modeling indoor propagation using an indirect hybrid method combining the utd and the fdtd methods",*European Conference on Wireless Technology*, pp.345–348, 2004.

[11] European Comisson, "Digital mobile radio towards future generation systems",*Cost 231 Final Report 1 ed. Bruselas, cap. 4*, pp.223–237, 1999.

[12] KEENAN, J.M. and MOTLEY, A.J., "Radio coverage in buildings",*British Telecom Technology Journal, Vol.8, no. 1*, pp.19–24, 1990.

[13] RAPPAPORT, T.S. and SEIDEL, S.Y., "914 MHz path loss prediction models for indoor wireless communications in buildings".*IEEE Transactions on Antennas and Propagation, vol. 40, no.2*, pp.207–217, February 1992.

[14] SALEH, A.M. and VALENZUELA, R.A., "A statistical. model for indoor multipath propagation",*IEEE Journal on Selected Areas in Communications vol. 5 no. 2*, pp.128–137, 1987.

[15] LOTT, M. and FORKEL, I., "A multi-wall-and-floor model for indoor radio propagation",*Proceedings of VTC 2001 - Vehicular Technology Spring Conference*, May 2001.

[16] Recomendação UIT-R P. 1238.

[17] SANDOR, Z.; NAGY, L.; SZABO, Z. and CSABA, T. "3D ray launching and moment method for indoor radio propagation purposes", *IEEE international symposium on personal indoor and mobile radio communications Nro. 8, Helsinki , Finlande*, pp.130–134, Sep 1997.

[18] THEOFILOGIANNAKOS, G. K.; YIOULTSIS, T. V. and XENOS, T. D. "Integral equation method for the analysis of wave propagation in highly complex indoor communication environments", *Wireless Personal*

PUC-Rio - Certificação Digital Nº 0310507/CB

Communications: An International Journal vol. 43, no. 2, pp.495–510, February 2007.

[19] DE BACKER, B.; OLYSLAGER, F. and DE ZUTTER, D., "An integral equation approach to the prediction of indoor wave propagation",*Radio Science, vol. 32, no. 5*, pp.495–510, 1997.

[20] TALBI, L.; DELISLE, G.Y. and AHERN, J.,"Characterization of wireless indoor communication channel: measurements and simulation",*Gateway to the 21st Century. Conference Record., 2nd International Conference on Personal Communications, vol. 2*, pp.616–620, October 1993.

[21] WANG.Z. and ZHANG, Y., "Indoor field strength prediction based on FMM",*The Journal of China Universities of Posts and Telecommunications, vol. 13, no. 3*, pp.616–620, September 2006.

[22] LU, C.C., "Indoor radio-wave propagation modeling by multilevel fast multipole algorithm",*Microwave and Optical Technology Letters, vol. 29, no. 3*, pp.168–175, March 2001.

[23] HO, C.M.P. and RAPPAPORT, T.S., "Wireless channel prediction in a modern office building using an image-based ray tracing method",*Proceeding of IEEE GLOBECOM'93. Houston, Texas, vol. 2*, pp.1247–1251, November 1993.

[24] ANGELET, F.A.; HERNANDO-RÁBANOS, J.M.; DE VICENT, F.I. and FONTAN, F.P.,"Efficient ray-tracing acceleration techniques for radio propagation modeling",*IEEE Transaction on Vehicular Technology, vol. 49, no. 6*, pp.2089–2104, November 2000.

[25] LAUER, A.; BAHR, A. and WOLF, I., "FDTD simulations of indoor propagation",*IEEE Vehicular Technology Conference Proceedings*, pp.883–886, 1994.

[26] WANG, Y.; SAFAVI-NAEINI, S. and CHAUDHURI, S.K., "Comparative study of lossy dielectric wedge diffraction for radio wave propagation modeling using UTD and FDTD",*IEEE Antennas and Propagation Society International Symposium, Orlando, FL, vol. 4*, pp.2826–2829, July 1999.

[27] SCHUSTER, J. and LUEBBERS, R., "FDTD techniques for evaluating the accuracy of ray-tracing propagation models for microcells",*presented*

PUC-Rio - Certificação Digital Nº 0310507/CB

at the Proc. 16th Annual Review of Progress in Applied Computational Electromagnetics at the Naval Postgraduate School, Monterey, CA, pp.20–25, March 2000.

[28] KONDYLIS, G.; FLAVIIS, F. D.; POTTIE, G. and RAHMAT-SAMII, Y., "Indoor channel characterization for wireless communications using reduced finite diference time domain (R-FDTD)",*presented at International Conference on Electromagnetics in AdvancedApplications (ICEAA 99), Torino, Italy*, pp.19–22, September 1999.

[29] HOLLOWAY, C. L.; COTTON, M. G. and MCKENNA, P., "A simplified model for calculating the decay rate of the impulse response for an indoor propagation channel",*Wireless Communications Conference, Boulder, CO.*, pp.210–214, August 1997.

[30] SCHÄFER, T. M. and WIESBECK, W. "Simulation of radiowave propagation in hospitals based on FDTD and ray-optical methods",*IEEE Transactions on Antennas and Propagation, vol. 53, no. 8*, pp.2381–2388, August 2005.

[31] JI, Z.; SARKAR, T. K. and LI, B., "Analysis of the effects of walls on indoor wavepropagation using the FDTD method", *Microwave and Optical Technology Letters, vol. 29, no. 1*, pp.19–21, April 2001.

[32] ZHAO, Y.; HAO, Y. and PARINI, C., "Two novel FDTD based UWB indoor propagation models", *IEEE International Conference on Ultra-Wideband, Zurich*, pp.124–129, September 2005.

[33] WALLACE, J.W. and JENSEN, M.A., "Validation of parameteric directional MIMO channel models from wideband FDTD simulations of a simple indoor environment", *IEEE Antennas and Propagation Society International Symposium, vol. 2*, June 2003.

[34] WANG, Y.; SAFAVI-NAEINI, S. and CHOUDHURI, S.K.,"A hybrid technique based on combining RAY TRACING and FDTD methods for site specific modeling of indoor radio wave propagation",*IEEE Transaction on Antennas and Propagation, vol. 48*, pp.743–754, May 2000.

[35] P. BERNARDI, M. CAVAGNARO, P. D'ATANASIO, E. DI PALMA, S. PISA and E. PIUZZI, "FDTD, multiple-region/FDTD, RAY-TRACING/FDTD: a comparison on their applicability for human, exposure evaluation", *International Journal of Numerical Modelling: Elec-*

PUC-Rio - Certificação Digital Nº 0310507/CB

tronic Networks, Devices and Fields, vol.15 no.5,6, pp.579–593, 2002.

[36] SADIKU, M.N.O., "Numerical techniques in electromagnetics",*2nd Ed. CRC Press*, 2000.

[37] TAFLOVE, A. and HAGNESS, A., "Computational electrodynamics: the finite-difference time-domain method",*Third Ed. Artech House*, 2005.

[38] YEE, K.S., "Numerical solution of initial boundary values problems involving Maxwell's equations in isotropic media".*IEEE Transaction on Antennas and Propagation, vol. 4, no 3*, pp.302–307, 1966.

[39] SHLAGER, K. L. and SCHNEIDER, J. B., "A selective survey of the finite-difference time-domain literature", *IEEE Antennas and Propagation Magazine, vol. 37, no. 4*, pp.39–56, 1995.

[40] MUR, G. "Absorbing boundary conditions for finite diference approximation of the time domain electromagnetic-field equation".*IEEE Transaction on Electromagnetic Compatibility, vol. 23, no. 4*, pp.377–382, November 1981.

[41] KATZ, D. S. and TAFLOVE, A., "Validation and extension to three dimensions of the Berenger PML absorbing boundary condition for FD-TD Meshes",*IEEE Microwave and Guided Wave Letters, vol.4.No.8*, pp.268–270, August 1994.

[42] BERENGER, J.P. "Perfectly matched layer for the FDTD Solutions of Wave-Structure Interaction Problems", *Journal of Computational Physics, vol. 114 no.2*, pp.185–200, October 1994.

[43] GEDNEY, S.D., "An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD Lattices", *IEEE Transaction on Antennas and Propagation, vol. 44, no. 12*, pp.1630–1639, December 1996.

[44] RODEN, J.A. and GEDNEY, S.D., "Convolutional PML (CPML): an efficient FDTD implementation of the CFS-PML for arbitrary media", *Microwave and Optical Technology Letters, vol. 27, no. 5*,pp.334–339, June 2000.

PUC-Rio - Certificação Digital Nº 0310507/CB

[45] TARRICONE L. and ESPOSITO, A. "Grid Computing for Electromagnetics",*Artech House*, 2004.

[46] TAFLOVE, A. and BRODWIN, M.E., "Numerical solution of steady-state electromagnetic scattering problems using the time dependent Maxwell equations".*IEEE Transaction on Microwave Theory and techniques, vol.23, no. 8*, pp.623–630, August 1975.

[47] HOLLAND, R. and WILLAMS, J., "Total Field versus Scattered-Field Finite-Difference Codes: A Comparative Assessment",*IEEE Transaction on Nuclear Science, vol. 30, no. 6*, pp.4583–4588, December 1983.

[48] FURSE, C.M.; ROPER, D.H.; BUECHLER, D.N.; CHRISTENSEN, D.A. and DURNEY, C.H., "The problem and tratament of DC offsets in FDTD simulations",*IEEE Transactions on Antennas and Propagations, vol. 48, no. 8*, pp.1198–1201, August 2000.

[49] GARCÍA, S.G.; BRETONES, A.R.; OLMEDO, B.G. and MARTÍN, R.G., "Finite difference time domain methods", *Departament of Electromagnetism and Matter Physics University of Granada (Spain).*

[50] SULLIVAN, D.M., "An unsplited step 3-D PML for use with the FDTD method",*IEEE Microwave and Guided Wave Letters, vol. 7*,pp.184–186, July 1997.

[51] LUEBBERS, R.J., "A heuristic UTD slope diffraction coefficient for rough lossy wedges",*IEEE Transaction on Antennas and Propropagation, vol. 37, no. 2*, pp.206–211, February 1989.

[52] CHEN, V. C. and LING, H. "Time-Frequency transforms for radar imaging and signal analysis", *Artech House*, 2002.

[53] TALBI, L. and DELISLE, G. Y., "Finite diference time domain characterization of indoor radio propagation",*Progress In Electromagnetics Research, PIER 12*, 1996.

[54] RAMIREZ, L. A. R.; SILVA MELLO, L. A. R. y HASSELMANN, F. J. V.,"Cálculo de la cobertura en una WLAN usando diferencias finitas en el dominio del tiempo". *Anales del III Congreso Iberoamericano De Estudiantes De Ingeniería Eléctrica.* Mérida, pp.158–163, Mayo 2008.

[55] RAMIREZ, L. A. R.; SILVA MELLO, L. A. R. e HASSELMANN, F. J. V., "Emprego do método FDTD para análise de cobertura em ambientes

PUC-Rio - Certificação Digital Nº 0310507/CB

interiores" *Anais do MOMAG 2008 (13º Simpósio Brasileiro de Micro-ondas e Optoeletrônica e 8º Congresso Brasileiro de Eletromagnetismo, Florianópolis*, pp.138–141, Setembro 2008.

[56] RAMIREZ, L. A. R.; SILVA MELLO, L. A. R. y HASSELMANN, F. J. V., "Modelamiento de redes inalámbricas usando dos métodos numéricos", *V Congreso Internacional de Matemática Aplicada, Chiclayo*, Noviembre 2008.

[57] ZHANG, J. T. and HUANG, Y., "Indoor channel characteristics comparisons for the same building with different dielectric parameters",*IEEE Conference, vol. 2*,pp. 916–920, 2002.

[58] GLISIC, S., "Advanced wireless communications 4G technologies",*John Wiley and Sons*, 2004.

[59] ALI, F. and HORTON, J.B., "Introduction to the special issue on emerging commercial and consumer circuits, systems, and their applications",*IEEE Transaction on Microwave Theory and Technology, vol. 43*, pp. 1633–1637, 1995.

[60] MEINEL, H.H., "Commercial applications of millimeter waves history, present status, and future trends",*IEEE Transaction on Microwave Theory and Technology vol 43*, pp. 1639–1653, 1995.

[61] SWANSON, D. G.; WOLFGANG, JR. and HOEFER, J.R., "Microwave circuit modeling using electromagnetic field simulation", *Artech House*, 2003.

[62] RAPPAPORT, T.S., "Wireless communications: principles and practice", *Pretince Hall*, 1996.

[63] BERTONI, H.L. "Radio propagation for modern wireless system", *Artech House*, 2001.

[64] ANDERSON, H.R., "Fixed broadband wireless system design", *John Wiley and Sons*, 2003.

[65] ALLEN, B.; DOHLER, M.; OKON, E.; MALIK, W.; BROW A. and Edwards, D., "Ultra wideband antennas and propagation for communications, radar and imaging", *John Wiley and Sons*, 2007.

[66] FOLCH, J.R.; PEREZ, J.; PINEDA, M. and PUCHE, R., "Quadtree meshes applied to the finite element computation of phase inductances

PUC-Rio - Certificação Digital Nº 0310507/CB

in an induction machine", Intelling Computer Techniques in Applied Electromagnetics",*Springer Berlin, Vol. 119*, 2008.

[67] BELEM, A.N. "Caracterização bidimensional de canais de rádio através de diferenças finitas no domínio do tempo", *Dissertação de Mestrado - Universidade Federal de Minas Gerais, Escola de Engenharia Eletrônica*, 2001.

[68] FOLCH, J.R.; PEREZ, J.; PINEDA, M. and PUCHE, R., "Quadtree meshes applied to the finite element computation", *Intelligent Computer Techniques in Applied Electromagnetics*, Springer 2008.

[69] DO REGO, C.G., "Formulações assintóticas para o espalhamento por superfícies condutoras no domínio do tempo e aplicações à análise de transientes em antenas refletoras", *Tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro*, 2001.

[70] A.N. NOORI, A.A. SHISHEGAR, E. JEDARI, "A new double counting cancellation technique for three-dimensional ray launching method", *Digest of IEEE Antennas and Propagation Society International Symposium*, pp. 2185–2188, July 2006.

[71] CATEDRA, M.F. and PÉREZ-ARRIAGA, J., "Cell Planning for Wireless Communications",*Artech House - Mobile Communications Series*, 1999.

PUC-Rio - Certificação Digital Nº 0310507/CB

A
Aspectos do lançamento e captação de raios

A.1
Antena Receptora (Rx)

Para a determinação do campo na antena receptora, a partir do rastreamento de raios ao longo do ambiente é necessário usar uma técnica conhecida como o teste da esfera de recepção, na qual uma esfera é posicionada ao redor da antena Rx e um raio é considerado como recebido por esta antena se ele atinge a esfera. O raio da esfera de recepção é dado por [2],[70]:

$$R = \frac{\alpha L}{\sqrt{3}} \tag{A-1}$$

onde α representa a separação angular entre dois raios adjacentes, L a distancia total percorrida pelo raio desde o transmissor até o receptor, também conhecida como distancia "unfolded".

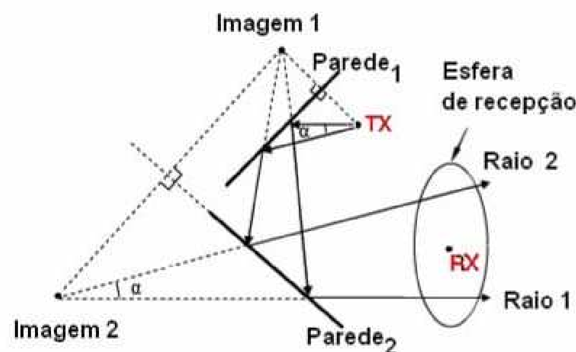

Figura A.1: Problema de dupla contagem na esfera de recepção [70]

O raio desta esfera deve ser dimensionado cuidadosamente de forma a evitar [3],[71] problemas de dupla recepção, este problema é ilustrado na figura (A.1). Uma possível solução a este problema em geometrias planares é o uso de esferas de recepção adaptativas (figura A.2). Para cada ponto de recepção é feita uma projeção perpendicular "d" do ponto de recepção ao segmento do raio. Usa-se a distancia "unfolded" L do ponto de transmissão a ponto de recepção. Uma região circular de raio $L/2$ é definida ao redor do ponto de recepção. Todos os raios que passem dentro desta região de recepção são

PUC-Rio - Certificação Digital Nº 0310507/CB

considerados como capazes de sensibilizar ao receptor. Logo, se $d < L/2$, o raio contribui para o sinal no receptor e se $d > L/2$, o raio não atinge ao receptor.

Quando-se considera o caso 3D e o esquema de lançamento é geralmente feito variando os ângulos azimutal e longitudinal com incrementos iguais, conforme figura (A.5). Desta forma, raios adjacente não apresentam a mesma separação angular e, para conseguir um lançamento de raios uniforme e garantir a recepção se utilizam, esferas geodésicas (seção A.2) no lançamento dos raios.

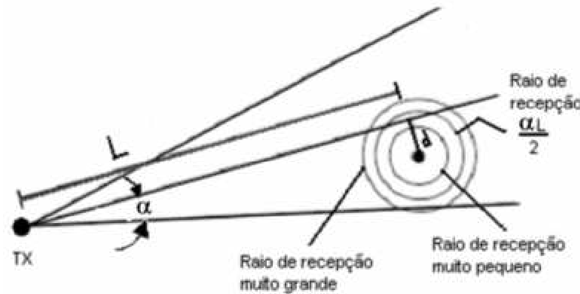

Figura A.2: Problema de dupla contagem na esfera de recepção [70]

PUC-Rio - Certificação Digital Nº 0310507/CB

Outros esquemas de recepção usam a informação de raios adjacentes, e se um destes ângulos for menor ou igual que "α", o raio correspondente e o novo raio recebido deverão pertencer à mesma frente de ondas [70]. A figura (A.3) ilustra êste critério, segundo o qual, para os raios 1 e 2 interceptaram a esfera de recepção, o segmento $\bar{AB}$ do triângulo OAB deve satisfazer a relação:

$$\sqrt{L_1^2 + L_2^2 - 2\ L_1 L_2\ cos\alpha} \leq 2r \qquad \text{(A-2)}$$

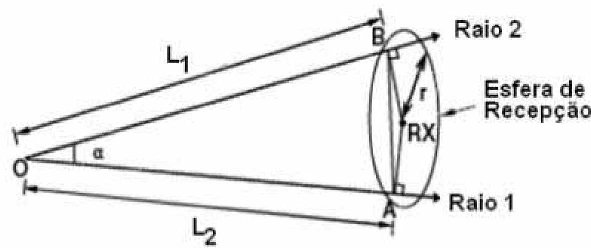

Figura A.3: Problema de dupla contagem na esfera de recepção

A.2 Antena Transmissora (Tx)[3]

No esquema do lançamento de raios, os raios emanam de uma esfera centrada na localização do transmissor. Existem duas características desejáveis na geometria do lançamento de raios, conhecidas como critérios de uniformidade:

– Uniformidade em grande escala - Os raios lançados serão distribuídos igualmente ao redor da esfera de modo que todas as regiões do espaço sejam iluminadas igualmente pelos raios, permitindo uma cobertura uniforme em três dimensões.

– Uniformidade em pequena escala - O modelo de raios é localmente uniforme, correspondendo a ângulos iguais entre um raio e seus vizinhos, o que ajuda na interpretação da informação de frentes de ondas.

Lançar raios a partir dos vértices de um poliedro regular é uma das formas de satisfazer os dois critérios de uniformidade. No entanto, nenhum poliedro regular tem mais de 20 vértices e lançar raios com precisão maior requer muito mais vértices, sendo necessário pesquisar geometrias como o icosaedro (figura A.4).

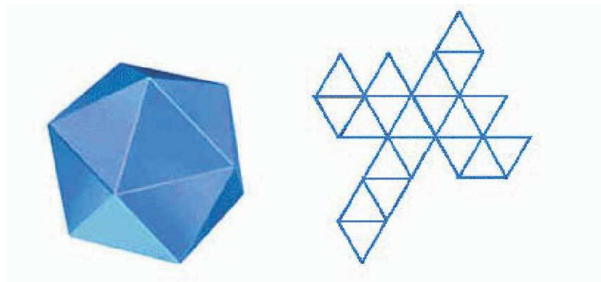

Figura A.4: Icosaedro formado por 20 triângulos

PUC-Rio - Certificação Digital Nº 0310507/CB

Coordenadas geométricas esféricas

A primeira tentativa de formular uma cobertura uniforme em três dimensões usualmente começa com o sistema de coordenadas esféricas, por ser simples de calcular e manipular. Uma forma direta, porém incorreta, corresponde à divisão dos ângulos de azimute (ϕ) e elevação (θ) em incrementos iguais conforme ilustrado na figura (A.5). Este método falha porque nos pólos a esfera apresenta um cluster de vértices densos, enquanto os vértices no equador são espaçados e o lançamento de raios nesta geometria terá uma grande variedade de ângulos de separação.

A matemática geodésica tem todas as características para ser usada no lançamento de raios. Já que o vértice da esfera geodésica prevê uma distribuição uniforme no espaçamento de pontos.

A esfera geodésica surge da divisão das faces de poliedros regulares de 20 lados (icosaedro) e da extrapolação do ponto de interseção à superfície de uma esfera. Cada face do icosaedro é então subdividida em triângulos equiláteros e N representa a freqüência de subdivisão.

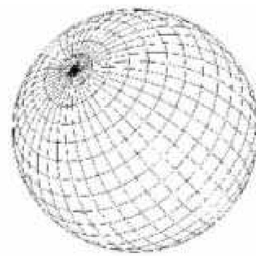

Figura A.5: Grade esférica com espaçamento angular uniforme

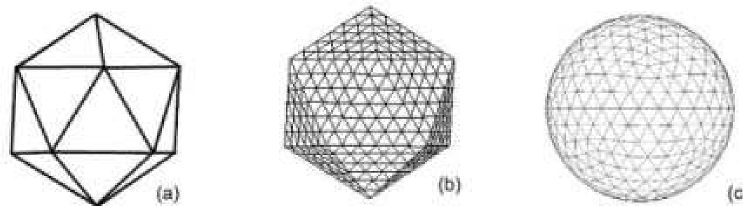

Figura A.6: (a) Um poliedro de 20 lados (b) uma extrapolação da superfície da esfera, (c) vértices de lançamento geocêntrico.

PUC-Rio - Certificação Digital Nº 0310507/CB

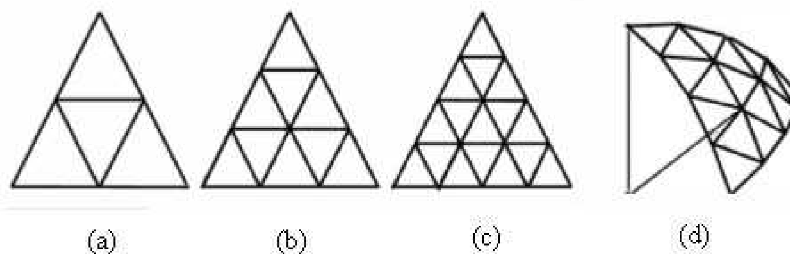

Figura A.7: Face com diferentes frequências de subdivisão : (a) N=2, (b) N=3, (c) N=4, (d) face extrapolada.

A Figura A.7 representa as facetas geodésicas e vértices para lançamento de raios com equivalente separação angular ao redor de uma esfera completa. Além disso, cada um dos doze pontos vértices do icosaedro tem exatamente seis raios vizinhos que rodeiam ao original em um modelo hexagonal.

A.2.1 Construção de Esferas Geodésicas

O número total de pontos sobre uma esfera geodésica está relacionada com sua freqüência de subdivisão N; uma estrutura sem subdivisões corresponde a $N = 1$. A Tabela A.1 mostra alguns parâmetros da esfera geodésica para valores particulares. Os pontos de lançamento de uma esfera geodésica estão referenciados como um conjunto de coordenadas inteiras que expandam todos os pontos sobre todas as faces do icosaedro subdividido. A Figura (A.8)

mostra como localizar um ponto sobre uma face, com ajuda de um sistema de coordenadas triangulares.

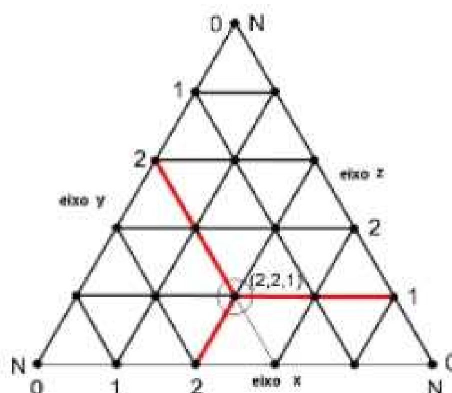

Figura A.8: Sistema de Coordenadas triangular ($N = 5$)

Observe-se que as coordenadas triangulares são dependentes uma da outra, para uma referência completa, só se precisa-se de x e y devido à identidade: (equação (A-3)).

$$x + y + z = N \tag{A-3}$$

Já que existe um total de 20 faces, outra coordenada inteira k é usada para representar a face de um icosaedro ($0 \leq k \leq 19$), ou seja, as três coordenadas (x, y, k) podem referenciar qualquer dos pontos geodésicos.

Sobre um icosaedro	Faces Bordas Vértices	20 30 12
Faces geodesicas	$20N^2$	
Total de Vertices geodesicos	$10N^2 + 2$	
Vértices geodésicos	Por borda de icos. Por Face de icos. Por Face (Int.) de icos	$N - 1$ $1/2(N + 2)(N + 1)$ $1/2(N - 1)(N - 2)$

Tabela A.1: Atributos de uma esfera geodésica em função da freqüência de subdivisão N.

PUC-Rio - Certificação Digital Nº 0310507/CB

B
Códigos corresponde aos programas

Neste apêndice são a presentados os códigos desenvolvidos, correspondendo aos métodos de traçado de raios e FDTD. Estes foram construídos usando a liguagem de programação C++ (Compilador Borland C++ Builder 5.0).

B.1
Códigos correspondentes ao método de traçado de raios

```
//----------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
USERES("Tracado.res");
USEFORM("fViewdxf.cpp", frmTracado);
USEFORM("AboutOGL.cpp", frmAboutOGL);
USEFORM("fNovaFonte.cpp", dlgNovaFonte);
USEUNIT("..\CETUC\LIB\FILEIO\FileIO.cpp");
USEUNIT("Tree.cpp");
USEUNIT("Vetor3D.cpp");
USEUNIT("Solido.cpp");
USEUNIT("DXF.cpp");
USEUNIT("SolidoDxf.cpp");
USEUNIT("Antena.cpp");
USEUNIT("Fonte.cpp");
USEFORM("TIField.cpp", IField);
USEFORM("TIFieldAtt.cpp", IFieldAtt);
USEFORM("Dlg_EnvSettings.cpp", EnvSettings);
//----------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   try
   {
      Application->Initialize();
      Application->CreateForm(__classid(TfrmTracado), &frmTracado);
Application->CreateForm(__classid(TfrmAboutOGL), &frmAboutOGL);
Application->CreateForm(__classid(TdlgNovaFonte), &dlgNovaFonte);
Application->CreateForm(__classid(TIField), &IField);
Application->CreateForm(__classid(TIFieldAtt), &IFieldAtt);
Application->CreateForm(__classid(TEnvSettings), &EnvSettings);
Application->Run();
   }
   catch (Exception &exception)
   {
      Application->ShowException(&exception);
   }
   return 0;
}
//----------------------------------------------------

//----------------------------------------------------
#ifndef TreeH
#define TreeH
//----------------------------------------------------
template <class T> class TreeNode
{
   private:
   public:
      const T Data;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      const int Level;
      TreeNode *Father;
      TreeNode *PrevBrother;
      TreeNode *Son;
      TreeNode *NextBrother;
   public:
      TreeNode( const T &D, TreeNode *F, TreeNode *B, int L );
      ~TreeNode( void );
      TreeNode *GetFather( void ) const { return Father };
      TreeNode *GetSon( void ) const { return Son };
      TreeNode *GetPrevBrother( void ) const { return PrevBrother };
      TreeNode *GetNextBrother( void ) const { return NextBrother };
      void AddBrother( const T &B, int L );
      void ChangeNextBrother( TreeNode *B );
      void AddSon( const T &S, int L );
      void ChangeSon( TreeNode *S );
};
//-------------------------------------------------------
template <class T> TreeNode<T>::TreeNode(
   const T &D, TreeNode<T> *F, TreeNode<T> *B, int L )
:
Data( D ),
Father( F ),
PrevBrother( B ),
Level( L + 1 )
{
   Son = NULL;
   NextBrother = NULL;
}
//-------------------------------------------------------
template <class T> TreeNode<T>::~TreeNode( void )
{
   if( Father != NULL )
   {
      // PrevBrother must be == NULL
      Father->ChangeSon( NextBrother );
   }
   else if( PrevBrother != NULL )
   {
      PrevBrother->ChangeNextBrother( NextBrother );
   }
   else // Root Node
   {
      // throw;
   }
   if( Son != NULL )
   {
      delete Son;
   }
}
//-------------------------------------------------------
template <class T> void TreeNode<T>::AddBrother(
   const T &B, int L )
{
   TreeNode *tn;
   TreeNode *LastBrother;
   tn = NextBrother;
   while( tn != NULL )
   {
      if( tn->NextBrother == NULL )
      {
         LastBrother = tn;
      }
      tn = tn->NextBrother;
   }
   LastBrother->NextBrother = new TreeNode<T>( B, NULL, this, L );
}
//-------------------------------------------------------
template <class T> void TreeNode<T>::ChangeNextBrother( TreeNode *B )
{
   NextBrother = B;
}
//-------------------------------------------------------
template <class T> void TreeNode<T>::AddSon( const T &S, int L )
{
   if( Son != NULL )
   {
      Son->AddBrother( S, L );
   }
   else
   {
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      Son = new TreeNode<T>( S, this, NULL, L );
   }
}
//---------------------------------------------------
template <class T> void TreeNode<T>::ChangeSon( TreeNode *S )
{
   Son = S;
}
//---------------------------------------------------
//TreeNode<int> * meuno = new TreeNode<int>(10);
//---------------------------------------------------
#endif
% Tree.h
\begin{verbatim}
//---------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
#include "Tree.h"
//---------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------

//---------------------------------------------------
#ifndef Vetor3DH
#define Vetor3DH
//---------------------------------------------------
#include <complex.h>
//---------------------------------------------------
class Vetor3D;
class CVetor3D
{
   private:
   public:
      complex<double> x;
      complex<double> y;
      complex<double> z;
   public:
      __fastcall CVetor3D( void );
      __fastcall CVetor3D(
         complex<double> px, complex<double> py, complex<double> pz );
      __fastcall CVetor3D( const CVetor3D &v );
      __fastcall CVetor3D( const Vetor3D &v );
      __fastcall ~CVetor3D( void );
      //
      // Soma e subtracao
      //
      CVetor3D operator+( const CVetor3D &A ) const;
      CVetor3D operator-( const CVetor3D &A ) const;
      CVetor3D operator-( void ) const;
      CVetor3D operator+=( const CVetor3D &c );
      CVetor3D operator-=( const CVetor3D &c );
      //
      // Multiplicacao e Divisao por escalar
      //
      CVetor3D operator*( complex<double> c ) const;
      CVetor3D operator/( complex<double> c ) const;
      CVetor3D operator*=( complex<double> c );
      CVetor3D operator/=( complex<double> c );
      //
      // Metrica, Norma e Produto interno
      //
      double Distancia( const CVetor3D &A ) const;
      double Norma( void ) const;
      CVetor3D Normaliza( void );
      CVetor3D Normalizado( void ) const;
      complex<double> operator*( const CVetor3D &A ) const;
      //
      // Produto vetorial
      //
      CVetor3D operator%( const CVetor3D &A ) const;
};
CVetor3D operator*( complex<double> c, CVetor3D &V );
class Vetor3D
{
   private:
   public:
      double x;
      double y;
      double z;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   public:
      __fastcall Vetor3D( void );
      __fastcall Vetor3D( double px, double py, double pz );
      __fastcall Vetor3D( const Vetor3D &v );
      __fastcall ~Vetor3D( void );
      //
      // Soma e subtracao
      //
      Vetor3D operator+( const Vetor3D &A ) const;
      Vetor3D operator-( const Vetor3D &A ) const;
      Vetor3D operator-( void ) const;
      Vetor3D operator+=( const Vetor3D &c );
      Vetor3D operator-=( const Vetor3D &c );
      //
      // Multiplicacao e Divisao por escalar
      //
      Vetor3D operator*( double c ) const;
      Vetor3D operator/( double c ) const;
      Vetor3D operator*=( double c );
      Vetor3D operator/=( double c );
      //
      // Multiplicação por escalar complexo
      //
      CVetor3D operator*( complex<double> c ) const;
      CVetor3D operator/( complex<double> c ) const;
      //
      // Metrica, Norma e Produto interno
      //
      double Distancia( const Vetor3D &A ) const;
      double Norma( void ) const;
      Vetor3D Normaliza( void );
      Vetor3D Normalizado( void ) const;
      double operator*( const Vetor3D &A ) const;
      //
      // Produto vetorial
      //
      Vetor3D operator%( const Vetor3D &A ) const;
};
Vetor3D operator*( double c, Vetor3D &V );
CVetor3D operator*( complex<double> c, Vetor3D &V );
class Operador3D
{
   public:
      double Axx;
      double Axy;
      double Axz;
      double Ayx;
      double Ayy;
      double Ayz;
      double Azx;
      double Azy;
      double Azz;
   public:
      __fastcall Operador3D( void );
      __fastcall Operador3D( const Operador3D &A );
      __fastcall Operador3D( const double e[3][3] );
      __fastcall Operador3D(
         const Vetor3D c1, const Vetor3D c2, const Vetor3D c3 );
      __fastcall ~Operador3D( void );
      void __fastcall LoadIdentidade( void );
      //
      // Soma e Subtração de Operadores
      //
      Operador3D operator+( const Operador3D &A ) const;
      Operador3D operator-( const Operador3D &A ) const;
      Operador3D operator-( void ) const;
      Operador3D operator+=( const Operador3D &A );
      Operador3D operator-=( const Operador3D &A );
      //
      // Multiplicacao e Divisao por escalar
      //
      Operador3D operator*( double c ) const;
      Operador3D operator/( double c ) const;
      Operador3D operator*=( double c );
      Operador3D operator/=( double c );
      //
      // Produto de operadores
      //
      Operador3D operator*( const Operador3D &A ) const;
      //
      // Transposta
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      //
      Operador3D operator!( void ) const;
      //
      // Operando em vetores
      //
      Vetor3D operator*( const Vetor3D &V ) const;
};
Operador3D Zero( void );
Operador3D Identidade( void );
Operador3D KetBra( const Vetor3D &k1, const Vetor3D &b1 );
Operador3D Projetor( const Vetor3D &p );
Operador3D operator*( double c, const Operador3D &A );
class COperador3D
{
   public:
      complex<double> Axx;
      complex<double> Axy;
      complex<double> Axz;
      complex<double> Ayx;
      complex<double> Ayy;
      complex<double> Ayz;
      complex<double> Azx;
      complex<double> Azy;
      complex<double> Azz;
   public:
      __fastcall COperador3D( void );
      __fastcall COperador3D( const Operador3D &A );
      __fastcall COperador3D( const COperador3D &A );
      __fastcall COperador3D( const complex<double> e[3][3] );
      __fastcall COperador3D(
         const CVetor3D c1, const CVetor3D c2, const CVetor3D c3 );
      __fastcall ~COperador3D( void );
      void __fastcall LoadIdentidade( void );
      //
      // Soma e Subtração de Operadores
      //
      COperador3D operator+( const COperador3D &A ) const;
      COperador3D operator-( const COperador3D &A ) const;
      COperador3D operator-( void ) const;
      COperador3D operator+=( const COperador3D &A );
      COperador3D operator-=( const COperador3D &A );
      //
      // Multiplicacao e Divisao por escalar
      //
      COperador3D operator*( complex<double> c ) const;
      COperador3D operator/( complex<double> c ) const;
      COperador3D operator*=( complex<double> c );
      COperador3D operator/=( complex<double> c );
      //
      // Produto de operadores
      //
      COperador3D operator*( const COperador3D &A ) const;
      //
      // Conjugado Hermitiano
      //
      COperador3D operator!( void ) const;
      //
      // Operando em vetores
      //
      CVetor3D operator*( const CVetor3D &V ) const;
};
COperador3D KetBra( const CVetor3D &k1, const CVetor3D &b1 );
COperador3D Projetor( const CVetor3D &p );
COperador3D operator*( double c, const COperador3D &A );
COperador3D operator*( complex<double> c, const COperador3D &A );
void CartesianToPolarAxis(
   const Vetor3D p, const Vetor3D ex, const Vetor3D ey, const Vetor3D ez,
   double *R, double *Theta, double *Phi );
//----------------------------------------------------
#endif

//----------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
#include "Vetor3D.h"
#include <math.h>
//----------------------------------------------------
#pragma package(smart_init)
//----------------------------------------------------
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
static double Minimo = 1.0e-10;
//----------------------------------------------------
__fastcall CVetor3D::CVetor3D( void )
{
   x = 0.0;
   y = 0.0;
   z = 0.0;
}
//----------------------------------------------------
__fastcall CVetor3D::CVetor3D(
   complex<double> px, complex<double> py, complex<double> pz )
{
   x = px;
   y = py;
   z = pz;
}
//----------------------------------------------------
__fastcall CVetor3D::CVetor3D( const CVetor3D &v )
{
   x = v.x;
   y = v.y;
   z = v.z;
}
//----------------------------------------------------
__fastcall CVetor3D::CVetor3D( const Vetor3D &v )
{
   x = v.x;
   y = v.y;
   z = v.z;
}
//----------------------------------------------------
__fastcall CVetor3D::~CVetor3D( void )
{
}
//----------------------------------------------------
CVetor3D CVetor3D::operator+( const CVetor3D &A ) const
{
   CVetor3D Result;
   Result.x = x + A.x;
   Result.y = y + A.y;
   Result.z = z + A.z;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator-( const CVetor3D &A ) const
{
   CVetor3D Result;
   Result.x = x - A.x;
   Result.y = y - A.y;
   Result.z = z - A.z;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator-( void ) const
{
   CVetor3D Result;
   Result.x = -x;
   Result.y = -y;
   Result.z = -z;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator*( complex<double> c ) const
{
   CVetor3D Result;
   Result.x = x * c;
   Result.y = y * c;
   Result.z = z * c;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator/( complex<double> c ) const
{
   CVetor3D Result;
   Result.x = x / c;
   Result.y = y / c;
   Result.z = z / c;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator+=( const CVetor3D &c )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
   x += c.x;
   y += c.y;
   z += c.z;
   return *this;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator-=( const CVetor3D &c )
{
   x -= c.x;
   y -= c.y;
   z -= c.z;
   return *this;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator*=( complex<double> c )
{
   x *= c;
   y *= c;
   z *= c;
   return *this;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator/=( complex<double> c )
{
   x /= c;
   y /= c;
   z /= c;
   return *this;
}
//----------------------------------------------------
double CVetor3D::Distancia( const CVetor3D &A ) const
{
   double Result;
   Result = (*this - A).Norma();
   return Result;
}
//----------------------------------------------------
double CVetor3D::Norma( void ) const
{
   double Result;
   Result = sqrt( abs( (*this) * (*this) ) );
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::Normaliza( void )
{
   double Norm = this->Norma();
   if( Norm > Minimo )
   {
      *this /= this->Norma();
   }
   return *this;
}
//----------------------------------------------------
CVetor3D CVetor3D::Normalizado( void ) const
{
   CVetor3D ret;
   double Norm = this->Norma();
   if( Norm > Minimo )
   {
      ret = *this / Norm;
   }
   return ret;
}
//----------------------------------------------------
complex<double> CVetor3D::operator*( const CVetor3D &A ) const
{
   complex<double> Result;
   Result = conj( x ) * A.x + conj( y ) * A.y + conj( z ) * A.z;
   return Result;
}
//----------------------------------------------------
CVetor3D CVetor3D::operator%( const CVetor3D &A ) const
{
   CVetor3D Result;
   Result.x = y * A.z - z * A.y;
   Result.y = z * A.x - x * A.z;
   Result.z = x * A.y - y * A.x;
   return Result;
}
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
//----------------------------------------------------
CVetor3D operator*( complex<double> c, CVetor3D &V )
{
   CVetor3D Result;
   Result.x = V.x * c;
   Result.y = V.y * c;
   Result.z = V.z * c;
   return Result;
}
//----------------------------------------------------
//----------------------------------------------------
__fastcall Vetor3D::Vetor3D( void )
{
   x = 0.0;
   y = 0.0;
   z = 0.0;
}
//----------------------------------------------------
__fastcall Vetor3D::Vetor3D( double px, double py, double pz )
{
   x = px;
   y = py;
   z = pz;
}
//----------------------------------------------------
__fastcall Vetor3D::Vetor3D( const Vetor3D &v )
{
   x = v.x;
   y = v.y;
   z = v.z;
}
//----------------------------------------------------
__fastcall Vetor3D::~Vetor3D( void )
{
}
//----------------------------------------------------
Vetor3D Vetor3D::operator+( const Vetor3D &A ) const
{
   Vetor3D Result;
   Result.x = x + A.x;
   Result.y = y + A.y;
   Result.z = z + A.z;
   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator-( const Vetor3D &A ) const
{
   Vetor3D Result;
   Result.x = x - A.x;
   Result.y = y - A.y;
   Result.z = z - A.z;

   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator-( void ) const
{
   Vetor3D Result;
   Result.x = -x;
   Result.y = -y;
   Result.z = -z;
   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator+=( const Vetor3D &c )
{
   x += c.x;
   y += c.y;
   z += c.z;
   return *this;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator-=( const Vetor3D &c )
{
   x -= c.x;
   y -= c.y;
   z -= c.z;
   return *this;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator*( double c ) const
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
   Vetor3D Result;
   Result.x = x * c;
   Result.y = y * c;
   Result.z = z * c;
   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator/( double c ) const
{
   Vetor3D Result;
   Result.x = x / c;
   Result.y = y / c;
   Result.z = z / c;
   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator*=( double c )
{
   x *= c;
   y *= c;
   z *= c;
   return *this;
}
//----------------------------------------------------
Vetor3D Vetor3D::operator/=( double c )
{
   x /= c;
   y /= c;
   z /= c;
   return *this;
}
//----------------------------------------------------
CVetor3D Vetor3D::operator*( complex<double> c ) const
{
   CVetor3D Result;
   Result.x = x * c;
   Result.y = y * c;
   Result.z = z * c;
   return Result;
}
//----------------------------------------------------
CVetor3D Vetor3D::operator/( complex<double> c ) const
{
   CVetor3D Result;
   Result.x = x / c;
   Result.y = y / c;
   Result.z = z / c;
   return Result;
}
//----------------------------------------------------
double Vetor3D::Distancia( const Vetor3D &A ) const
{
   double Result;
   Result = (*this - A).Norma();
   return Result;
}
//----------------------------------------------------
double Vetor3D::Norma( void ) const
{
   double Result;
   Result = sqrt( (*this) * (*this) );
   return Result;
}
//----------------------------------------------------
Vetor3D Vetor3D::Normaliza( void )
{
   double Norm = this->Norma();
   if( Norm > Minimo )
   {
      *this /= Norm;
   }
   return *this;
}
//----------------------------------------------------
Vetor3D Vetor3D::Normalizado( void ) const
{
   Vetor3D ret;
   double Norm = this->Norma();
   if( Norm > Minimo )
   {
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      ret = *this / Norm;
   }
   return ret;
}
//-----------------------------------------------------
double Vetor3D::operator*( const Vetor3D &A ) const
{
   double Result;
   Result = x * A.x + y * A.y + z * A.z;
   return Result;
}
//-----------------------------------------------------
Vetor3D Vetor3D::operator%( const Vetor3D &A ) const
{
   Vetor3D Result;
   Result.x = y * A.z - z * A.y;
   Result.y = z * A.x - x * A.z;
   Result.z = x * A.y - y * A.x;
   return Result;
}
//-----------------------------------------------------
Vetor3D operator*( double c, Vetor3D &V )
{
   Vetor3D Result;
   Result.x = V.x * c;
   Result.y = V.y * c;
   Result.z = V.z * c;
   return Result;
}
//-----------------------------------------------------
CVetor3D operator*( complex<double> c, Vetor3D &V )
{
   CVetor3D Result;
   Result.x = V.x * c;
   Result.y = V.y * c;
   Result.z = V.z * c;
   return Result;
}
//-----------------------------------------------------
//-----------------------------------------------------
__fastcall Operador3D::Operador3D( void )
{
   Axx = 0.0;
   Axy = 0.0;
   Axz = 0.0;
   Ayx = 0.0;
   Ayy = 0.0;
   Ayz = 0.0;
   Azx = 0.0;
   Azy = 0.0;
   Azz = 0.0;
}
//-----------------------------------------------------
__fastcall Operador3D::Operador3D( const Operador3D &A )
{
   Axx = A.Axx;
   Axy = A.Axy;
   Axz = A.Axz;
   Ayx = A.Ayx;
   Ayy = A.Ayy;
   Ayz = A.Ayz;
   Azx = A.Azx;
   Azy = A.Azy;
   Azz = A.Azz;
}
//-----------------------------------------------------
__fastcall Operador3D::Operador3D( const double e[3][3] )
{
   Axx = e[0][0];
   Axy = e[0][1];
   Axz = e[0][2];
   Ayx = e[1][0];
   Ayy = e[1][1];
   Ayz = e[1][2];
   Azx = e[2][0];
   Azy = e[2][1];
   Azz = e[2][2];
}
//-----------------------------------------------------
__fastcall Operador3D::Operador3D(
   const Vetor3D c1, const Vetor3D c2, const Vetor3D c3 )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
   Axx = c1.x;
   Axy = c1.y;
   Axz = c1.z;
   Ayx = c2.x;
   Ayy = c2.y;
   Ayz = c2.z;
   Azx = c3.x;
   Azy = c3.y;
   Azz = c3.z;
}
//-------------------------------------------------------
__fastcall Operador3D::~Operador3D( void )
{
}
//-------------------------------------------------------
void __fastcall Operador3D::LoadIdentidade( void )
{
   Axx = 1.0;
   Axy = 0.0;
   Axz = 0.0;
   Ayx = 0.0;
   Ayy = 1.0;
   Ayz = 0.0;
   Azx = 0.0;
   Azy = 0.0;
   Azz = 1.0;
}
//-------------------------------------------------------
Operador3D Operador3D::operator+( const Operador3D &A ) const
{
   Operador3D ret;
   ret.Axx = Axx + A.Axx;
   ret.Axy = Axy + A.Axy;
   ret.Axz = Axz + A.Axz;
   ret.Ayx = Ayx + A.Ayx;
   ret.Ayy = Ayy + A.Ayy;
   ret.Ayz = Ayz + A.Ayz;
   ret.Azx = Azx + A.Azx;
   ret.Azy = Azy + A.Azy;
   ret.Azz = Azz + A.Azz;
   return ret;
}
//-------------------------------------------------------
Operador3D Operador3D::operator-( const Operador3D &A ) const
{
   Operador3D ret;
   ret.Axx = Axx - A.Axx;
   ret.Axy = Axy - A.Axy;
   ret.Axz = Axz - A.Axz;
   ret.Ayx = Ayx - A.Ayx;
   ret.Ayy = Ayy - A.Ayy;
   ret.Ayz = Ayz - A.Ayz;
   ret.Azx = Azx - A.Azx;
   ret.Azy = Azy - A.Azy;
   ret.Azz = Azz - A.Azz;
   return ret;
}
//-------------------------------------------------------
Operador3D Operador3D::operator-( void ) const
{
   Operador3D ret;
   ret.Axx = -Axx;
   ret.Axy = -Axy;
   ret.Axz = -Axz;
   ret.Ayx = -Ayx;
   ret.Ayy = -Ayy;
   ret.Ayz = -Ayz;
   ret.Azx = -Azx;
   ret.Azy = -Azy;
   ret.Azz = -Azz;
   return ret;
}
//-------------------------------------------------------
Operador3D Operador3D::operator+=( const Operador3D &A )
{
   Operador3D ret;
   Axx += A.Axx;
   Axy += A.Axy;
   Axz += A.Axz;
   Ayx += A.Ayx;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   Ayy += A.Ayy;
   Ayz += A.Ayz;
   Azx += A.Azx;
   Azy += A.Azy;
   Azz += A.Azz;
   return ret;
}
//-----------------------------------------------------
Operador3D Operador3D::operator-=( const Operador3D &A )
{
   Operador3D ret;
   Axx -= A.Axx;
   Axy -= A.Axy;
   Axz -= A.Axz;
   Ayx -= A.Ayx;
   Ayy -= A.Ayy;
   Ayz -= A.Ayz;
   Azx -= A.Azx;
   Azy -= A.Azy;
   Azz -= A.Azz;
   return ret;
}
//-----------------------------------------------------
Operador3D Operador3D::operator*( double c ) const
{
   Operador3D ret;
   ret.Axx = Axx * c;
   ret.Axy = Axy * c;
   ret.Axz = Axz * c;
   ret.Ayx = Ayx * c;
   ret.Ayy = Ayy * c;
   ret.Ayz = Ayz * c;
   ret.Azx = Azx * c;
   ret.Azy = Azy * c;
   ret.Azz = Azz * c;
   return ret;
}
//-----------------------------------------------------
Operador3D Operador3D::operator/( double c ) const
{
   Operador3D ret;
   ret.Axx = Axx / c;
   ret.Axy = Axy / c;
   ret.Axz = Axz / c;
   ret.Ayx = Ayx / c;
   ret.Ayy = Ayy / c;
   ret.Ayz = Ayz / c;
   ret.Azx = Azx / c;
   ret.Azy = Azy / c;
   ret.Azz = Azz / c;
   return ret;
}
//-----------------------------------------------------
Operador3D Operador3D::operator*=( double c )
{
   Operador3D ret;
   Axx *= c;
   Axy *= c;
   Axz *= c;
   Ayx *= c;
   Ayy *= c;
   Ayz *= c;
   Azx *= c;
   Azy *= c;
   Azz *= c;
   return ret;
}
//-----------------------------------------------------
Operador3D Operador3D::operator/=( double c )
{
   Operador3D ret;
   Axx /= c;
   Axy /= c;
   Axz /= c;
   Ayx /= c;
   Ayy /= c;
   Ayz /= c;
   Azx /= c;
   Azy /= c;
   Azz /= c;
   return ret;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
}
//----------------------------------------------------
Operador3D Operador3D::operator*( const Operador3D &A ) const
{
   Operador3D ret;
   ret.Axx = Axx * A.Axx + Axy * A.Ayx + Axz * A.Azx;
   ret.Axy = Axx * A.Axy + Axy * A.Ayy + Axz * A.Azy;
   ret.Axz = Axx * A.Axz + Axy * A.Ayz + Axz * A.Azz;
   ret.Ayx = Ayx * A.Axx + Ayy * A.Ayx + Ayz * A.Azx;
   ret.Ayy = Ayx * A.Axy + Ayy * A.Ayy + Ayz * A.Azy;
   ret.Ayz = Ayx * A.Axz + Ayy * A.Ayz + Ayz * A.Azz;
   ret.Azx = Azx * A.Axx + Azy * A.Ayx + Azz * A.Azx;
   ret.Azy = Azx * A.Axy + Azy * A.Ayy + Azz * A.Azy;
   ret.Azz = Azx * A.Axz + Azy * A.Ayz + Azz * A.Azz;
   return ret;
}
//----------------------------------------------------
Operador3D Operador3D::operator!( void ) const
{
   Operador3D ret;
   ret.Axx = Axx;
   ret.Axy = Ayx;
   ret.Axz = Azx;
   ret.Ayx = Axy;
   ret.Ayy = Ayy;
   ret.Ayz = Azy;
   ret.Azx = Axz;
   ret.Azy = Ayz;
   ret.Azz = Azz;
   return ret;
}
//----------------------------------------------------
Vetor3D Operador3D::operator*( const Vetor3D &V ) const
{
   Vetor3D ret;
   ret.x = Axx * V.x + Axy * V.y + Axz * V.z;
   ret.y = Ayx * V.x + Ayy * V.y + Ayz * V.z;
   ret.z = Azx * V.x + Azy * V.y + Azz * V.z;
   return ret;
}
//----------------------------------------------------
Operador3D Zero( void )
{
   Operador3D ret;
   ret.Axx = 0.0;
   ret.Axy = 0.0;
   ret.Axz = 0.0;
   ret.Ayx = 0.0;
   ret.Ayy = 0.0;
   ret.Ayz = 0.0;
   ret.Azx = 0.0;
   ret.Azy = 0.0;
   ret.Azz = 0.0;
   return ret;
}
//----------------------------------------------------
Operador3D Identidade( void )
{
   Operador3D ret;
   ret.Axx = 1.0;
   ret.Axy = 0.0;
   ret.Axz = 0.0;
   ret.Ayx = 0.0;
   ret.Ayy = 1.0;
   ret.Ayz = 0.0;
   ret.Azx = 0.0;
   ret.Azy = 0.0;
   ret.Azz = 1.0;
   return ret;
}
//----------------------------------------------------
Operador3D KetBra( const Vetor3D &k1, const Vetor3D &b1 )
{
   Operador3D ret;
   Vetor3D k( k1.Normalizado() );
   Vetor3D b( b1.Normalizado() );
   ret.Axx = k.x * b.x;
   ret.Axy = k.x * b.y;
   ret.Axz = k.x * b.z;
   ret.Ayx = k.y * b.x;
   ret.Ayy = k.y * b.y;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ret.Ayz = k.y * b.z;
   ret.Azx = k.z * b.x;
   ret.Azy = k.z * b.y;
   ret.Azz = k.z * b.z;
   return ret;
}
//-----------------------------------------------------
Operador3D Projetor( const Vetor3D &p )
{
   Operador3D ret;
   Vetor3D n( p.Normalizado() );
   ret.Axx = n.x * n.x;
   ret.Axy = n.x * n.y;
   ret.Axz = n.x * n.z;
   ret.Ayx = n.y * n.x;
   ret.Ayy = n.y * n.y;
   ret.Ayz = n.y * n.z;
   ret.Azx = n.z * n.x;
   ret.Azy = n.z * n.y;
   ret.Azz = n.z * n.z;
   return ret;
}
//-----------------------------------------------------
Operador3D operator*( double c, const Operador3D &A )
{
   Operador3D ret;
   ret.Axx = A.Axx * c;
   ret.Axy = A.Axy * c;
   ret.Axz = A.Axz * c;
   ret.Ayx = A.Ayx * c;
   ret.Ayy = A.Ayy * c;
   ret.Ayz = A.Ayz * c;
   ret.Azx = A.Azx * c;
   ret.Azy = A.Azy * c;
   ret.Azz = A.Azz * c;
   return ret;
}
//-----------------------------------------------------
__fastcall COperador3D::COperador3D( void )
{
   Axx = 0.0;
   Axy = 0.0;
   Axz = 0.0;
   Ayx = 0.0;
   Ayy = 0.0;
   Ayz = 0.0;
   Azx = 0.0;
   Azy = 0.0;
   Azz = 0.0;
}
//-----------------------------------------------------
__fastcall COperador3D::COperador3D( const Operador3D &A )
{
   Axx = A.Axx;
   Axy = A.Axy;
   Axz = A.Axz;
   Ayx = A.Ayx;
   Ayy = A.Ayy;
   Ayz = A.Ayz;
   Azx = A.Azx;
   Azy = A.Azy;
   Azz = A.Azz;
}
//-----------------------------------------------------
__fastcall COperador3D::COperador3D( const COperador3D &A )
{
   Axx = A.Axx;
   Axy = A.Axy;
   Axz = A.Axz;
   Ayx = A.Ayx;
   Ayy = A.Ayy;
   Ayz = A.Ayz;
   Azx = A.Azx;
   Azy = A.Azy;
   Azz = A.Azz;
}
//-----------------------------------------------------
__fastcall COperador3D::COperador3D( const complex<double> e[3][3] )
{
   Axx = e[0][0];
   Axy = e[0][1];
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   Axz = e[0][2];
   Ayx = e[1][0];
   Ayy = e[1][1];
   Ayz = e[1][2];
   Azx = e[2][0];
   Azy = e[2][1];
   Azz = e[2][2];
}
//----------------------------------------------------
__fastcall COperador3D::COperador3D(
   const CVetor3D c1, const CVetor3D c2, const CVetor3D c3 )
{
   Axx = c1.x;
   Axy = c1.y;
   Axz = c1.z;
   Ayx = c2.x;
   Ayy = c2.y;
   Ayz = c2.z;
   Azx = c3.x;
   Azy = c3.y;
   Azz = c3.z;
}
//----------------------------------------------------
__fastcall COperador3D::~COperador3D( void )
{
}
//----------------------------------------------------
void __fastcall COperador3D::LoadIdentidade( void )
{
   Axx = 1.0;
   Axy = 0.0;
   Axz = 0.0;
   Ayx = 0.0;
   Ayy = 1.0;
   Ayz = 0.0;
   Azx = 0.0;
   Azy = 0.0;
   Azz = 1.0;
}
//----------------------------------------------------
COperador3D COperador3D::operator+( const COperador3D &A ) const
{
   COperador3D ret;
   ret.Axx = Axx + A.Axx;
   ret.Axy = Axy + A.Axy;
   ret.Axz = Axz + A.Axz;
   ret.Ayx = Ayx + A.Ayx;
   ret.Ayy = Ayy + A.Ayy;
   ret.Ayz = Ayz + A.Ayz;
   ret.Azx = Azx + A.Azx;
   ret.Azy = Azy + A.Azy;
   ret.Azz = Azz + A.Azz;
   return ret;
}
//----------------------------------------------------
COperador3D COperador3D::operator-( const COperador3D &A ) const
{
   COperador3D ret;
   ret.Axx = Axx - A.Axx;
   ret.Axy = Axy - A.Axy;
   ret.Axz = Axz - A.Axz;
   ret.Ayx = Ayx - A.Ayx;
   ret.Ayy = Ayy - A.Ayy;
   ret.Ayz = Ayz - A.Ayz;
   ret.Azx = Azx - A.Azx;
   ret.Azy = Azy - A.Azy;
   ret.Azz = Azz - A.Azz;
   return ret;
}
//----------------------------------------------------
COperador3D COperador3D::operator-( void ) const
{
   COperador3D ret;
   ret.Axx = -Axx;
   ret.Axy = -Axy;
   ret.Axz = -Axz;
   ret.Ayx = -Ayx;
   ret.Ayy = -Ayy;
   ret.Ayz = -Ayz;
   ret.Azx = -Azx;
   ret.Azy = -Azy;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ret.Azz = -Azz;
   return ret;
}
//------------------------------------------------------
COperador3D COperador3D::operator+=( const COperador3D &A )
{
   COperador3D ret;
   Axx += A.Axx;
   Axy += A.Axy;
   Axz += A.Axz;
   Ayx += A.Ayx;
   Ayy += A.Ayy;
   Ayz += A.Ayz;
   Azx += A.Azx;
   Azy += A.Azy;
   Azz += A.Azz;
   return ret;
}
//----------------------------------------------------------
COperador3D COperador3D::operator-=( const COperador3D &A )
{
   COperador3D ret;
   Axx -= A.Axx;
   Axy -= A.Axy;
   Axz -= A.Axz;
   Ayx -= A.Ayx;
   Ayy -= A.Ayy;
   Ayz -= A.Ayz;
   Azx -= A.Azx;
   Azy -= A.Azy;
   Azz -= A.Azz;
   return ret;
}
//-------------------------------------------------------------
COperador3D COperador3D::operator*( complex<double> c ) const
{
   COperador3D ret;
   ret.Axx = Axx * c;
   ret.Axy = Axy * c;
   ret.Axz = Axz * c;
   ret.Ayx = Ayx * c;
   ret.Ayy = Ayy * c;
   ret.Ayz = Ayz * c;
   ret.Azx = Azx * c;
   ret.Azy = Azy * c;
   ret.Azz = Azz * c;
   return ret;
}
//-----------------------------------------------------------
COperador3D COperador3D::operator/( complex<double> c ) const
{
   COperador3D ret;
   ret.Axx = Axx / c;
   ret.Axy = Axy / c;
   ret.Axz = Axz / c;
   ret.Ayx = Ayx / c;
   ret.Ayy = Ayy / c;
   ret.Ayz = Ayz / c;
   ret.Azx = Azx / c;
   ret.Azy = Azy / c;
   ret.Azz = Azz / c;
   return ret;
}
//--------------------------------------------------------
COperador3D COperador3D::operator*=( complex<double> c )
{
   COperador3D ret;
   Axx *= c;
   Axy *= c;
   Axz *= c;
   Ayx *= c;
   Ayy *= c;
   Ayz *= c;
   Azx *= c;
   Azy *= c;
   Azz *= c;
   return ret;
}
//--------------------------------------------------------
COperador3D COperador3D::operator/=( complex<double> c )
{
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   COperador3D ret;
   Axx /= c;
   Axy /= c;
   Axz /= c;
   Ayx /= c;
   Ayy /= c;
   Ayz /= c;
   Azx /= c;
   Azy /= c;
   Azz /= c;
   return ret;
}
//------------------------------------------------------------
COperador3D COperador3D::operator*( const COperador3D &A ) const
{
   COperador3D ret;
   ret.Axx = Axx * A.Axx + Axy * A.Ayx + Axz * A.Azx;
   ret.Axy = Axx * A.Axy + Axy * A.Ayy + Axz * A.Azy;
   ret.Axz = Axx * A.Axz + Axy * A.Ayz + Axz * A.Azz;
   ret.Ayx = Ayx * A.Axx + Ayy * A.Ayx + Ayz * A.Azx;
   ret.Ayy = Ayx * A.Axy + Ayy * A.Ayy + Ayz * A.Azy;
   ret.Ayz = Ayx * A.Axz + Ayy * A.Ayz + Ayz * A.Azz;
   ret.Azx = Azx * A.Axx + Azy * A.Ayx + Azz * A.Azx;
   ret.Azy = Azx * A.Axy + Azy * A.Ayy + Azz * A.Azy;
   ret.Azz = Azx * A.Axz + Azy * A.Ayz + Azz * A.Azz;
   return ret;
}
//-----------------------------------------------------------
COperador3D COperador3D::operator!( void ) const
{
   COperador3D ret;
   ret.Axx = conj( Axx );
   ret.Axy = conj( Ayx );
   ret.Axz = conj( Azx );
   ret.Ayx = conj( Axy );
   ret.Ayy = conj( Ayy );
   ret.Ayz = conj( Azy );
   ret.Azx = conj( Axz );
   ret.Azy = conj( Ayz );
   ret.Azz = conj( Azz );
   return ret;
}
//----------------------------------------------------------
CVetor3D COperador3D::operator*( const CVetor3D &V ) const
{
   CVetor3D ret;
   ret.x = Axx * V.x + Axy * V.y + Axz * V.z;
   ret.y = Ayx * V.x + Ayy * V.y + Ayz * V.z;
   ret.z = Azx * V.x + Azy * V.y + Azz * V.z;
   return ret;
}
//-----------------------------------------------------------
COperador3D KetBra( const CVetor3D &k1, const CVetor3D &b1 )
{
   COperador3D ret;
   CVetor3D k( k1.Normalizado() );
   CVetor3D b( b1.Normalizado() );
   ret.Axx = k.x * conj( b.x );
   ret.Axy = k.x * conj( b.y );
   ret.Axz = k.x * conj( b.z );
   ret.Ayx = k.y * conj( b.x );
   ret.Ayy = k.y * conj( b.y );
   ret.Ayz = k.y * conj( b.z );
   ret.Azx = k.z * conj( b.x );
   ret.Azy = k.z * conj( b.y );
   ret.Azz = k.z * conj( b.z );
   return ret;
}
//--------------------------------------------------------
COperador3D Projetor( const CVetor3D &p )
{
   COperador3D ret;
   CVetor3D n( p.Normalizado() );
   ret.Axx = n.x * conj( n.x );
   ret.Axy = n.x * conj( n.y );
   ret.Axz = n.x * conj( n.z );
   ret.Ayx = n.y * conj( n.x );
   ret.Ayy = n.y * conj( n.y );
   ret.Ayz = n.y * conj( n.z );
   ret.Azx = n.z * conj( n.x );
   ret.Azy = n.z * conj( n.y );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ret.Azz = n.z * conj( n.z );
   return ret;
}
//-----------------------------------------------------------
COperador3D operator*( complex<double> c, const COperador3D &A )
{
   COperador3D ret;
   ret.Axx = A.Axx * c;
   ret.Axy = A.Axy * c;
   ret.Axz = A.Axz * c;
   ret.Ayx = A.Ayx * c;
   ret.Ayy = A.Ayy * c;
   ret.Ayz = A.Ayz * c;
   ret.Azx = A.Azx * c;
   ret.Azy = A.Azy * c;
   ret.Azz = A.Azz * c;
   return ret;
}
//----------------------------------------------------------
//
void CartesianToPolarAxis(
const Vetor3D p, const Vetor3D ex, const Vetor3D ey, const Vetor3D ez,
   double *R, double *Theta, double *Phi )
{
   Operador3D AxisToGlobal( ex, ey, ez );
   Operador3D GlobalToAxis( !AxisToGlobal );
   Vetor3D p2 = GlobalToAxis * p;
 *R = p2.Norma();
 if( *R > Minimo )
 {
*Theta = acos( p2.z / *R );
if(p2.x > Minimo)
*Phi = atan2( p2.y, p2.x );
else
*Phi = ((p2.y > 0)?1:-1) *4.0*atan(1.0f);
   }
 else
   {
      *Theta = 0.0;
      *Phi = 0.0;
   }
}
//---------------------------------------------------------------


//---------------------------------------------------------------
#ifndef DXFH
#define DXFH
//---------------------------------------------------------------
//#include "..\lib\fileio\fileio.h"
#include "Vetor3D.h"
//---------------------------------------------------------------
class DxfVertex
{
public:
   Vetor3D p;
   unsigned char flags;
   int v1;
   int v2;
   int v3;
   int v4;
   DxfVertex *tail;
public:
   __fastcall DxfVertex( void );
   __fastcall DxfVertex(
      double x1, double y1, double z1, unsigned char f,
      int v11, int v21, int v31, int v41 );
   __fastcall ~DxfVertex( void );
};

class dxf
{
private:

public:
   int nGroups;
   double ExtMinX;
   double ExtMinY;
   double ExtMinZ;
   double ExtMaxX;
   double ExtMaxY;
   double ExtMaxZ;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   double LimMinX;
   double LimMinY;
   double LimMaxX;
   double LimMaxY;

public:
   __fastcall dxf( void );
   __fastcall ~dxf( void );
   int __fastcall ReadDxfFile(
      TCollection *TCol, const AnsiString &DxfFilename );
   inline int __fastcall ReadGroup(
      const FileObject &fp, int *code, char *value );
   int __fastcall SkipSection( const FileObject &fp );
   int __fastcall ProcessHeader( const FileObject &fp );
   int __fastcall ProcessExtent( const FileObject &fp,
      double *ExtX, double *ExtY, double *ExtZ );
   int __fastcall ProcessLimit( const FileObject &fp,
      double *LimX, double *LimY );
   int __fastcall ProcessEntities( TCollection *TCol, const FileObject &fp );
   int __fastcall ProcessPolyline( TCollection *TCol, const FileObject &fp );
   int __fastcall ProcessVertexList(
      const FileObject &fp, DxfVertex **Head, int *n );
   void __fastcall AddSolido( TCollection *TCol, DxfVertex *vList, int Color );
   void __fastcall AddSolido2( TCollection *TCol, DxfVertex *vList, int Color );
};
//---------------------------------------------------------------
#endif


//-----------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
#include "DXF.h"
#include <math.h>
#include "SolidoDxf.h"
//-------------------------------------------------------------
#pragma package(smart_init)
//------------------------------------------------------------
static const double pi = 4.0 * atan( 1.0 );
static const Vetor3D Zero3D( 0, 0, 0 );
//------------------------------------------------------------
__fastcall DxfVertex::DxfVertex( void )
:
p( 0.0, 0.0, 0.0 )
{
   flags = 0;
   v1 = 0;
   v2 = 0;
   v3 = 0;
   v4 = 0;
   tail = NULL;
}
//------------------------------------------------------------
__fastcall DxfVertex::DxfVertex(
 double x1, double y1, double z1, unsigned char f,
   int v11, int v21, int v31, int v41 )
:
p( x1, y1, z1 )
{
   flags = f;
   v1 = v11;
   v2 = v21;
   v3 = v31;
   v4 = v41;
 tail = NULL;
}
//-------------------------------------------------------
__fastcall DxfVertex::~DxfVertex( void )
{
   if( tail != NULL )
   {
      delete tail;
   }
}
//-------------------------------------------------------
__fastcall dxf::dxf( void )
{
   ExtMinX = 0.0;
   ExtMinY = 0.0;
   ExtMinZ = 0.0;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ExtMaxX = 0.0;
   ExtMaxY = 0.0;
   ExtMaxZ = 0.0;
   LimMinX = 0.0;
 LimMinY = 0.0;
   LimMaxX = 0.0;
   LimMaxY = 0.0;
}
//----------------------------------------------------
__fastcall dxf::~dxf( void )
{
}
//----------------------------------------------------
int __fastcall dxf::ReadDxfFile(
 TCollection *TCol, const AnsiString &DxfFilename )
{
   int code;
   char value[256];
   int ReadOk;
   int Eof;
   nGroups = 0;
   FileObject fp( DxfFilename.c_str(), "rt" );
   ReadOk = true;
 Eof = false;
   while( !fp.fend() && ReadOk && !Eof )
   {
      ReadOk = ReadGroup( fp, &code, value );
      if( !strcmp( value, "SECTION" ) )
      {
         ReadOk = ReadGroup( fp, &code, value );
         if( !strcmp( value, "HEADER" ) )
         {
            ProcessHeader( fp );
 }
         else if( !strcmp( value, "TABLES" ) )
         {
            SkipSection( fp );
         }
         else if( !strcmp( value, "BLOCKS" ) )
         {
            SkipSection( fp );
         }
         else if( !strcmp( value, "ENTITIES" ) )
 {
            ProcessEntities( TCol, fp );
         }
      }
      else if( !strcmp( value, "EOF" ) )
      {
         Eof = true;
      }
      else
      {
 ReadOk = false;
      }
   }
   return ReadOk && Eof;
}
//----------------------------------------------------
inline int __fastcall dxf::ReadGroup(
   const FileObject &fp, int *code, char *value )
{
   int ret;
   ret = fp.scanf( " %d%*1[\n]%255[^\n] ", code, value );
   if( ret == 1 )
   {
      *value = 0;
   }
   ret = ret == 2 || ret == 1;
   nGroups++;
   Application->ProcessMessages();
   return ret;
}
//----------------------------------------------------
int __fastcall dxf::SkipSection( const FileObject &fp )
{
   int ReadOk;
   int EndSection;
   int code;
   char value[256];
   EndSection = false;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ReadOk = true;
   while( !fp.fend() && ReadOk && !EndSection )
   {
      ReadOk = ReadGroup( fp, &code, value );
      EndSection = !strcmp( value, "ENDSEC" );
   }
   return EndSection;
}
//----------------------------------------------------
int __fastcall dxf::ProcessHeader( const FileObject &fp )
{
   int ReadOk;
   int EndSection;
   int code;
   char value[256];
   EndSection = false;
   ReadOk = true;
   while( !fp.fend() && ReadOk && !EndSection )
   {
      ReadOk = ReadGroup( fp, &code, value );
      switch( code )
      {
         case 0:
            if( !strcmp( value, "ENDSEC" ) )
            {
               EndSection = true;
}
            else
            {
               // ReadOk = false;
               // ignore other types
               ReadOk = true;
            }
            break;
         case 9:
            if( !strcmp( value, "$EXTMIN" ) )
            {
               ReadOk = ProcessExtent( fp, &ExtMinX, &ExtMinY, &ExtMinZ );
            }
            else if( !strcmp( value, "$EXTMAX" ) )
            {
               ReadOk = ProcessExtent( fp, &ExtMaxX, &ExtMaxY, &ExtMaxZ );
            }
            if( !strcmp( value, "$LIMMIN" ) )
            {
               ReadOk = ProcessLimit( fp, &LimMinX, &LimMinY );
}
            else if( !strcmp( value, "$LIMMAX" ) )
            {
               ReadOk = ProcessLimit( fp, &LimMaxX, &LimMaxY );
            }
            else
            {
               // ReadOk = false;
               // ignore other types
               ReadOk = true;
            }
            break;
      }
   }
   return EndSection;
}
//----------------------------------------------------
int __fastcall dxf::ProcessExtent( const FileObject &fp,
   double *ExtX, double *ExtY, double *ExtZ )
{
   int code;
   char value[256];
   int read10 = false;
   int read20 = false;
   int read30 = false;
   int EndExtent = false;
   int ReadOk = true;
   while( !fp.fend() && ReadOk && !EndExtent )
   {
      ReadOk = ReadGroup( fp, &code, value );
      switch( code )
      {
         case 10:
            *ExtX = atof( value );
            read10 = true;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
            break;
         case 20:
            *ExtY = atof( value );
    read20 = true;
            break;
         case 30:
            *ExtZ = atof( value );
            read30 = true;
            break;
         default:
            break;
      }
      EndExtent = read10 && read20 && read30;
   }
   return EndExtent;
}
//-------------------------------------------------------
int __fastcall dxf::ProcessLimit( const FileObject &fp,
   double *LimX, double *LimY )
{
   int code;
   char value[256];
   int read10 = false;
   int read20 = false;
   int EndLimit = false;
   int ReadOk = true;
   while( !fp.fend() && ReadOk && !EndLimit )
   {
      ReadOk = ReadGroup( fp, &code, value );
      switch( code )
      {
         case 10:
            *LimX = atof( value );
            read10 = true;
            break;
         case 20:
            *LimY = atof( value );
            read20 = true;
            break;
         default:
            break;
}
      EndLimit = read10 && read20;
   }
   return EndLimit;
}
//-------------------------------------------------------
int __fastcall dxf::ProcessEntities( TCollection *TCol, const FileObject &fp )
{
   int ReadOk;
   int EndSection;
   int code;
   char value[256];
   char layer[256];
   EndSection = false;
   ReadOk = true;
   while( !fp.fend() && ReadOk && !EndSection )
   {
      ReadOk = ReadGroup( fp, &code, value );
if( !strcmp( value, "POLYLINE" ) )
      {
         // Name of the layer on wich the entity ( Polyline ) resides
         ReadOk = ReadGroup( fp, &code, layer );
         if( code == 8 )
         {
            ReadOk = ProcessPolyline( TCol, fp );
         }
         else
         {
            ReadOk = false;
         }
         #if defined( UNDEFINED1 )
         strupr( layer );
         if(
            !strcmp( layer, "EDIFICACOES" ) ||
            !strcmp( layer, "EDIFICAÇÕES" ) ||
            !strcmp( layer, "IGREJA" ) )
         {
            ReadOk = ProcessPolyline( TCol, fp );
 }
         #endif
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      }
      else if( !strcmp( value, "POINT" ) )
      {
      }
      else if( !strcmp( value, "LINE" ) )
      {
      }
      else if( !strcmp( value, "CIRCLE" ) )
      {
      }
      else if( !strcmp( value, "ARC" ) )
      {
      }
      else if( !strcmp( value, "TRACE" ) )
      {
      }
      else if( !strcmp( value, "SOLID" ) )
      {
}
      else if( !strcmp( value, "3DFACE" ) )
      {
      }
      else if( !strcmp( value, "ENDSEC" ) )
{
         EndSection = true;
      }
      else
      {
         // ReadOk = false;
         // ignore other types
         ReadOk = true;
      }
   }
   return EndSection;
}
//-----------------------------------------------------
int __fastcall dxf::ProcessPolyline( TCollection *TCol, const FileObject &fp )
{
   int code;
   char value[256];
   char layer[256];
   int EndPolyline = false;
 int NumVertex = 0;
 DxfVertex *vList = NULL;
 int PolylineColor = 0;
 try
 {
//
// Leitura da lista de vertices
//
int ReadOk = true;
while( !fp.fend() && ReadOk && !EndPolyline )
{
 ReadOk = ReadGroup( fp, &code, value );
 switch( code )
 {
case 0:
 if( !strcmp( value, "VERTEX" ) )
 {
ReadOk = ReadGroup( fp, &code, layer );
EndPolyline = ProcessVertexList( fp, &vList, &NumVertex );
NumVertex++;
//EndPolyline = true;
 }
 else
 {
// ReadOk = false;
// ignore other types
ReadOk = true;
 }
 break;
case 40:
               // Default starting width - optional 0
               break;
            case 41:
               // Default ending width - optional 0
 break;
            case 62:
               // Color number
 PolylineColor = atoi( value );
               break;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
case 66:
 // Vertices follow flag - must be 1
 break;
case 70:
 // Polyline flags - optional 0
 break;
case 71:
 // Polygon mesh count ( Faces ) - Do not rely on this
 break;
case 72:
 // Polygon vertex count - Do not rely on this
 break;
case 73:
 // smooth surfaces - optional 0
 break;
case 74:
 // densities - optional 0
 break;
case 75:
 // curves and smooth surface type - optional 0
 break;
default:
 // ReadOk = false;
 // ignore other types
 ReadOk = true;
 break;
 }
}
 }
 __finally
 {
//
// Processamento da lista de vertices
//
if( vList != NULL )
{
//         AddSolido( TCol, vList, PolylineColor );
 AddSolido2( TCol, vList, PolylineColor );
         delete vList;
      }
   }
   return EndPolyline;
}
//---------------------------------------------------------
int __fastcall dxf::ProcessVertexList(
   const FileObject &fp, DxfVertex **Head, int *n )
{
   int ReadOk;
   int EndVertexList;
 int code;
   char value[256];
   char layer[256];
   DxfVertex *v;
 *Head = new DxfVertex;
   v = *Head;
   EndVertexList = false;
   ReadOk = true;
   while( !fp.fend() && ReadOk && !EndVertexList )
   {
      ReadOk = ReadGroup( fp, &code, value );
      switch( code )
      {
         case 0:
            if( !strcmp( value, "VERTEX" ) )
            {
               ReadOk = ReadGroup( fp, &code, layer );
               EndVertexList = ProcessVertexList( fp, &v->tail, n );
               (*n)++;
}
            else if( !strcmp( value, "SEQEND" ) )
            {
               ReadOk = ReadGroup( fp, &code, layer );
               (*n) = 0;
 EndVertexList = true;
            }
            else
            {
               // ReadOk = false;
               // ignore other types
               ReadOk = true;
            }
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
            break;
         case 10:
            v->p.x = atof( value );
            break;
         case 20:
v->p.y = atof( value );
            break;
         case 30:
            v->p.z = atof( value );
break;
         case 70:
            v->flags = (unsigned char)atoi( value );
            break;
         case 71:
            v->v1 = atoi( value );
            break;
         case 72:
            v->v2 = atoi( value );
            break;
         case 73:
v->v3 = atoi( value );
            break;
         case 74:
            v->v4 = atoi( value );
break;
         default:
            // ignore other types
            ReadOk = true;
break;
      }
   }
   return EndVertexList;
}
//-------------------------------------------------------
void __fastcall dxf::AddSolido(
 TCollection *TCol, DxfVertex *vList, int Color )
{
 register int i;
   register int j;
   //
   // Processa Cor
   //
//   int Cor = RGB(
//      ( Color & 0x07 ),
//      ( Color & 0x38 ) >> 3,
//      ( Color & 0xC0 ) >> 6 );
 int Cor = RGB(
( Color & 0x07 ) << 5,
      ( Color & 0x38 ) << 2,
( Color & 0xC0 ) );
   //
 // Conta o numero de vertices e o número de faces
   //
   int nVertices = 0;
   int nFaces = 0;
   DxfVertex *v = vList;
   while( v != NULL )
 {
      if( ( v->flags & 64  ) &&
          ( v->flags & 128 ) )
      {
         nVertices++;
}
else if( !( v->flags & 64  ) &&
                ( v->flags & 128 ) )
      {
         nFaces++;
}
      v = v->tail;
 }
   //
   // Cria os vetores de ponteiros necessarios para o construtor de solidos
   //
   Vertice **v1 = new Vertice *[nVertices];
   Vertice **vTri = new Vertice *[3];
   Aresta **aTri = new Aresta *[3];
   for( i = 0; i < 3; i++ )
 {
      vTri[i] = new Vertice( Zero3D, Cor );
      aTri[i] = new Aresta(
         Vertice( Zero3D, Cor ),
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
        Vertice( Zero3D, Cor ),
 true, Cor );
   }
   Face **f1 = new Face *[nFaces];
   i = 0;
   j = 0;
 v = vList;
   while( v != NULL )
 {
      if( ( v->flags & 64  ) &&
          ( v->flags & 128 ) )
      {
         v1[i] = new Vertice( v->p, Cor );
         i++;
      }
      else if( !( v->flags & 64  ) &&
( v->flags & 128 ) )
      {
         // Prepara os vertices do triangulo
         vTri[0]->p = v1[abs(v->v1)-1]->p;
         vTri[1]->p = v1[abs(v->v2)-1]->p;
 vTri[2]->p = v1[abs(v->v3)-1]->p;
         // Prepara as arestas do triangulo
         aTri[0]->v1      = *vTri[0];
         aTri[0]->v2      = *vTri[1];
         aTri[0]->Visivel = v->v1 > 0;
 aTri[1]->v1      = *vTri[1];
         aTri[1]->v2      = *vTri[2];
 aTri[1]->Visivel = v->v2 > 0;
         aTri[2]->v1      = *vTri[2];
         aTri[2]->v2      = *vTri[0];
         aTri[2]->Visivel = v->v3 > 0;
         f1[j] = new Triangulo( vTri, aTri, 2*EPSILON0, SIGMA0, MI0, Cor );
         j++;
      }
      v = v->tail;
 }
   //
   // Conta e cria as arestas distintas
   //
   register int k;
 register int l;
   int achou;
   // Usa a Relacao de Euler para saber o numero de arestas
//   int nArestas = 0;
   int nArestas = nVertices + nFaces - 2;
 Aresta **a1 = new Aresta *[nArestas];
   k = 0;
 for( i = 0; i < nFaces; i++ )
   {
      for( j = 0; j < 3; j++ )
      {
         l = 0;
         achou = false;
         while( l < k && !achou )
         {
achou = *f1[i]->pAresta[j] == *a1[l];
            l++;
         }
         if( !achou )
         {
a1[k] = new Aresta( *f1[i]->pAresta[j] );
            k++;
         }
      }
   }
 // Erro! Solido nao Euleriano!
   if( k != nArestas ) throw;
   //
   // Cria o solido ( já é automaticamente colocado na colecao )
   //
//   new Solido( TCol, nVertices, v1 );
   new Solido( TCol, nVertices, v1, nArestas, a1, nFaces, f1, Color );
   //
   // Desaloca memoria
   //
 for( i = 0; i < nFaces; i++ )
   {
      delete f1[i];
   }
   delete [] f1;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 for( i = 0; i < nArestas; i++ )
   {
      delete a1[i];
   }
   delete [] a1;
 for( i = 0; i < nVertices; i++ )
   {
      delete v1[i];
   }
   delete [] v1;
   for( i = 0; i < 3; i++ )
   {
delete aTri[i];
 }
 delete [] aTri;
 for( i = 0; i < 3; i++ )
 {
delete vTri[i];
 }
 delete [] vTri;
}
//---------------------------------------------------
void __fastcall dxf::AddSolido2(
 TCollection *TCol, DxfVertex *vList, int Color )
{
 register int i;
 register int j;
 register int k;
   register int l;
   int achou;
   //
   // Processa Cor
 //
 int Cor = RGB(
( Color & 0x07 ) << 5,
( Color & 0x38 ) << 2,
( Color & 0xC0 ) );
 Vertice **vTri = NULL;
   Aresta **aTri = NULL;
 Vertice **v1 = NULL;
   Face **f1 = NULL;
   Face **f2 = NULL;
   Face **f3 = NULL;
   Aresta **a1 = NULL;
 int nVertices;
   int nFaces;
   int nFacesReal;
   int nArestas;
   try
   {
      //
      //
      nVertices = 0;
      nFaces = 0;
      DxfVertex *v = vList;
      while( v != NULL )
{
         if( ( v->flags & 64  ) &&
             ( v->flags & 128 ) )
         {
            nVertices++;
 }
         else if( !( v->flags & 64  ) &&
                    ( v->flags & 128 ) )
         {
            nFaces++;
         }
         v = v->tail;
      }
      //
      //
      v1 = new Vertice *[nVertices];
      vTri = new Vertice *[3];
      aTri = new Aresta *[3];
for( i = 0; i < 3; i++ )
{
         vTri[i] = new Vertice( Zero3D, Cor );
         aTri[i] = new Aresta(
            Vertice( Zero3D, Cor ),
            Vertice( Zero3D, Cor ),
true, Cor );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      }
      f1 = new Face *[nFaces];
      i = 0;
      j = 0;
      v = vList;
      while( v != NULL )
      {
         if( ( v->flags & 64  ) &&
             ( v->flags & 128 ) )
         {
            v1[i] = new Vertice( v->p, Cor );
            i++;
         }
         else if( !( v->flags & 64  ) &&
 ( v->flags & 128 ) )
         {
            // Prepara os vertices do triangulo
            vTri[0]->p = v1[abs(v->v1)-1]->p;
            vTri[1]->p = v1[abs(v->v2)-1]->p;
vTri[2]->p = v1[abs(v->v3)-1]->p;
            // Prepara as arestas do triangulo
            aTri[0]->v1      = *vTri[0];
            aTri[0]->v2      = *vTri[1];
            aTri[0]->Visivel = v->v1 > 0;
            aTri[1]->v1      = *vTri[1];
            aTri[1]->v2      = *vTri[2];
            aTri[1]->Visivel = v->v2 > 0;
            aTri[2]->v1      = *vTri[2];
            aTri[2]->v2      = *vTri[0];
            aTri[2]->Visivel = v->v3 > 0;
            f1[j] = new Triangulo( vTri, aTri, 2*EPSILON0, SIGMA0, MI0, Cor );
            j++;
         }
         v = v->tail;
}
      //
      //
f2 = new Face *[nFaces];
      nFacesReal = 0;
      for( i = 0; i < nFaces; i++ )
      {
         j = 0;
         achou = false;
         while( j < nFacesReal && !achou )
         {
            achou = f1[i]->Coplanar( *f2[j], 0.1 * pi / 180.0 );
            if( !achou )
            {
               j++;
            }
         }
         if( achou )
 {
            f2[j]->MergeTriangulo( *f1[i] );
         }
         else
         {
f2[nFacesReal] = new Face( *f1[i] );
            nFacesReal++;
         }
      }
      //
      //    "Enxuga" f2
      //
      f3 = new Face *[nFacesReal];
for( i = 0; i < nFacesReal; i++ )
      {
         f3[i] = f2[i];
      }
      //
//      int nArestas = 0;
      nArestas = nVertices + nFacesReal - 2;
      a1 = new Aresta *[nArestas];
      k = 0;
for( i = 0; i < nFacesReal; i++ )
      {
         for( j = 0; j < f3[i]->NumArestas; j++ )
         {
            l = 0;
            achou = false;
            while( l < k && !achou )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
            {
               if( l < nArestas )
               {
                  achou = *f3[i]->pAresta[j] == *a1[l];
               }
               l++;
            }
            if( !achou )
{
               if( k < nArestas )
               {
                  a1[k] = new Aresta( *f3[i]->pAresta[j] );
               }
 k++;
            }
         }
      }
      // Erro! Solido nao Euleriano!
      if( k != nArestas )
      {
         Application->MessageBox( "Sólido não Euleriano.",
            "Erro no DXF", MB_OK );
//        throw ESolidoException( "Sólido não Euleriano." );
      }
      //
      // Cria o solido ( já é automaticamente colocado na colecao )
      //
new Solido( TCol, nVertices, v1, nArestas, a1, nFacesReal, f3, Color );
 }
   __finally
   {
      //
      // Desaloca memoria
//
      if( f3 != NULL )
      {
         for( i = 0; i < nFacesReal; i++ )
         {
            delete f3[i];
         }
         delete [] f3;
      }
      if( f2 != NULL )
      {
         delete [] f2;
      }
      if( f1 != NULL )
      {
 for( i = 0; i < nFaces; i++ )
         {
            delete f1[i];
         }
         delete [] f1;
}
      if( a1 != NULL )
      {
         for( i = 0; i < nArestas; i++ )
         {
            delete a1[i];
         }
         delete [] a1;
      }
      if( v1 != NULL )
      {
         for( i = 0; i < nVertices; i++ )
         {
            delete v1[i];
         }
 delete [] v1;
      }
      if( aTri != NULL )
      {
         for( i = 0; i < 3; i++ )
 {
            delete aTri[i];
         }
         delete [] aTri;
      }
      if( vTri != NULL )
      {
 for( i = 0; i < 3; i++ )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
        {
           delete vTri[i];
        }
        delete [] vTri;
     }
  }
}
//----------------------------------------------------

//----------------------------------------------------
#ifndef AntenaH
#define AntenaH
//----------------------------------------------------
#include "Vetor3D.h"
#include "Solido.h"
//----------------------------------------------------
class Antena
{
public:
   const Vetor3D p;
   const Vetor3D ax;
   const Vetor3D ay;
   const Vetor3D az;
   const double GdB;
   const double G;

private:
   Operador3D GTA;
   Operador3D ATG;

public:
   __fastcall Antena(
      const Vetor3D &p1,
      const Vetor3D &ax1,
      const Vetor3D &ay1,
      const Vetor3D &az1,
      double GdB1 );
   __fastcall Antena( const Antena &A );
   virtual __fastcall ~Antena( void );
   Vetor3D __fastcall GlobalToAntena( const Vetor3D &v ) const;
   Vetor3D __fastcall AntenaToGlobal( const Vetor3D &v ) const;
   CVetor3D __fastcall AntenaToGlobal( const CVetor3D &v ) const;
   void __fastcall CartesianToPolar(
      const Vetor3D &v, double *R, double *Theta, double *Phi ) const;
   //
   // Calculo do campo num dado ponto
   //
   CVetor3D __fastcall E( const Vetor3D &p, double k ) const;
   CVetor3D __fastcall EDir( const Vetor3D &p1 ) const;
   //
   //
   Vetor3D __fastcall EAxis(
      double Theta, double Phi,
      const Vetor3D &ax, const Vetor3D &ay, const Vetor3D &az ) const;
   //
   // Metodos puramente virtuais
   //

   //
   //
   virtual Vetor3D __fastcall E0( double Theta, double Phi ) const = 0;
   virtual Antena * __fastcall CopiaAntena( void ) const = 0;
   virtual Antena * __fastcall MirrorAntena( const Face &f ) const = 0;
};

class AntenaIsotropica : public Antena
{
public:
   const Vetor3D DirE0; // sempre na direção z

public:
   __fastcall AntenaIsotropica(
      const Vetor3D &p1,
      const Vetor3D &ax1,
      const Vetor3D &ay1,
      const Vetor3D &az1,
      double GdB1 );
   __fastcall AntenaIsotropica( const AntenaIsotropica &A );
   virtual __fastcall ~AntenaIsotropica( void );
   //
   // Implementação dos Metodos puramente virtuais
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   //
   virtual Vetor3D __fastcall E0( double Theta, double Phi ) const;
   virtual Antena * __fastcall CopiaAntena( void ) const;
   virtual Antena * __fastcall MirrorAntena( const Face &f ) const;
};
//-----------------------------------------------------
#endif

//-----------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop

#include "Antena.h"
#include "Dlg_EnvSettings.h"

//-----------------------------------------------------
#pragma package(smart_init)
//-----------------------------------------------------
static double pi = 4.0 * atan( 1.0 );
static double Epsilon0 = 8.854e-12;
static double Mi0 = 4.0*pi*1e-7;
static double Eta = sqrt( Mi0 / Epsilon0 );
static double Minimo = 1.0e-10;
//-----------------------------------------------------
__fastcall Antena::Antena(
   const Vetor3D &p1,
   const Vetor3D &ax1,
   const Vetor3D &ay1,
   const Vetor3D &az1,
   double GdB1 )
:
p( p1 ),
ax( ax1.Normalizado() ),
ay( ay1.Normalizado() ),
az( az1.Normalizado() ),
GdB( GdB1 ),
G( pow( 10.0, ( GdB1 / 10.0 ) ) )
{
   double MGTA[3][3];

   MGTA[0][0] = ax.x;
   MGTA[0][1] = ax.y;
   MGTA[0][2] = ax.z;
   MGTA[1][0] = ay.x;
   MGTA[1][1] = ay.y;
   MGTA[1][2] = ay.z;
   MGTA[2][0] = az.x;
   MGTA[2][1] = az.y;
   MGTA[2][2] = az.z;

   GTA = Operador3D( MGTA );
   // A inversa é a transposta
   ATG = !GTA;
}
//-----------------------------------------------------
__fastcall Antena::Antena( const Antena &A )
:
p( A.p ),
ax( A.ax ),
ay( A.ay ),
az( A.az ),
GdB( A.GdB ),
G( A.G ),
GTA( A.GTA ),
ATG( A.ATG )
{
}
//-----------------------------------------------------
__fastcall Antena::~Antena( void )
{
}
//-----------------------------------------------------
Vetor3D __fastcall Antena::GlobalToAntena( const Vetor3D &v ) const
{
   Vetor3D ret;

   ret = GTA * v;

   return ret;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
}
//----------------------------------------------------
Vetor3D __fastcall Antena::AntenaToGlobal( const Vetor3D &v ) const
{
   Vetor3D ret;

   ret = ATG * v;

   return ret;
}
//----------------------------------------------------
CVetor3D __fastcall Antena::AntenaToGlobal( const CVetor3D &v ) const
{
   CVetor3D ret;

   ret = (COperador3D)ATG * v;

   return ret;
}
//----------------------------------------------------
void __fastcall Antena::CartesianToPolar(
   const Vetor3D &v, double *R, double *Theta, double *Phi ) const
{
   *R = v.Norma();
   if( fabs( *R ) > Minimo )
   {
      *Theta = acos( v.z / *R );
      if( v.x != 0.0 || v.y != 0 )
      {
         *Phi = atan2( v.y, v.x );
      }
      else
      {
         *Phi = 0.0;
      }
   }
   else
   {
      *Theta = 0.0;
      *Phi = 0.0;
   }
}
//----------------------------------------------------
CVetor3D __fastcall Antena::E( const Vetor3D &p1, double k ) const
{
   CVetor3D ret;
   Vetor3D Eaux;
   complex<double> j( 0.0, 1.0 );
   double R;
   double Theta;
   double Phi;


   Vetor3D DeltaP = p1 - p;
   DeltaP = GlobalToAntena( DeltaP );
   CartesianToPolar( DeltaP, &R, &Theta, &Phi );
 R /= EnvSettings->pxxm;
 Eaux = E0( Theta, Phi );
   Eaux = AntenaToGlobal( Eaux );
   if( fabs( R ) > Minimo )
   {
      ret = Eaux * exp( - j * k * R ) / R;
   }
   else
   {
      ret = Eaux;
   }

   return ret;
}
//----------------------------------------------------
CVetor3D __fastcall Antena::EDir( const Vetor3D &p1 ) const
{
   CVetor3D Eret;
   Vetor3D Eaux;
   double R;
   double Theta;
   double Phi;

   Vetor3D DeltaP = p1 - p;
   DeltaP = GlobalToAntena( DeltaP );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   CartesianToPolar( DeltaP, &R, &Theta, &Phi );
   Eaux = E0( Theta, Phi );
   Eret = AntenaToGlobal( Eaux );

   return Eret;
}
//-----------------------------------------------------
Vetor3D __fastcall Antena::EAxis(
 double Theta, double Phi,
 const Vetor3D &ax, const Vetor3D &ay, const Vetor3D &az ) const
//   complex<double> *ETheta, complex<double> *EPhi ) const
{
 Vetor3D dir1(
sin( Theta ) * cos( Phi ),
sin( Theta ) * sin( Phi ),
cos( Theta )
 );
 Operador3D AxisToGlobal( ax, ay, az );
 Vetor3D dir2( AxisToGlobal * dir1 );
 Vetor3D dir3( GlobalToAntena( dir2 ) );
 double RA, ThetaA, PhiA;
 CartesianToPolar( dir3, &RA, &ThetaA, &PhiA );
 Vetor3D E1 = E0( ThetaA, PhiA );
 Vetor3D E2 = AntenaToGlobal( E1 );
 Operador3D GlobalToAxis( !AxisToGlobal );
 Vetor3D E3 = GlobalToAxis * E2;
 /*
 Vetor3D uThetaAxis(
cos( Theta ) * cos( Phi ),
cos( Theta ) * sin( Phi ),
-sin( Theta )
 );
 Vetor3D uPhiAxis(
-sin( Phi ),
cos( Phi ),
      0.0
   );
   *ETheta = uThetaAxis * E3;
   *EPhi   = uPhiAxis   * E3;
   */

   return E3;
}
//-----------------------------------------------------
//-----------------------------------------------------
//-----------------------------------------------------
__fastcall AntenaIsotropica::AntenaIsotropica(
   const Vetor3D &p1,
   const Vetor3D &ax1,
   const Vetor3D &ay1,
   const Vetor3D &az1,
   double GdB1 )
:
Antena( p1, ax1, ay1, az1, GdB1 ),
DirE0( 0.0, 0.0, 1.0 )
//DirE0( Dir.Normalizado() )
{
   //
   //    Antena isotrópica: E0 sempre na direção z da antena, e o ganho não
   // depende de Phi
   //
}
//-----------------------------------------------------
__fastcall AntenaIsotropica::AntenaIsotropica( const AntenaIsotropica &A )
:
Antena( A ),
DirE0( A.DirE0 )
{
}
//-----------------------------------------------------
__fastcall AntenaIsotropica::~AntenaIsotropica( void )
{
}
//-----------------------------------------------------
Vetor3D __fastcall AntenaIsotropica::E0( double Theta, double Phi ) const
{
   Vetor3D ret;

   //
   // usar a direção de E0 junto com a direção de propagação para
   // definir o plano de E.
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   //
   Vetor3D ur(
      sin( Theta ) * cos( Phi ),
      sin( Theta ) * sin( Phi ),
      cos( Theta ) );
 Vetor3D uaux = ( ur % DirE0 ).Normalizado();
   Vetor3D DirE = uaux % ur;
   ret = DirE * sin( Theta ) * sqrt( Eta * G / ( 2.0 * pi ) );

   return ret;
}
//-------------------------------------------------------
Antena * __fastcall AntenaIsotropica::CopiaAntena( void ) const
{
   Antena *ret;

   ret = new AntenaIsotropica( *this );

   return ret;
}
//-------------------------------------------------------
Antena * __fastcall AntenaIsotropica::MirrorAntena( const Face &f ) const
{
   double Distancia = 2.0 * ( f.D + f.Normal * p );
   Vetor3D posImg = p - f.Normal * Distancia;

   Operador3D Mirror = Identidade() - 2.0 * Projetor( f.Normal );

   Vetor3D ax1 = Mirror * ax;
   Vetor3D ay1 = Mirror * ay;
   Vetor3D az1 = Mirror * az;

   return new AntenaIsotropica( posImg, ax1, ay1, az1, GdB );
}
//-------------------------------------------------------


//--------------------------------------------------------
#ifndef AboutOGLH
#define AboutOGLH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
//--------------------------------------------------------
#include <gl\gl.h>
//--------------------------------------------------------
class TfrmAboutOGL : public TForm
{
__published:
TButton *OKButton;
   TGroupBox *GroupBox1;
   TLabel *staRenderer;
   TLabel *staVendor;
   TLabel *staVersion;
   TLabel *staExtensions;
   TGroupBox *GroupBox2;
   TGroupBox *GroupBox3;
   TLabel *staExtensions2;
   TLabel *staVersion2;
   TLabel *lblVendor;
   TLabel *lblRenderer;
   TLabel *lblVersion;
   TLabel *lblExtensions;
   TLabel *lblVersion2;
   TLabel *lblExtensions2;
   TLabel *lblErrors;
   void __fastcall FormShow(TObject *Sender);
private:
   GLenum glError;

public:
virtual __fastcall TfrmAboutOGL(TComponent* AOwner);
};
//--------------------------------------------------------
extern PACKAGE TfrmAboutOGL *frmAboutOGL;
//--------------------------------------------------------
#endif

//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
#include <stdlib.h>
#pragma hdrstop

#include "AboutOGL.h"

#include <gl\glu.h>

//---------------------------------------------------------------------
#pragma resource "*.dfm"
TfrmAboutOGL *frmAboutOGL;
//---------------------------------------------------------------------
__fastcall TfrmAboutOGL::TfrmAboutOGL(TComponent* AOwner)
: TForm(AOwner)
{
}
//---------------------------------------------------------------------
void __fastcall TfrmAboutOGL::FormShow(TObject *Sender)
{
   lblVendor->Caption      = (char *)glGetString( GL_VENDOR );
   lblRenderer->Caption    = (char *)glGetString( GL_RENDERER );
   lblVersion->Caption     = (char *)glGetString( GL_VERSION );
   lblExtensions->Caption  = (char *)glGetString( GL_EXTENSIONS );
   lblVersion2->Caption    = (char *)gluGetString( GLU_VERSION );
   lblExtensions2->Caption = (char *)gluGetString( GLU_EXTENSIONS );
   lblErrors->Caption = "";
   register int i = 0;
   do
   {
      glError = glGetError();
      lblErrors->Caption =
         lblErrors->Caption + (char *)gluErrorString( glError );
      i++;
   }
   while( i < 6 && glError != GL_NO_ERROR );
}
//-----------------------------------------------------

//------------------------------------------------------
#ifndef fNovaFonteH
#define fNovaFonteH
#include <Classes.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
#include <StdCtrls.hpp>
//------------------------------------------------------
class TdlgNovaFonte : public TForm
{
__published:
TButton *OKBtn;
TButton *CancelBtn;
TBevel *Bevel1;
   TLabel *Label1;
   TEdit *edtX;
   TLabel *Label2;
   TEdit *edtY;
   TLabel *Label3;
   TEdit *edtZ;
   TLabel *Label4;
   TEdit *edtF;
   TLabel *Label5;
   TEdit *edtL;
   TLabel *Label6;
private:
public:
virtual __fastcall TdlgNovaFonte(TComponent* AOwner);
   int __fastcall GetNovaFonteCoord(
      double *x, double *y, double *z, double *f, int *l );
};
//------------------------------------------------------
extern PACKAGE TdlgNovaFonte *dlgNovaFonte;
//------------------------------------------------------
#endif

//---------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop

#include "fNovaFonte.h"
//---------------------------------------------------------------------
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
#pragma resource "*.dfm"
TdlgNovaFonte *dlgNovaFonte;
//---------------------------------------------------------------------
__fastcall TdlgNovaFonte::TdlgNovaFonte(TComponent* AOwner)
: TForm(AOwner)
{
}
//---------------------------------------------------------------------
int __fastcall TdlgNovaFonte::GetNovaFonteCoord(
   double *x, double *y, double *z, double *f, int *l )
{
   int ret;

   ret = ShowModal();
   if( ret == mrOk )
   {
      *x = atof( edtX->Text.c_str() );
      *y = atof( edtY->Text.c_str() );
      *z = atof( edtZ->Text.c_str() );
      *f = atof( edtF->Text.c_str() ) * 1.0e9;
      *l = atoi( edtL->Text.c_str() );
   }
   ret = ret == mrOk;

   return ret;
}
//---------------------------------------------------------------------


//---------------------------------------------------------
#ifndef FonteH
#define FonteH
//---------------------------------------------------------
#include <gl\gl.h>
#include <gl\glu.h>
//---------------------------------------------------------
#include "Antena.h"
#include "Vetor3D.h"
#include "Solido.h"
#include "Tree.h"
//---------------------------------------------------------
static int count; // numero de fontes instanciadas


class Fonte
{
public:
 const Antena *Ant;
 const double Frequencia;
 const double Potencia;
 const Face *f;            // Se f == NULL é porque não é imagem
 const Vetor3D p;          // Posição da fonte
 const double k;           // Numero de onda

private:
 static GLUquadricObj *gluObj;

public:
 bool __fastcall RaioObstruido( const Vetor3D &p1, const Vetor3D &p2,
 const Face *face) const;
 static GLUquadricObj *Fonte::CreateSphere( void );
 __fastcall Fonte( const Antena *Ant1, double Freq, double Pot,
const Face *f1 );
 __fastcall Fonte( const Fonte &F );
 __fastcall ~Fonte( void );
 bool InsideReflectionSpace(
 const Vetor3D &p1, Vetor3D *ReflectionPoint, const Face *face) const;
 Fonte * __fastcall NewImagem( const Face &f ) const;
 double __fastcall AnguloDeIncidencia( const Vetor3D &ur ) const;
 void __fastcall FacetFixedAxis(
const Vetor3D &ur, Vetor3D *xf, Vetor3D *yf, Vetor3D *zf ) const;
 void __fastcall Fresnel(
double *Theta,
complex<double> *GamaS, complex<double> *GamaH,
Vetor3D *uh, Vetor3D *us, const Vetor3D &ur ) const;
 Vetor3D __fastcall Fonte::EAxis( double Theta, double Phi,
const Vetor3D &ax, const Vetor3D &ay, const Vetor3D &az ) const;
 void __fastcall Draw( double R ) const;
};

class ArvoreFonte : public TreeNode<Fonte>
{
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
private:
ArvoreFonte *rfont;

public:
 __fastcall ArvoreFonte(
const Fonte &N, TreeNode<Fonte> *F, TreeNode<Fonte> *B, int L,
SolidoCollection &colSolido, int LMax );
 __fastcall ~ArvoreFonte( void );
 int __fastcall ContaNo( void ) const;
 bool __fastcall SetThisFont(void);
 CVetor3D __fastcall EAxis(
double Theta, double Phi,
const Vetor3D &ax, const Vetor3D &ay, const Vetor3D &az ) const;
 CVetor3D __fastcall CampoFonteAtual( const Vetor3D &p1 ) const;
 bool __fastcall InsideReflectionSpace( const Vetor3D &p1) const;
 CVetor3D __fastcall CampoEm( const Vetor3D &p1 ) const;
 void __fastcall DrawFonte( double R ) const;
};

//-----------------------------------------------------
#endif


//-----------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include "Fonte.h"
#include "Dlg_EnvSettings.h"

#pragma hdrstop
#pragma package(smart_init)


static double pi = 4.0 * atan( 1.0 );
static double c = 2.997925e8; // ( m/s )
static double Epsilon0 = 8.854e-12;
static double Minimo = 1.0e-10;

__fastcall Fonte::Fonte(
 const Antena *Ant1, double Freq, double Pot, const Face *f1 )
:
Ant( Ant1->CopiaAntena() ),
Frequencia( Freq ),
Potencia( Pot ),
f( f1 ),
p( Ant1->p ),
k( 2.0 * pi * Freq / c )
{
}
//-----------------------------------------------------
__fastcall Fonte::Fonte( const Fonte &F )
:
Ant( F.Ant->CopiaAntena() ),
Frequencia( F.Frequencia ),
Potencia( F.Potencia ),
f( F.f ),
p( F.p ),
k( F.k )
{
count = 0;
}
//-----------------------------------------------------
__fastcall Fonte::~Fonte( void )
{
 delete Ant;
}
//-----------------------------------------------------
bool Fonte::InsideReflectionSpace(
 const Vetor3D &p1, Vetor3D *ReflectionPoint, const Face *face ) const
{
 bool ret;
 if(face)
 {
Vetor3D dir( p1 - p );
double dt = dir.Norma();
double Distancia;

double delta = face->Normal * dir;
if( fabs( delta ) > 1.0e-10 )
{
 Distancia = - ( face->D + f->Normal * p ) / delta;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
}
else
{
 Distancia = 0.0;
}

Vetor3D Intersection = p + dir * Distancia;
*ReflectionPoint = Intersection;
        ret = face->PertenceAFace( Intersection )
        & !RaioObstruido(Intersection, p1, face);
 }
 else
 {
*ReflectionPoint = p1;
ret = !RaioObstruido(p, p1, NULL);
 }
 return ret;
}
//----------------------------------------------------
Fonte * __fastcall Fonte::NewImagem( const Face &f ) const
{
 Antena *pAnt = Ant->MirrorAntena( f );
 Fonte *Img = new Fonte( pAnt, Frequencia, Potencia, &f );
 delete pAnt;

 return Img;
}
//-------------------------------------------------------
//
// Calcula o angulo de incidência
//
double __fastcall Fonte::AnguloDeIncidencia( const
                  Vetor3D &ur ) const
{
 //
 // Angulo de incidência/Reflexão
 //
 double ThetaI = acos( ur * f->Normal );
   if( ThetaI > pi / 2 )
   {
      ThetaI = pi - ThetaI;
   }
   return ThetaI;
}
//----------------------------------------------------
void __fastcall Fonte::FacetFixedAxis(
   const Vetor3D &ur, Vetor3D *xf, Vetor3D *yf,
             Vetor3D *zf ) const
{
 *zf = f->Normal;
 *yf =  ur % *zf;
 *xf = *yf % *zf;
}
//--------------------------------------------------
void __fastcall Fonte::Fresnel(
 double *ThetaI,
 complex<double> *GamaH,
     complex<double> *GamaS,
 Vetor3D *uh, Vetor3D *us,
     const Vetor3D &ur ) const
{
 //
 // Calcula o angulo de incidência/reflexão
 //
 *ThetaI = AnguloDeIncidencia( ur );
 //
 //   Direção "Soft" == ur x Normal == ur x zf == yf
 //   Direção "Hard" == us x ur
 //
 *us =  ur % f->Normal;
 *uh = *us % ur;
 //
 // Coeficientes de Fresnel
 //
 complex<double> j( 0.0, 1.0 );
 double sinT = sin( *ThetaI );
 double cosT = cos( *ThetaI );
 double Omega = c * k;
 complex<double> EpsilonR =
( f->Epsilon - j * f->Sigma / Omega ) / Epsilon0;
 complex<double> aux1 = sqrt( EpsilonR - sinT * sinT );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 complex<double> aux2 = EpsilonR * cosT;
 *GamaH = ( aux2 - aux1 ) / ( aux2 + aux1 );
 *GamaS = ( cosT - aux1 ) / ( cosT + aux1 );
}
//---------------------------------------------------------
Vetor3D __fastcall Fonte::EAxis(
 double Theta, double Phi,
 const Vetor3D &ax, const Vetor3D &ay,
 const Vetor3D &az ) const
{
 Vetor3D E = Ant->EAxis( Theta, Phi, ax, ay, az );
 double sqrtPot = sqrt( Potencia );
 E *= sqrtPot;

 return E;
}
//------------------------------------------------------
void __fastcall Fonte::Draw( double R ) const
{
 //glEnable(GL_LOGIC_OP);
 //glLogicOp(GL_XOR);

 glPointSize(4.0f);
 if(!(count++))
glColor3f(1.0f, 0.0f, 0.0f);
 else
glColor3f( 1.0f, 1.0f, 0.0f );

 glBegin(GL_POINTS);
 glVertex3f(p.x, p.y, p.z);
 glEnd();

 //glDisable(GL_LOGIC_OP);
}
//-----------------------------------------------
//
//
__fastcall ArvoreFonte::ArvoreFonte(
const Fonte &N, TreeNode<Fonte> *F, TreeNode<Fonte> *B, int L,
SolidoCollection &colSolido, int LMax )
:
TreeNode<Fonte>( N, F, B, L )
{
 register int i;
 register int j;
 Solido *s1;
 Face *f1;
 TreeNode<Fonte> *LastSon;

 if( Level < LMax )
 {
if( colSolido.Count != 0 )
{
 LastSon = NULL;
 for( i = 0; i < colSolido.Count; i++ )
 {
s1 = (Solido *)colSolido.Items[i];
for( j = 0; j < s1->NumFaces; j++ )
{
 f1 = s1->pFace[j];
 if( Data.f == NULL )
 {
goto NaoPrecisaTestar;
 }
 if( *f1 != *Data.f )
 {
//
if( !f1->BackFace( Data.p ) )
{
     Fonte *Imagem = Data.NewImagem( *f1 );
 if( LastSon == NULL )
 {
 Son = new ArvoreFonte(
 *Imagem, this, NULL, Level, colSolido, LMax );
 LastSon = Son;
 }
     else
 {
LastSon->NextBrother = new ArvoreFonte(
    *Imagem, this, LastSon, Level, colSolido, LMax );
LastSon = LastSon->NextBrother;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
     }
      delete Imagem;
}
 }
           }
        }
      }
 }
}
//------------------------------------------------
__fastcall ArvoreFonte::~ArvoreFonte( void )
{
}
//----------------------------------------------------
int __fastcall ArvoreFonte::ContaNo( void ) const
{
   int ret;

   ret = 1;
   if( Son != NULL )
 {
      ret += ((ArvoreFonte *)Son)->ContaNo();
   }
   if( NextBrother != NULL )
   {
ret += ((ArvoreFonte *)NextBrother)->ContaNo();
   }

   return ret;
}
//------------------------------------------------------
CVetor3D __fastcall ArvoreFonte::EAxis(
 double Theta, double Phi,
 const Vetor3D &ax, const Vetor3D
     &ay, const Vetor3D &az ) const
{
 CVetor3D E;

 if( Father == NULL )
 {
E = Data.EAxis( Theta, Phi, ax, ay, az );
 }
 else
 {
//
//
CVetor3D EI =
((ArvoreFonte *)Father)->EAxis( pi - Theta, Phi, ax, ay, az );
//
// Coeficientes de Fresnel e direções hard/soft
//
Vetor3D uh, us, ur;
complex<double> GamaH, GamaS;
ur.x = sin( Theta ) * cos( Phi );
ur.y = sin( Theta ) * sin( Phi );
ur.z = cos( Theta );
double Theta1; // Deve voltar igual a Theta
Data.Fresnel( &Theta1, &GamaH, &GamaS, &uh, &us, ur );
complex<double> EIH = CVetor3D( uh ) * EI;
complex<double> EIS = CVetor3D( us ) * EI;
//
complex<double> ERH = GamaH * EIH;
complex<double> ERS = GamaS * EIS;
E = - ERH * uh - ERS * us;
E = EI;
 }
 return E;
}
//---------------------------------------------------
CVetor3D __fastcall ArvoreFonte::CampoFonteAtual( const
        Vetor3D &p1 ) const
{
 CVetor3D Ep1;
 double RI;
 double ThetaI;
 double PhiI;


 if( Data.f == NULL )
 {
Vetor3D p1Fonte( p1 - Data.p );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
Vetor3D ex( 1.0, 0.0, 0.0 );
Vetor3D ey( 0.0, 1.0, 0.0 );
Vetor3D ez( 0.0, 0.0, 1.0 );
CartesianToPolarAxis( p1Fonte, ex, ey, ez,
                      &RI, &ThetaI, &PhiI );
Ep1 = Data.EAxis( ThetaI, PhiI, ex, ey, ez );
 }
 else
 {
Vetor3D p1Fonte( p1 - Data.p );
Vetor3D ur( p1Fonte.Normalizado() );
Vetor3D xf, yf, zf;
Data.FacetFixedAxis( ur, &xf, &yf, &zf );
CartesianToPolarAxis( p1Fonte, xf, yf, zf,
      &RI, &ThetaI, &PhiI );
Ep1 = EAxis( ThetaI, PhiI, xf, yf, zf );
 }
 RI /= EnvSettings->pxxm;

 complex<double> j( 0.0, 1.0 );
 if( RI > Minimo )
 {
Ep1 = Ep1 * exp( - j * Data.k * RI ) / RI;
 }
 return Ep1;
}
//---------------------------------------
bool __fastcall ArvoreFonte::SetThisFont(void)
{
rfont = this;
}

bool __fastcall ArvoreFonte::InsideReflectionSpace(
 const Vetor3D &p1) const
{
bool ret;
Vetor3D r1;
ArvoreFonte *rf = rfont;
int l = Level-1;

while(l-- > 0)
{
if(!rf) {ShowMessage( "Bug de Arvore");return false;}
rf = (ArvoreFonte *)rf->Father;
}
if(!rf) {ShowMessage( "Bug de Arvore");return false;}
ret = Data.InsideReflectionSpace( p1, &r1, rf->Data.f);
if (!ret) return false;

if(Father)
{
ret = ((ArvoreFonte *)Father)->
        InsideReflectionSpace(r1);
}

return ret;
}

bool __fastcall Fonte::RaioObstruido( const Vetor3D
        &p1, const Vetor3D &p2,
         const Face *face) const
{
register int i;
register int j;
Solido *s;
Face *f;
Vertice *v;
Vetor3D pt;

if( colSolido != NULL )
{
for( i = 0; i < colSolido->Count; i++ )
{
s = (Solido *)colSolido->Items[i];
for( j = 0; j < s->NumFaces; j++ )
{
f = s->pFace[j];
if(f == face) continue;
if(!f->Intersection(pt, p1, p2)) continue;
if(f->PertenceAFace( pt )) return true;
}
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
}
}
return false;
}
//------------------------------------------------------
CVetor3D __fastcall ArvoreFonte::CampoEm( const
         Vetor3D &p1 ) const
{
 CVetor3D Ep1; // O Construtor já inicializa para zero
 double dEp1;


 SetThisFont();
 if(InsideReflectionSpace(p1))
 {
//Ep1 = CampoFonteAtual(p1);
dEp1 = CampoFonteAtual(p1).Norma();
Ep1 = CVetor3D(Vetor3D(dEp1,0,0));
 }
 if(Son)
 {
Ep1 += ((ArvoreFonte *)Son)->CampoEm(p1);
 }
 if(NextBrother)
 {
Ep1 += ((ArvoreFonte *)NextBrother)->CampoEm(p1);
 }

return Ep1;
}
//------------------------------------------------------
void __fastcall ArvoreFonte::DrawFonte( double R ) const
{
 Data.Draw( R );
   if( Son != NULL )
   {
((ArvoreFonte *)Son)->DrawFonte( R );
   }
   if( NextBrother != NULL )
   {
      ((ArvoreFonte *)NextBrother)->DrawFonte( R );
 }
}
//---------------------------------------------------------

//---------------------------------------------------------
#ifndef fViewdxfH
#define fViewdxfH
#include <Classes.hpp>
#include <ComCtrls.hpp>
#include <Controls.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>
#include <StdCtrls.hpp>
#include <ExtCtrls.hpp>
//----------------------------------------------------------
#include <gl\gl.h>
#include "Fonte.h"
#include "Tree.h"
#include "Vetor3D.h"
//----------------------------------------------------------


class TfrmTracado : public TForm
{
__published: // IDE-managed Components
   TOpenDialog *OpenDialog1;
   TMainMenu *MainMenu1;
   TMenuItem *MenuArquivo;
   TMenuItem *MenuArquivoAbrir;
   TMenuItem *MenuAjuda;
   TMenuItem *MenuAjudaOpenGL;
   TScrollBar *sbLongitude;
 TScrollBar *sbLatitude;
   TStatusBar *staStatus;
   TButton *btnReset;
   TMenuItem *MenuOpcoes;
   TMenuItem *MenuOpcoesDepthTest;
   TMenuItem *MenuOpcoesCullFace;
   TMenuItem *MenuOpcoesBackFill;
   TMenuItem *MenuOpcoesPerspectiveView;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   TScrollBar *sbZoom;
   TMenuItem *MenuOpcoesFrontFill;
   TMenuItem *MenuOpcoesLighting;
   TMenuItem *MenuFonte;
   TMenuItem *MenuFonteNova;
   TMenuItem *MenuArquivoSair;
TMenuItem *Resultados1;
TMenuItem *CampodeIntensidades1;
TMenuItem *N1;
TMenuItem *Ambiente1;
 void __fastcall MenuArquivoAbrirClick(TObject *Sender);
 void __fastcall FormCreate(TObject *Sender);
 void __fastcall FormDestroy(TObject *Sender);
 void __fastcall MenuAjudaOpenGLClick(TObject *Sender);
 void __fastcall FormResize(TObject *Sender);
 void __fastcall FormPaint(TObject *Sender);
 void __fastcall FormKeyDown(TObject *Sender, WORD &Key,
TShiftState Shift);
 void __fastcall sbLongitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos);
 void __fastcall sbLatitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos);
 void __fastcall sbZoomScroll(TObject *Sender, TScrollCode ScrollCode,
int &ScrollPos);
 void __fastcall btnResetClick(TObject *Sender);
 void __fastcall MenuOpcoesDepthTestClick(TObject *Sender);
 void __fastcall MenuOpcoesCullFaceClick(TObject *Sender);
 void __fastcall MenuOpcoesBackFillClick(TObject *Sender);
 void __fastcall MenuOpcoesPerspectiveViewClick(TObject *Sender);
 void __fastcall MenuOpcoesFrontFillClick(TObject *Sender);
 void __fastcall MenuOpcoesLightingClick(TObject *Sender);
 void __fastcall MenuFonteNovaClick(TObject *Sender);
 void __fastcall MenuArquivoSairClick(TObject *Sender);
 void __fastcall CampodeIntensidades1Click(TObject *Sender);
 void __fastcall Ambiente1Click(TObject *Sender);

private:
 int LeftOffset;
 int TopOffset;
 int BottomOffset;
   int NewClientWidth;
   int NewClientHeight;
   HWND hWnd;
 HDC hDC;
 int PixelFormat;
 GLfloat glPointSizes[2];
 GLfloat glPointStep;
 GLfloat glLineSizes[2];
 GLfloat glLineStep;

 GLuint SolidsList;
 GLuint SourceList;
 GLuint IFieldPlaneList;

 double R;
 GLfloat Phi;
 GLfloat Theta;
 GLfloat Scale;
   GLfloat xObs;
   GLfloat yObs;
   GLfloat zObs;

   int LMax;
 Fonte *pFonte;
 ArvoreFonte *pFonteTree;

 void __fastcall _GetDC(void);
 void __fastcall _ReleaseDC(void);
 void FillOutIFieldBM(void);
 void RedrawIField(void);
 class TBitmap *IFieldBM;
 GLbyte *bits;
 int oldwidth;
 int oldheight;
 GLuint IFieldTexture;
 double fonx,fony,fonz; //posiccao da fonte real

public: // User declarations
 __fastcall TfrmTracado(TComponent* Owner);
 __fastcall ~TfrmTracado( void );
 void __fastcall CampodeIntensidades(void);
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 void __fastcall GeraImagens(void);
 void __fastcall SetPixelFormatDescriptor( void );
 void __fastcall SetupRC( void );
   void __fastcall SetupLighting( void );
   void __fastcall SetupTextures( void );
   void __fastcall CreateObjects( void );
   void __fastcall DestroyObjects( void );
   void __fastcall CalcNormal(
      GLfloat *p1, GLfloat *p2, GLfloat *p3,
      GLfloat *Normal );
   void __fastcall DrawObjects( void );
 void __fastcall SetupLights( void );
   void __fastcall RenderScene( void );
   void __fastcall ChangeSize( GLsizei x,
   GLsizei y, GLsizei w, GLsizei h );
   void __fastcall MoveObserver( void );
   void __fastcall ResetObserver( void );

 HGLRC hRC;
};
//----------------------------------------
extern PACKAGE TfrmTracado *frmTracado;
//----------------------------------------
#endif

//-----------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop

#include "fViewdxf.h"
#include "TIFieldAtt.h"
#include "TIFIeld.h"
#include "Dlg_EnvSettings.h"

#include <math.h>
#include <gl/glu.h>

#include "SolidoDxf.h"
#include "AboutOGL.h"
#include "DXF.h"
#include "fNovaFonte.h"
#include "Fonte.h"
//-----------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmTracado *frmTracado;

//----------------------------------------------
static const double PI = 4.0 * atan( 1.0 );
static const double G2R = PI / 180.0;
//----------------------------------------------

void SpectralColor(float Int,byte &r,byte &g,byte &b)
{
// linear scale
if (Int < 0.)
 { r = 0; g = 0; b = 255; }
else if (Int < 0.25)
 { r = 0; g = Int*4.*255; b = 255; }
else if (Int < 0.5)
 { r = 0; g = 255; b = 255-(Int-0.25)*4.*255; }
else if (Int < 0.75)
 { r = (Int-0.5)*4.*255; g = 255; b = 0; }
else if (Int < 1.)
 { r = 255; g = 255-(Int-0.75)*4*255; b = 0; }
else
 { r = 255; g = 0; b = 0; }
}


__fastcall TfrmTracado::TfrmTracado(TComponent* Owner)
 : TForm(Owner)
{
 hRC = NULL;
 colSolido = NULL;
 LeftOffset = sbZoom->Width;
 TopOffset = 0;
 BottomOffset = sbLongitude->Height;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 sbZoom->Left = 0;
 sbZoom->Top = 0;
   sbLongitude->Left = LeftOffset;
   sbLatitude->Top = TopOffset;

 xObs = 0.0;
   yObs = 0.0;
   zObs = 0.0;
   Theta = 0.0;
 Phi = 0.0;
 R = 1.0;
 Scale = 1.0;

 pFonte = NULL;
 pFonteTree = NULL;

 SolidsList = 0;
 SourceList = 0;
 IFieldPlaneList = 0;

 IFieldBM = NULL;
 bits = NULL;
}
//---------------------------------------------
__fastcall TfrmTracado::~TfrmTracado( void )
{
 if( colSolido != NULL )
   {
      delete colSolido;
 }
   if( pFonte != NULL )
   {
      delete pFonte;
 }
   if( pFonteTree != NULL )
   {
delete pFonteTree;
 }
 delete IFieldBM;
 delete bits;

}

void __fastcall TfrmTracado::_GetDC(void)
{
 if(hRC) return;
 hWnd = Handle;
 hDC = GetDC( hWnd );

 SetPixelFormatDescriptor();
 hRC = wglCreateContext( hDC );
 if(!hRC)
 {
ShowMessage( "hrc = NULL" );
 }
 if(wglMakeCurrent(hDC, hRC)==false)
 {
ShowMessage( "Could not MakeCurrent" );
 }
}

void __fastcall TfrmTracado::_ReleaseDC(void)
{
 ReleaseDC( hWnd, hDC );
}
//-----------------------------------------------------
void __fastcall TfrmTracado::FormCreate(TObject *Sender)
{
 _GetDC();

 SetupRC();
 SetupLighting();
 SetupTextures();
 CreateObjects();
}
//--------------------------------------------------
void __fastcall TfrmTracado::FormDestroy(TObject *Sender)
{
 DestroyObjects();
 wglMakeCurrent( NULL, NULL );
 wglDeleteContext( hRC );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 _ReleaseDC();
}
//-------------------------------------------------
void __fastcall TfrmTracado::FormResize(TObject *Sender)
{
 NewClientWidth = ClientWidth
- sbLatitude->Width - LeftOffset;
 NewClientHeight = ClientHeight
- staStatus->Height - sbLongitude->Height - TopOffset;
 sbZoom->Height = NewClientHeight;
 sbLongitude->Top = TopOffset + NewClientHeight;
 sbLongitude->Width = NewClientWidth;
 sbLatitude->Left = LeftOffset + NewClientWidth;
 sbLatitude->Height = NewClientHeight;
 btnReset->Left = LeftOffset + NewClientWidth;
 btnReset->Top = TopOffset + NewClientHeight;
 ChangeSize( LeftOffset, BottomOffset,
              NewClientWidth, NewClientHeight );
}
//---------------------------------------------------
void __fastcall TfrmTracado::FormPaint(TObject *Sender)
{
 //_ReleaseDC();
 //_GetDC();

 RenderScene();
 SwapBuffers( hDC );
}
//---------------------------------------------------
void __fastcall TfrmTracado::FormKeyDown(TObject *Sender,
        WORD &Key, TShiftState Shift)
{


 switch( Key )
 {

case VK_HOME:
 break;

case VK_END:
 break;

case VK_UP:
 break;

case VK_DOWN:
 break;

case VK_LEFT:
 break;

case VK_RIGHT:
 break;

case VK_PRIOR:
 break;

case VK_NEXT:
 break;
 }
}
//-----------------------------------------
void __fastcall TfrmTracado::
SetPixelFormatDescriptor( void )
{
 PIXELFORMATDESCRIPTOR pfd = {
sizeof(PIXELFORMATDESCRIPTOR),
1,
PFD_DRAW_TO_WINDOW |
PFD_SUPPORT_OPENGL |
PFD_DOUBLEBUFFER,
PFD_TYPE_RGBA,
24,
0, 0, 0, 0, 0, 0,
0, 0,
0, 0, 0, 0, 0,
32,
0,
0,
PFD_MAIN_PLANE,
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
0,
      0, 0, 0
   };
 PixelFormat = ChoosePixelFormat( hDC, &pfd );
 SetPixelFormat( hDC, PixelFormat, &pfd );
}
//--------------------------------------------
void __fastcall TfrmTracado::SetupRC( void )
{
 glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
 glColor3f( 0.0f, 0.70f, 0.0f );
 //
 // Point and line parameters
 //
 glGetFloatv( GL_POINT_SIZE_RANGE, glPointSizes );
 glGetFloatv( GL_POINT_SIZE_GRANULARITY, &glPointStep );
 glGetFloatv( GL_LINE_WIDTH_RANGE, glLineSizes );
 glGetFloatv( GL_LINE_WIDTH_GRANULARITY, &glLineStep );
 //
 glFrontFace( GL_CCW );
 //
 // Shadding model
 //
 glShadeModel( GL_SMOOTH );
 //  glShadeModel( GL_FLAT );
 //
 // Diversos ( Opcoes default do Menu )
 //
 glEnable( GL_POINT_SMOOTH );
 glEnable( GL_DEPTH_TEST );
 glEnable( GL_CULL_FACE );
 glPolygonMode( GL_FRONT, GL_FILL );
 glPolygonMode( GL_BACK, GL_LINE );
 glEnable( GL_LIGHTING );
}
//-----------------------------------------------------
void __fastcall TfrmTracado::SetupLighting( void )
{
 if( MenuOpcoesLighting->Checked )
 {
GLfloat GlobalAmbientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
glLightModeli( GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE );
glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE );
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, GlobalAmbientLight );
GLfloat LightAmbient[]  = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat LightDiffuse[]  = { 1.0f, 1.0f, 0.5f, 1.0f };
GLfloat LightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glLightfv( GL_LIGHT0, GL_AMBIENT, LightAmbient );
glLightfv( GL_LIGHT0, GL_DIFFUSE, LightDiffuse );
glLightfv( GL_LIGHT1, GL_AMBIENT, LightAmbient );
glLightfv( GL_LIGHT1, GL_DIFFUSE, LightDiffuse );

glEnable( GL_COLOR_MATERIAL );
glColorMaterial( GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE );
GLfloat ambientRef[] = { .5f, .5f, .5f, 1.0f };
GLfloat difuseRef[] =  { .7f, .7f, .7f, .7f };

glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT, ambientRef );
glMaterialfv( GL_FRONT_AND_BACK, GL_DIFFUSE, difuseRef );
 }
}
//----------------------------------------------------------
void __fastcall TfrmTracado::SetupTextures( void )
{
}
//----------------------------------------------------------
void __fastcall TfrmTracado::CreateObjects( void )
{
}
//----------------------------------------------------------
void __fastcall TfrmTracado::DestroyObjects( void )
{

}
//----------------------------------------------------------
void __fastcall TfrmTracado::CalcNormal(
 GLfloat *p1, GLfloat *p2, GLfloat *p3, GLfloat *Normal )
{
 GLfloat v1[3];
 GLfloat v2[3];
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 v1[0] = p2[0] - p1[0];
 v1[1] = p2[1] - p1[1];
 v1[2] = p2[2] - p1[2];
 v2[0] = p3[0] - p2[0];
 v2[1] = p3[1] - p2[1];
 v2[2] = p3[2] - p2[2];
 Normal[0] = v1[1]*v2[2] - v1[2]*v2[1];
 Normal[1] = v1[2]*v2[0] - v1[0]*v2[2];
 Normal[2] = v1[0]*v2[1] - v1[1]*v2[0];
 double Norma = sqrt(
Normal[0] * Normal[0] + Normal[1] * Normal[1] +
Normal[2] * Normal[2] );
 Normal[0] /= Norma;
 Normal[1] /= Norma;
 Normal[2] /= Norma;
}

//---------------------------------------------------------
void AdjustColors(GLfloat &r, GLfloat &g, GLfloat &b)
{
GLfloat &max = (r>b)?((r>g)?r:g):((b>g)?b:g);
max -= max/2;
r += max/6;
g += max/6;
b += max/6;
}
//---------------------------------------------------------
void __fastcall TfrmTracado::DrawObjects( void )
{
register int i;
register int j;
register int k;
GLfloat r, g, b;
Solido *s;
Face *f;
Vertice *v;

if( colSolido != NULL )
{
glPushMatrix();
glEdgeFlag( GL_TRUE );
for( i = 0; i < colSolido->Count; i++ )
{
s = (Solido *)colSolido->Items[i];
for( j = 0; j < s->NumFaces; j++ )
{
f =s->pFace[j];
r = (GLfloat)GetRValue( f->Cor ) / 255.0f;
g = (GLfloat)GetGValue( f->Cor ) / 255.0f;
b = (GLfloat)GetBValue( f->Cor ) / 255.0f;
 AdjustColors(r,g,b);
glColor3f( r, g , b);
glNormal3d( f->Normal.x, f->Normal.y, f->Normal.z );
glBegin( GL_TRIANGLE_FAN );
for( k = 0; k < f->NumVertices; k++ )
{
 v = f->pVertice[k];
 glVertex3d( v->p.x, v->p.y, v->p.z );
}
glEnd();
}
}
glPopMatrix();
}
}
//-------------------------------------------------------
void __fastcall TfrmTracado::SetupLights( void )
{
 if( MenuOpcoesLighting->Checked )
 {
GLfloat Light0Pos[] = { R, R, R, 1.0f };
GLfloat Light1Pos[] = { -R, -R, -R, 1.0f };
glLightfv( GL_LIGHT0, GL_POSITION, Light0Pos );
glEnable( GL_LIGHT0 );
glLightfv( GL_LIGHT1, GL_POSITION, Light1Pos );
glEnable( GL_LIGHT1 );
 }
}
//-----------------------------------------------------
void __fastcall TfrmTracado::RenderScene( void )
{
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
 glCallList( SolidsList );
 if( pFonteTree != NULL )
 {
glCallList( SourceList );
 }
 if( IFieldPlaneList )
 {
glCallList (IFieldPlaneList);
glEnable(GL_TEXTURE_2D);
glEnable(GL_BLEND);

glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0);glVertex3f(IFieldAtt->fXmin,
              IFieldAtt->fYmin, IFieldAtt->fZ);
glTexCoord2f(0.0, 1.0);glVertex3f(IFieldAtt->fXmax,
              IFieldAtt->fYmin, IFieldAtt->fZ);
glTexCoord2f(1.0, 1.0);glVertex3f(IFieldAtt->fXmax,
IFieldAtt->fYmax, IFieldAtt->fZ);
glTexCoord2f(1.0, 0.0);glVertex3f(IFieldAtt->fXmin,
             IFieldAtt->fYmax, IFieldAtt->fZ);
glEnd();
glDisable(GL_BLEND);
glDisable(GL_TEXTURE_2D);
 }
 glFlush();
}
//-------------------------------------------------------
void __fastcall TfrmTracado::ChangeSize(
 GLsizei x, GLsizei y, GLsizei w, GLsizei h )
{
 //
 // Evita divisão por 0
 //
 if( h == 0 )
 {
h = 1;
 }
 //
 // Seta Matriz de Viewport para as dimensões da janela
 //
 glViewport( x, y, w, h );
 double Left, Right, Bottom, Top, Near, Far;
 SolidoDxfCollection *colSolidoDxf =
 (SolidoDxfCollection *)colSolido;
 R = 0.0;
 if( colSolidoDxf != NULL )
 {
Left = colSolidoDxf->DxfReader.ExtMinX;
Right = colSolidoDxf->DxfReader.ExtMaxX;
Bottom = colSolidoDxf->DxfReader.ExtMinY;
Top = colSolidoDxf->DxfReader.ExtMaxY;
Near = colSolidoDxf->DxfReader.ExtMinZ;
Far = colSolidoDxf->DxfReader.ExtMaxZ;
R = max( R, fabs( Left ) );
R = max( R, fabs( Right ) );
R = max( R, fabs( Bottom ) );
R = max( R, fabs( Top ) );
R = max( R, fabs( Near ) );
R = max( R, fabs( Far ) );
 }
 else
 {
Left = -1.0;
Right = 1.0;
Bottom = -1.0;
Top = 1.0;
Near = 1.0;
Far = -1.0;
R = 100.0f;
 }
 R *= sqrt( 3.0 );
 glMatrixMode( GL_PROJECTION );
 glLoadIdentity();
 if( MenuOpcoesPerspectiveView->Checked )
 {
GLfloat aspect = (GLfloat)w/(GLfloat)h;
gluPerspective(
 60.0,
 aspect,     // Aspect ratio
 1.0,//10.0,        // Z near
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 9999.0 );   // Z far
 }
 else
 {
if( w <= h )
{
 glOrtho( -R, R, -R*h/w, R*h/w, -R, R );
}
else
{
 glOrtho( -R*w/h, R*w/h, -R, R, -R, R );
}
 }
 glRotatef( -90.0f, 0.0f, 1.0f, 0.0f );
 glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
 glMatrixMode( GL_MODELVIEW );
 glLoadIdentity();

 //glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
 //glRotatef( -90.0f, 0.0f, 0.0f, 1.0f );
 //
 //
 glPushMatrix();
 //
 //
 SetupLights();
 //
 //
 MoveObserver();
 RenderScene();
}
//------------------------------------------------------------
void __fastcall TfrmTracado::MoveObserver( void )
{
 if( MenuOpcoesPerspectiveView->Checked )
 {
double CosTheta = cos( Theta * G2R );
double SinTheta = sin( Theta * G2R );
double CosPhi = cos( Phi * G2R );
double SinPhi = sin( Phi * G2R );
xObs = R * CosTheta * CosPhi;
yObs = R * CosTheta * SinPhi;
zObs = R * SinTheta;

glRotatef( Theta, 0.0f, 1.0f, 0.0f );
glRotatef( -Phi, 0.0f, 0.0f, 1.0f );
glTranslatef( -Scale*xObs, -Scale*yObs, -Scale*zObs );
 }
 else
 {
glRotatef( Theta, 0.0f, 1.0f, 0.0f );
glRotatef( -Phi, 0.0f, 0.0f, 1.0f );
glScalef( Scale, Scale, Scale );
 }
}
//------------------------------------------------------------
void __fastcall TfrmTracado::sbLongitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
 Phi = ScrollPos;
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::sbLatitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
 Theta = -ScrollPos;
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//---------------------------------------------------------
void __fastcall TfrmTracado::sbZoomScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 Scale = pow( 2.0, (double)ScrollPos/2.0 );
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//------------------------------------------------------------
void __fastcall TfrmTracado::btnResetClick(TObject *Sender)
{
 ResetObserver();
 Paint();
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::ResetObserver( void )
{
 sbLongitude->Position = 0;
 sbLatitude->Position = 0;
 sbZoom->Position = 0;
 Theta = 0.0;
 Phi = 0.0;
 Scale = 1.0;
 glMatrixMode( GL_MODELVIEW );
   glPopMatrix();
   glPushMatrix();
   MoveObserver();
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::MenuArquivoAbrirClick(TObject *Sender)
{
 OpenDialog1->FileName = "";
   if( OpenDialog1->Execute() )
   {
 delete colSolido;
 colSolido = NULL;
 delete pFonte;
 pFonte = NULL;
 delete pFonteTree;
 pFonteTree = NULL;
 if( SolidsList != 0 )
 {
 glDeleteLists( SolidsList, 1 );
 SolidsList = 0;
 }
 if( SourceList != 0 )
 {
glDeleteLists( SourceList, 1 );
SourceList = 0;
 }
 try
 {
frmTracado->Enabled = false;
colSolido = new SolidoDxfCollection(
__classid(Solido), OpenDialog1->FileName );
SolidsList = glGenLists( 1 );
    glNewList( SolidsList, GL_COMPILE );
DrawObjects();
glEndList();
 }
  __finally
 {
 frmTracado->Enabled = true;
frmTracado->Caption = "Traçado de Raios - " +
           OpenDialog1->FileName;
Resize();
Paint();
 }
}
}
//-------------------------------------------------------------
void __fastcall TfrmTracado::MenuArquivoSairClick(TObject *Sender)
{
 Application->Terminate();
}
//--------------------------------------------------------------
void __fastcall TfrmTracado::MenuAjudaOpenGLClick(TObject *Sender)
{
 frmAboutOGL->ShowModal();
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesDepthTestClick(TObject *Sender)
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
   MenuOpcoesDepthTest->Checked = !MenuOpcoesDepthTest->Checked;
   if( MenuOpcoesDepthTest->Checked )
   {
      glEnable( GL_DEPTH_TEST );
 }
   else
   {
      glDisable( GL_DEPTH_TEST );
   }
   Paint();
}
//---------------------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesCullFaceClick(TObject *Sender)
{
   MenuOpcoesCullFace->Checked = !MenuOpcoesCullFace->Checked;
   if( MenuOpcoesCullFace->Checked )
   {
      glEnable( GL_CULL_FACE );
   }
   else
 {
      glDisable( GL_CULL_FACE );
   }
   Paint();
}
//----------------------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesBackFillClick(TObject *Sender)
{
 MenuOpcoesBackFill->Checked = !MenuOpcoesBackFill->Checked;
   if( MenuOpcoesBackFill->Checked )
 {
glPolygonMode( GL_BACK , GL_FILL );
   }
   else
   {
      glPolygonMode( GL_BACK , GL_LINE );
 }
   Paint();
}
//--------------------------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesPerspectiveViewClick(
      TObject *Sender)
{
MenuOpcoesPerspectiveView->Checked = !MenuOpcoesPerspectiveView->
     Checked;
 if( MenuOpcoesPerspectiveView->Checked )
   {
   }
   else
 {
   }
 Resize();
   Paint();
}
//-------------------------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesFrontFillClick(TObject *Sender)
{
   MenuOpcoesFrontFill->Checked = !MenuOpcoesFrontFill->Checked;
   if( MenuOpcoesFrontFill->Checked )
 {
      glPolygonMode( GL_FRONT, GL_FILL );
 }
   else
   {
      glPolygonMode( GL_FRONT, GL_LINE );
   }
   Paint();
}
//---------------------------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesLightingClick(TObject *Sender)
{
 MenuOpcoesLighting->Checked = !MenuOpcoesLighting->Checked;
 if( MenuOpcoesLighting->Checked )
 {
glEnable( GL_LIGHTING );
 }
 else
 {
glDisable( GL_LIGHTING );
 }
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 Resize();
 Paint();
}
//-------------------------------------------------------------
void __fastcall TfrmTracado::MenuFonteNovaClick(TObject *Sender)
{
 double Freq;
 if(!dlgNovaFonte->GetNovaFonteCoord( &fonx, &fony,
     &fonz, &Freq, &LMax ) )
 return;

 delete pFonte;

 Vetor3D pos( fonx, fony, fonz );
 Vetor3D ex( 1.0, 0.0, 0.0 );
 Vetor3D ey( 0.0, 1.0, 0.0 );
 Vetor3D ez( 0.0, 0.0, 1.0 );
 double Potencia = 1.0;
 //
 //    Preparação da fonte.
 //
 AntenaIsotropica *pAnt =
 new AntenaIsotropica( pos, ex, ey, ez, 0.0 );
 pFonte = new Fonte( pAnt, Freq, Potencia, NULL );
 delete pAnt;

 GeraImagens();
 if(IFieldPlaneList)
CampodeIntensidades();
}
//-------------------------------------------------
void __fastcall TfrmTracado::GeraImagens(void)
{
 delete pFonteTree;
 //
 // Cria a arvore do método das imagens
 //
 if(!pFonte)
return;

 if(!colSolido)
 {
 ShowMessage( "É necessário abrir primeiramente o arquivo DXF" );
 return;
 }
 Cursor = crHourGlass;
pFonteTree = new ArvoreFonte( *pFonte, NULL, NULL, -1,
 *colSolido, LMax );
 Vetor3D p( -50.0, -50.0, 100.0 );
 //
 //    Calcula o campo no ponto p.
 //
 CVetor3D E( pFonteTree->CampoEm( p ) );
 int n( pFonteTree->ContaNo() );
 n++;
 n--;

 if( SourceList != 0 )
 {
 glDeleteLists( SourceList, 1 );
 }
 SourceList = glGenLists( 1 );
 glNewList( SourceList, GL_COMPILE );
 count = 0; ///////
 pFonteTree->DrawFonte( R );
 glEndList();

 Paint();
 Cursor = crDefault;
}
//-----------------------------------------------------

void TfrmTracado::RedrawIField(void)
{
double x0 = IFieldAtt->fXmin;
double x1 = IFieldAtt->fXmax;
double y0 = IFieldAtt->fYmin;
double y1 = IFieldAtt->fYmax;
double z  = IFieldAtt->fZ;
int width = (int) x1 - x0 + 1;
int height= (int) y1 - y0 + 1;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
delete bits;

oldwidth = width;
oldheight = height;

bits = new unsigned char [height*width*4];
for(int i = 0; i < height; i++)
{
for(int j = 0; j < width; j++)
{
bits[(i*width + j)*4 + 0]= GetRValue(IFieldBM->
       Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 1]= GetGValue(IFieldBM->
       Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 2]= GetBValue(IFieldBM->
        Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 3]= 50;
}
}

IFieldPlaneList = 5;glGenLists(1);
glNewList(IFieldPlaneList, GL_COMPILE);
glPixelTransferi(GL_ALPHA_SCALE, 1.0f);

glTexImage2D(GL_TEXTURE_2D, 0, 4, width, height,
          0, GL_RGBA, GL_UNSIGNED_BYTE,bits);

glEndList();
}

void TfrmTracado::FillOutIFieldBM(void)
{
int y=0;
Graphics::TBitmap * BM = IFieldBM;
BM->Width = (int) (IFieldAtt->fXmax -
                 IFieldAtt->fXmin)+1;
BM->Height = (int) (IFieldAtt->fYmax -
                  IFieldAtt->fYmin)+1;
BM->HandleType = bmDIB;
BM->PixelFormat = pf24bit;

double value;
double max=0;
double min=10e32;
CVetor3D cv;
Vetor3D dv;

for (int yy = (int) IFieldAtt->fYmin;
             yy <= (int) IFieldAtt->fYmax; yy++) {
byte *BackgroundSL = ((byte *)BM->ScanLine[y++]);

for (int xx = (int) IFieldAtt->fXmin;
             xx <= (int) IFieldAtt->fXmax; xx++) {
dv = Vetor3D(xx,yy,IFieldAtt->fZ);

  cv = pFonteTree->CampoEm(dv);
value = cv.Norma();
value *= value;
max = (max>value)?max:value;
min = (min<value)?min:value;
}
staStatus->SimpleText = AnsiString("Processando: ");
staStatus->SimpleText = staStatus->SimpleText +
                   AnsiString((int)(50*(yy-IFieldAtt->fYmin)/
                  (IFieldAtt->fYmax - IFieldAtt->fYmin)));
staStatus->SimpleText = staStatus->
             SimpleText + AnsiString(" %");
}
if ( min==max )max=1 + min;
y=0;
for (int yy = (int) IFieldAtt->fYmin;
          yy <= (int) IFieldAtt->fYmax; yy++) {
byte *BackgroundSL = ((byte *)BM->ScanLine[y++]);

for (int xx = (int) IFieldAtt->fXmin;
              xx <= (int) IFieldAtt->fXmax; xx++) {
dv = Vetor3D(xx,yy,IFieldAtt->fZ);
cv = pFonteTree->CampoEm(dv);
value = cv.Norma();
value *= value;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
value = log10((9*(value-min)/(max-min)) + 1);
byte r,g,b;
SpectralColor(value,r,g,b);

BackgroundSL[0] = b;
BackgroundSL[2] = g;
BackgroundSL[1] = r;

BackgroundSL+=3;
}
staStatus->SimpleText = AnsiString("Processando: ");
staStatus->SimpleText = staStatus->SimpleText +
  AnsiString((int)(50 + 50*(yy-IFieldAtt->fYmin)/
  (IFieldAtt->fYmax - IFieldAtt->fYmin)));
staStatus->SimpleText = staStatus->SimpleText +
   AnsiString(" %");
}
staStatus->SimpleText = AnsiString("Cálculo Efetuado");
}

void __fastcall TfrmTracado::CampodeIntensidades(void)
{
delete IFieldBM;
IFieldBM = new Graphics::TBitmap;

IField->ClientWidth = 200;
IField->ClientHeight = 200;
FillOutIFieldBM();

if(IFieldPlaneList)
{
 glDeleteLists(IFieldPlaneList, 1);
 IFieldPlaneList = 0;
}

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

RedrawIField();
Paint();

IField->Setup(IFieldAtt->fXmin, IFieldAtt->fYmin,
IFieldAtt->fXmax, IFieldAtt->fYmax, IFieldAtt->fZ);
//IField->Show();
if( wglMakeCurrent( hDC, hRC ) == false )
{
ShowMessage( "Could not MakeCurrent" );
}
glViewport( 0, 0, ClientWidth, ClientHeight);

Cursor = crDefault;
}

void __fastcall TfrmTracado::
          CampodeIntensidades1Click(TObject *Sender)
{

if(IFieldAtt->ShowModal()==mrCancel)
return;

if(!pFonteTree)
return;

CampodeIntensidades();
}
//---------------------------------------------------------
void __fastcall TfrmTracado::Ambiente1Click(TObject *Sender)
{
EnvSettings->ShowModal();
}
//---------------------------------------------------------


//-------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
#include "fViewdxf.h"
#include "TIFieldAtt.h"
#include "TIFIeld.h"
#include "Dlg_EnvSettings.h"

#include <math.h>
#include <gl/glu.h>

#include "SolidoDxf.h"
#include "AboutOGL.h"
#include "DXF.h"
#include "fNovaFonte.h"
#include "Fonte.h"
//---------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmTracado *frmTracado;

//--------------------------------------------
static const double PI = 4.0 * atan( 1.0 );
static const double G2R = PI / 180.0;
//----------------------------------------
void SpectralColor(float Int,byte &r,byte &g,byte &b)
{
// linear scale
if (Int < 0.)
 { r = 0; g = 0; b = 255; }
else if (Int < 0.25)
 { r = 0; g = Int*4.*255; b = 255; }
else if (Int < 0.5)
 { r = 0; g = 255; b = 255-(Int-0.25)*4.*255; }
else if (Int < 0.75)
 { r = (Int-0.5)*4.*255; g = 255; b = 0; }
else if (Int < 1.)
 { r = 255; g = 255-(Int-0.75)*4*255; b = 0; }
else
 { r = 255; g = 0; b = 0; }
}



__fastcall TfrmTracado::TfrmTracado(TComponent* Owner)
 : TForm(Owner)
{
 hRC = NULL;
 colSolido = NULL;
 LeftOffset = sbZoom->Width;
 TopOffset = 0;
 BottomOffset = sbLongitude->Height;
 sbZoom->Left = 0;
 sbZoom->Top = 0;
   sbLongitude->Left = LeftOffset;
   sbLatitude->Top = TopOffset;

 xObs = 0.0;
   yObs = 0.0;
   zObs = 0.0;
   Theta = 0.0;
 Phi = 0.0;
 R = 1.0;
 Scale = 1.0;

 pFonte = NULL;
 pFonteTree = NULL;

 SolidsList = 0;
 SourceList = 0;
 IFieldPlaneList = 0;

 IFieldBM = NULL;
 bits = NULL;
}
//-------------------------------------------------
__fastcall TfrmTracado::~TfrmTracado( void )
{
 if( colSolido != NULL )
   {
      delete colSolido;
 }
   if( pFonte != NULL )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   {
      delete pFonte;
 }
   if( pFonteTree != NULL )
   {
delete pFonteTree;
 }
 delete IFieldBM;
 delete bits;

}

void __fastcall TfrmTracado::_GetDC(void)
{
 if(hRC) return;
 hWnd = Handle;
 hDC = GetDC( hWnd );

 SetPixelFormatDescriptor();
 hRC = wglCreateContext( hDC );
 if(!hRC)
 {
ShowMessage( "hrc = NULL" );
 }
 if(wglMakeCurrent(hDC, hRC)==false)
 {
ShowMessage( "Could not MakeCurrent" );
 }
}

void __fastcall TfrmTracado::_ReleaseDC(void)
{
 ReleaseDC( hWnd, hDC );
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::FormCreate(TObject *Sender)
{
 _GetDC();

 SetupRC();
 SetupLighting();
 SetupTextures();
 CreateObjects();
}
//----------------------------------------------------------------
void __fastcall TfrmTracado::FormDestroy(TObject *Sender)
{
 DestroyObjects();
 wglMakeCurrent( NULL, NULL );
 wglDeleteContext( hRC );
 _ReleaseDC();
}
//----------------------------------------------------------------
void __fastcall TfrmTracado::FormResize(TObject *Sender)
{
 NewClientWidth = ClientWidth
- sbLatitude->Width - LeftOffset;
 NewClientHeight = ClientHeight
- staStatus->Height - sbLongitude->Height - TopOffset;
 sbZoom->Height = NewClientHeight;
 sbLongitude->Top = TopOffset + NewClientHeight;
 sbLongitude->Width = NewClientWidth;
 sbLatitude->Left = LeftOffset + NewClientWidth;
 sbLatitude->Height = NewClientHeight;
 btnReset->Left = LeftOffset + NewClientWidth;
 btnReset->Top = TopOffset + NewClientHeight;
 ChangeSize( LeftOffset, BottomOffset,
     NewClientWidth, NewClientHeight );
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::FormPaint(TObject *Sender)
{
 //_ReleaseDC();
 //_GetDC();

 RenderScene();
 SwapBuffers( hDC );
}
//--------------------------------------------------------------
void __fastcall TfrmTracado::FormKeyDown(TObject *Sender, WORD &Key,
TShiftState Shift)
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{

 switch( Key )
 {

case VK_HOME:
 break;

case VK_END:
 break;

case VK_UP:
 break;

case VK_DOWN:
 break;

case VK_LEFT:
 break;

case VK_RIGHT:
 break;

case VK_PRIOR:
 break;

case VK_NEXT:
 break;
 }
}
//---------------------------------------------------------
void __fastcall TfrmTracado::SetPixelFormatDescriptor( void )
{
 PIXELFORMATDESCRIPTOR pfd = {
sizeof(PIXELFORMATDESCRIPTOR),
1,
PFD_DRAW_TO_WINDOW |
PFD_SUPPORT_OPENGL |
PFD_DOUBLEBUFFER,
PFD_TYPE_RGBA,
24,
0, 0, 0, 0, 0, 0,
0, 0,
0, 0, 0, 0, 0,
32,
0,
0,
PFD_MAIN_PLANE,
0,
      0, 0, 0
   };
 PixelFormat = ChoosePixelFormat( hDC, &pfd );
 SetPixelFormat( hDC, PixelFormat, &pfd );
}
//--------------------------------------------------------
void __fastcall TfrmTracado::SetupRC( void )
{
 glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
 glColor3f( 0.0f, 0.70f, 0.0f );
 //
 // Point and line parameters
 //
 glGetFloatv( GL_POINT_SIZE_RANGE, glPointSizes );
 glGetFloatv( GL_POINT_SIZE_GRANULARITY, &glPointStep );
 glGetFloatv( GL_LINE_WIDTH_RANGE, glLineSizes );
 glGetFloatv( GL_LINE_WIDTH_GRANULARITY, &glLineStep );
 //
 glFrontFace( GL_CCW );
 //
 // Shading model
 //
 glShadeModel( GL_SMOOTH );
 //  glShadeModel( GL_FLAT );
 //
 // Diversos ( Opcoes default do Menu )
 //
 glEnable( GL_POINT_SMOOTH );
 glEnable( GL_DEPTH_TEST );
 glEnable( GL_CULL_FACE );
 glPolygonMode( GL_FRONT, GL_FILL );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 glPolygonMode( GL_BACK, GL_LINE );
 glEnable( GL_LIGHTING );
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::SetupLighting( void )
{
 if( MenuOpcoesLighting->Checked )
 {
//
//
GLfloat GlobalAmbientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
glLightModeli( GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE );
glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE );
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, GlobalAmbientLight );
//
//
GLfloat LightAmbient[]  = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat LightDiffuse[]  = { 1.0f, 1.0f, 0.5f, 1.0f };
GLfloat LightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glLightfv( GL_LIGHT0, GL_AMBIENT, LightAmbient );
glLightfv( GL_LIGHT0, GL_DIFFUSE, LightDiffuse );

glLightfv( GL_LIGHT1, GL_AMBIENT, LightAmbient );
glLightfv( GL_LIGHT1, GL_DIFFUSE, LightDiffuse );

// Coloração de material
//
glEnable( GL_COLOR_MATERIAL );
glColorMaterial( GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE );
GLfloat ambientRef[] = { .5f, .5f, .5f, 1.0f };
GLfloat difuseRef[] =  { .7f, .7f, .7f, .7f };

glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT, ambientRef );
glMaterialfv( GL_FRONT_AND_BACK, GL_DIFFUSE, difuseRef );
 }
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::SetupTextures( void )
{
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::CreateObjects( void )
{
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::DestroyObjects( void )
{

}
//-----------------------------------------------------------
void __fastcall TfrmTracado::CalcNormal(
 GLfloat *p1, GLfloat *p2, GLfloat *p3, GLfloat *Normal )
{
 GLfloat v1[3];
 GLfloat v2[3];

 v1[0] = p2[0] - p1[0];
 v1[1] = p2[1] - p1[1];
 v1[2] = p2[2] - p1[2];
 v2[0] = p3[0] - p2[0];
 v2[1] = p3[1] - p2[1];
 v2[2] = p3[2] - p2[2];
 Normal[0] = v1[1]*v2[2] - v1[2]*v2[1];
 Normal[1] = v1[2]*v2[0] - v1[0]*v2[2];
 Normal[2] = v1[0]*v2[1] - v1[1]*v2[0];
 double Norma = sqrt(
Normal[0] * Normal[0] + Normal[1] * Normal[1]
           + Normal[2] * Normal[2] );
 Normal[0] /= Norma;
 Normal[1] /= Norma;
 Normal[2] /= Norma;
}

//---------------------------------------------------------
void AdjustColors(GLfloat &r, GLfloat &g, GLfloat &b)
{
GLfloat &max = (r>b)?((r>g)?r:g):((b>g)?b:g);
max -= max/2;
r += max/6;
g += max/6;
b += max/6;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
}
//-------------------------------------------------------
void __fastcall TfrmTracado::DrawObjects( void )
{
register int i;
register int j;
register int k;
GLfloat r, g, b;
Solido *s;
Face *f;
Vertice *v;

if( colSolido != NULL )
{
glPushMatrix();
glEdgeFlag( GL_TRUE );
for( i = 0; i < colSolido->Count; i++ )
{
s = (Solido *)colSolido->Items[i];
for( j = 0; j < s->NumFaces; j++ )
{
f =s->pFace[j];
r = (GLfloat)GetRValue( f->Cor ) / 255.0f;
g = (GLfloat)GetGValue( f->Cor ) / 255.0f;
b = (GLfloat)GetBValue( f->Cor ) / 255.0f;
        AdjustColors(r,g,b);
glColor3f( r, g , b);
glNormal3d( f->Normal.x, f->Normal.y, f->Normal.z );
glBegin( GL_TRIANGLE_FAN );
for( k = 0; k < f->NumVertices; k++ )
{
 v = f->pVertice[k];
 glVertex3d( v->p.x, v->p.y, v->p.z );
}
glEnd();
}
}
glPopMatrix();
}
}
//----------------------------------------------------
void __fastcall TfrmTracado::SetupLights( void )
{
 //
 // Habilita iluminação
 //
 if( MenuOpcoesLighting->Checked )
 {
GLfloat Light0Pos[] = { R, R, R, 1.0f };
GLfloat Light1Pos[] = { -R, -R, -R, 1.0f };
glLightfv( GL_LIGHT0, GL_POSITION, Light0Pos );
glEnable( GL_LIGHT0 );
glLightfv( GL_LIGHT1, GL_POSITION, Light1Pos );
glEnable( GL_LIGHT1 );
 }
}
//-----------------------------------------------------
void __fastcall TfrmTracado::RenderScene( void )
{
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
 glCallList( SolidsList );
 if( pFonteTree != NULL )
 {
glCallList( SourceList );
 }
 if( IFieldPlaneList )
 {
glCallList (IFieldPlaneList);
glEnable(GL_TEXTURE_2D);
glEnable(GL_BLEND);

glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0);glVertex3f(IFieldAtt->fXmin,
IFieldAtt->fYmin, IFieldAtt->fZ);
glTexCoord2f(0.0, 1.0);glVertex3f(IFieldAtt->fXmax,
IFieldAtt->fYmin, IFieldAtt->fZ);
glTexCoord2f(1.0, 1.0);glVertex3f(IFieldAtt->fXmax,
IFieldAtt->fYmax, IFieldAtt->fZ);
glTexCoord2f(1.0, 0.0);glVertex3f(IFieldAtt->fXmin,
IFieldAtt->fYmax, IFieldAtt->fZ);
glEnd();
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
glDisable(GL_BLEND);
glDisable(GL_TEXTURE_2D);
 }
 glFlush();
}
//-------------------------------------------------------------
void __fastcall TfrmTracado::ChangeSize(
 GLsizei x, GLsizei y, GLsizei w, GLsizei h )
{
 //
 // Evita divisão por 0
 //
 if( h == 0 )
 {
h = 1;
 }
 glViewport( x, y, w, h );
 //
 double Left, Right, Bottom, Top, Near, Far;
 SolidoDxfCollection *colSolidoDxf =
       (SolidoDxfCollection *)colSolido;
 R = 0.0;
 if( colSolidoDxf != NULL )
 {
Left = colSolidoDxf->DxfReader.ExtMinX;
Right = colSolidoDxf->DxfReader.ExtMaxX;
Bottom = colSolidoDxf->DxfReader.ExtMinY;
Top = colSolidoDxf->DxfReader.ExtMaxY;
Near = colSolidoDxf->DxfReader.ExtMinZ;
Far = colSolidoDxf->DxfReader.ExtMaxZ;
R = max( R, fabs( Left ) );
R = max( R, fabs( Right ) );
R = max( R, fabs( Bottom ) );
R = max( R, fabs( Top ) );
R = max( R, fabs( Near ) );
R = max( R, fabs( Far ) );
 }
 else
 {
Left = -1.0;
Right = 1.0;
Bottom = -1.0;
Top = 1.0;
Near = 1.0;
Far = -1.0;
R = 100.0f;
 }
 R *= sqrt( 3.0 );
 //
 // Reseta a pilha da matriz de Projeção
 //
 glMatrixMode( GL_PROJECTION );
 glLoadIdentity();
 if( MenuOpcoesPerspectiveView->Checked )
 {

GLfloat aspect = (GLfloat)w/(GLfloat)h;
gluPerspective(
 60.0,
 aspect,
 1.0,//10.0,        // Z near
 9999.0 );   // Z far
 }
 else
 {
if( w <= h )
{
 glOrtho( -R, R, -R*h/w, R*h/w, -R, R );
}
else
{
 glOrtho( -R*w/h, R*w/h, -R, R, -R, R );
}
 }
 glRotatef( -90.0f, 0.0f, 1.0f, 0.0f );
 glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
 //
 // Reseta a pilha da matriz Modelview
 //
 glMatrixMode( GL_MODELVIEW );
 glLoadIdentity();
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 //glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
 //glRotatef( -90.0f, 0.0f, 0.0f, 1.0f );
 glPushMatrix();
 //
 //
 SetupLights();
 //
 //
 MoveObserver();
 RenderScene();
}
//-----------------------------------------------
void __fastcall TfrmTracado::MoveObserver( void )
{
 if( MenuOpcoesPerspectiveView->Checked )
 {
double CosTheta = cos( Theta * G2R );
double SinTheta = sin( Theta * G2R );
double CosPhi = cos( Phi * G2R );
double SinPhi = sin( Phi * G2R );
xObs = R * CosTheta * CosPhi;
yObs = R * CosTheta * SinPhi;
zObs = R * SinTheta;

glRotatef( Theta, 0.0f, 1.0f, 0.0f );
glRotatef( -Phi, 0.0f, 0.0f, 1.0f );
glTranslatef( -Scale*xObs, -Scale*yObs, -Scale*zObs );
 }
 else
 {
glRotatef( Theta, 0.0f, 1.0f, 0.0f );
glRotatef( -Phi, 0.0f, 0.0f, 1.0f );
glScalef( Scale, Scale, Scale );
 }
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::sbLongitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
 Phi = ScrollPos;
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//---------------------------------------------------------------
void __fastcall TfrmTracado::sbLatitudeScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
 Theta = -ScrollPos;
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::sbZoomScroll(TObject *Sender,
TScrollCode ScrollCode, int &ScrollPos)
{
 Scale = pow( 2.0, (double)ScrollPos/2.0 );
 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();
 glPushMatrix();
 MoveObserver();
 Paint();
}
//-----------------------------------------------------------
void __fastcall TfrmTracado::btnResetClick(TObject *Sender)
{
 ResetObserver();
 Paint();
}
//----------------------------------------------------------
void __fastcall TfrmTracado::ResetObserver( void )
{
 sbLongitude->Position = 0;
 sbLatitude->Position = 0;
 sbZoom->Position = 0;
 Theta = 0.0;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 Phi = 0.0;
 Scale = 1.0;
 glMatrixMode( GL_MODELVIEW );
   glPopMatrix();
   glPushMatrix();
   MoveObserver();
}
//-------------------------------------------------------------
void __fastcall TfrmTracado::MenuArquivoAbrirClick(TObject
*Sender)
{
 OpenDialog1->FileName = "";
   if( OpenDialog1->Execute() )
   {
 delete colSolido;
 colSolido = NULL;
 delete pFonte;
 pFonte = NULL;
 delete pFonteTree;
 pFonteTree = NULL;
 if( SolidsList != 0 )
 {
 glDeleteLists( SolidsList, 1 );
 SolidsList = 0;
 }
 if( SourceList != 0 )
 {
glDeleteLists( SourceList, 1 );
SourceList = 0;
 }
 try
 {
frmTracado->Enabled = false;
colSolido = new SolidoDxfCollection(
__classid(Solido), OpenDialog1->FileName );
SolidsList = glGenLists( 1 );
     glNewList( SolidsList, GL_COMPILE );
DrawObjects();
glEndList();
 }
 __finally
 {
 frmTracado->Enabled = true;
frmTracado->Caption = "Traçado de Raios - " +
             OpenDialog1->FileName;
Resize();
Paint();
 }
}
}
//---------------------------------------------------------
void __fastcall TfrmTracado::
              MenuArquivoSairClick(TObject *Sender)
{
 Application->Terminate();
}
//-------------------------------------------------------------
void __fastcall TfrmTracado::
              MenuAjudaOpenGLClick(TObject *Sender)
{
 frmAboutOGL->ShowModal();
}
//--------------------------------------------------------
void __fastcall TfrmTracado::
               MenuOpcoesDepthTestClick(TObject *Sender)
{
   MenuOpcoesDepthTest->Checked =
           !MenuOpcoesDepthTest->Checked;
   if( MenuOpcoesDepthTest->Checked )
   {
      glEnable( GL_DEPTH_TEST );
 }
   else
   {
      glDisable( GL_DEPTH_TEST );
   }
   Paint();
}
//----------------------------------------------
void __fastcall TfrmTracado::
MenuOpcoesCullFaceClick(TObject *Sender)
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
   MenuOpcoesCullFace->Checked =
   !MenuOpcoesCullFace->Checked;
   if( MenuOpcoesCullFace->Checked )
   {
      glEnable( GL_CULL_FACE );
   }
   else
 {
      glDisable( GL_CULL_FACE );
   }
   Paint();
}
//-----------------------------------------------
void __fastcall TfrmTracado::
MenuOpcoesBackFillClick(TObject *Sender)
{
 MenuOpcoesBackFill->Checked =
        !MenuOpcoesBackFill->Checked;
   if( MenuOpcoesBackFill->Checked )
 {
glPolygonMode( GL_BACK , GL_FILL );
   }
   else
   {
      glPolygonMode( GL_BACK , GL_LINE );
 }
   Paint();
}
//------------------------------------------------
void __fastcall TfrmTracado::MenuOpcoesPerspectiveViewClick(
      TObject *Sender)
{
 MenuOpcoesPerspectiveView->Checked =
       !MenuOpcoesPerspectiveView->Checked;
 if( MenuOpcoesPerspectiveView->Checked )
   {
   }
   else
 {
   }
 Resize();
   Paint();
}
//----------------------------------------------------
void __fastcall TfrmTracado::
            MenuOpcoesFrontFillClick(TObject *Sender)
{
   MenuOpcoesFrontFill->Checked =
   !MenuOpcoesFrontFill->Checked;
   if( MenuOpcoesFrontFill->Checked )
 {
      glPolygonMode( GL_FRONT, GL_FILL );
 }
   else
   {
      glPolygonMode( GL_FRONT, GL_LINE );
   }
   Paint();
}
//-----------------------------------------------------
void __fastcall TfrmTracado::
         MenuOpcoesLightingClick(TObject *Sender)
{
 MenuOpcoesLighting->Checked =
          !MenuOpcoesLighting->Checked;
 if( MenuOpcoesLighting->Checked )
 {
glEnable( GL_LIGHTING );
 }
 else
 {
glDisable( GL_LIGHTING );
 }
 Resize();
 Paint();
}
//--------------------------------------------------
void __fastcall TfrmTracado::
         MenuFonteNovaClick(TObject *Sender)
{
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 double Freq;
 if(!dlgNovaFonte->GetNovaFonteCoord( &fonx, &fony,
              &fonz, &Freq, &LMax ) )
 return;

 delete pFonte;

 Vetor3D pos( fonx, fony, fonz );
 Vetor3D ex( 1.0, 0.0, 0.0 );
 Vetor3D ey( 0.0, 1.0, 0.0 );
 Vetor3D ez( 0.0, 0.0, 1.0 );
 double Potencia = 1.0;
 //
 //    Preparação da fonte.
 //
 AntenaIsotropica *pAnt =
 new AntenaIsotropica( pos, ex, ey, ez, 0.0 );
 pFonte = new Fonte( pAnt, Freq, Potencia, NULL );
 delete pAnt;

 GeraImagens();
 if(IFieldPlaneList)
CampodeIntensidades();
}
//-------------------------------------------------------
void __fastcall TfrmTracado::GeraImagens(void)
{
 delete pFonteTree;
 //
 // Cria a arvore do método das imagens
 //
 if(!pFonte)
return;

 if(!colSolido)
 {
ShowMessage( "É necessário abrir primeiramente o arquivo DXF" );
 return;
 }
 Cursor = crHourGlass;
pFonteTree = new ArvoreFonte( *pFonte, NULL,
          NULL, -1, *colSolido, LMax );
 Vetor3D p( -50.0, -50.0, 100.0 );
 //
 //    Calcula o campo no ponto p.
 //
 CVetor3D E( pFonteTree->CampoEm( p ) );

 int n( pFonteTree->ContaNo() );
 n++;
 n--;

 if( SourceList != 0 )
 {
 glDeleteLists( SourceList, 1 );
 }
 SourceList = glGenLists( 1 );
 glNewList( SourceList, GL_COMPILE );
 count = 0; ///////
 pFonteTree->DrawFonte( R );
 glEndList();

 Paint();
 Cursor = crDefault;
}
//------------------------------------------------------

void TfrmTracado::RedrawIField(void)
{
double x0 = IFieldAtt->fXmin;
double x1 = IFieldAtt->fXmax;
double y0 = IFieldAtt->fYmin;
double y1 = IFieldAtt->fYmax;
double z  = IFieldAtt->fZ;
int width = (int) x1 - x0 + 1;
int height= (int) y1 - y0 + 1;

delete bits;

oldwidth = width;
oldheight = height;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
bits = new unsigned char [height*width*4];
for(int i = 0; i < height; i++)
{
for(int j = 0; j < width; j++)
{
bits[(i*width + j)*4 + 0]= GetRValue(IFieldBM->
               Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 1]= GetGValue(IFieldBM->
               Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 2]= GetBValue(IFieldBM->
               Canvas->Pixels[i][j]);
bits[(i*width + j)*4 + 3]= 50;
}
}

IFieldPlaneList = 5;glGenLists(1);
glNewList(IFieldPlaneList, GL_COMPILE);
glPixelTransferi(GL_ALPHA_SCALE, 1.0f);

glTexImage2D(GL_TEXTURE_2D, 0, 4, width, height, 0,
                  GL_RGBA, GL_UNSIGNED_BYTE,bits);

glEndList();
}

void TfrmTracado::FillOutIFieldBM(void)
{
int y=0;
Graphics::TBitmap * BM = IFieldBM;
BM->Width = (int) (IFieldAtt->fXmax - IFieldAtt->fXmin)+1;
BM->Height = (int) (IFieldAtt->fYmax - IFieldAtt->fYmin)+1;
BM->HandleType = bmDIB;
BM->PixelFormat = pf24bit;

double value;
double max=0;
double min=10e32;
CVetor3D cv;
Vetor3D dv;

for (int yy = (int) IFieldAtt->fYmin; yy <=
    (int) IFieldAtt->fYmax; yy++) {
byte *BackgroundSL = ((byte *)BM->ScanLine[y++]);

for (int xx = (int) IFieldAtt->fXmin;
        xx <= (int) IFieldAtt->fXmax; xx++) {
dv = Vetor3D(xx,yy,IFieldAtt->fZ);

  cv = pFonteTree->CampoEm(dv);
value = cv.Norma();
value *= value;
max = (max>value)?max:value;
min = (min<value)?min:value;
}
staStatus->SimpleText = AnsiString("Processando: ");
staStatus->SimpleText = staStatus->SimpleText +
AnsiString((int)(50*(yy-IFieldAtt->fYmin)/(IFieldAtt->fYmax
          - IFieldAtt->fYmin)));
staStatus->SimpleText = staStatus->SimpleText +
AnsiString(" %");
}
if ( min==max )max=1 + min;
y=0;
for (int yy = (int) IFieldAtt->fYmin;
         yy <= (int) IFieldAtt->fYmax; yy++) {
byte *BackgroundSL = ((byte *)BM->ScanLine[y++]);

for (int xx = (int) IFieldAtt->fXmin;
       xx <= (int) IFieldAtt->fXmax; xx++) {
dv = Vetor3D(xx,yy,IFieldAtt->fZ);
cv = pFonteTree->CampoEm(dv);
value = cv.Norma();
value *= value;
value = log10((9*(value-min)/(max-min)) + 1);
byte r,g,b;
SpectralColor(value,r,g,b);

BackgroundSL[0] = b;
BackgroundSL[2] = g;
BackgroundSL[1] = r;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
BackgroundSL+=3;
}
staStatus->SimpleText =
AnsiString("Processando: ");
staStatus->SimpleText =
staStatus->SimpleText
+ AnsiString((int)(50 + 50*
    (yy-IFieldAtt->fYmin)/(IFieldAtt->fYmax
        - IFieldAtt->fYmin)));
staStatus->SimpleText =
staStatus->SimpleText + AnsiString(" %");
}
staStatus->SimpleText =
AnsiString("Cálculo Efetuado");
}

void __fastcall TfrmTracado::CampodeIntensidades(void)
{
delete IFieldBM;
IFieldBM = new Graphics::TBitmap;
IField->ClientWidth = 200;
IField->ClientHeight = 200;
FillOutIFieldBM();
if(IFieldPlaneList)
{
 glDeleteLists(IFieldPlaneList, 1);
 IFieldPlaneList = 0;
}
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
RedrawIField();
Paint();
IField->Setup(IFieldAtt->fXmin, IFieldAtt->fYmin,
IFieldAtt->fXmax, IFieldAtt->fYmax, IFieldAtt->fZ);
//IField->Show();
if( wglMakeCurrent( hDC, hRC ) == false )
{
ShowMessage( "Could not MakeCurrent" );
}
glViewport( 0, 0, ClientWidth, ClientHeight);

Cursor = crDefault;
}

void __fastcall TfrmTracado::
         CampodeIntensidades1Click(TObject *Sender)
{
if(IFieldAtt->ShowModal()==mrCancel)
return;
if(!pFonteTree)
return;
CampodeIntensidades();
}
//-------------------------------------------------------
void __fastcall TfrmTracado::Ambiente1Click(TObject *Sender)
{
EnvSettings->ShowModal();
}
//-----------------------------------------------------


//-----------------------------------------------------
#ifndef SolidoH
#define SolidoH
//-----------------------------------------------------
#include "Vetor3D.h"
//-----------------------------------------------------
#define EPSILON0  (8.854e-12)
#define SIGMA0    0
#define MI0       (4*pi*1e-7)
#define SIGN( x ) ( ((x)>0)?1:( ((x)<0)?-1:0 ) )
//---------------------------------------------------
class ESolidoException : public Exception
{
public:
   __fastcall ESolidoException( const AnsiString Msg )
      : Exception( Msg ) {};
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
};

class Vertice
{
public:
   Vetor3D p;
   int Cor;

public:
   __fastcall Vertice( const Vetor3D &p1, int nCor );
   __fastcall Vertice( const Vertice &v );
   __fastcall ~Vertice( void );
   int __fastcall operator==( const Vertice &v ) const;
};

class Aresta
{
public:
   Vertice v1;
   Vertice v2;
   int Visivel;
   int Cor;
   Vetor3D CM;
   Vetor3D Nf;
   int SignCM;

public:
   __fastcall Aresta( const Vertice &v1,
     const Vertice &v2, int vis, int nCor );
   __fastcall Aresta( const Aresta &a );
   __fastcall ~Aresta( void );
   int __fastcall operator==( const Aresta &a ) const;
};

class Face
{
public:
   int NumVertices;
   Vertice **pVertice;
   int NumArestas;
   Aresta **pAresta;
   Vetor3D Normal; // Nx = A; Ny = B; Nz = C;
   double D;       // Ax + By + Cz + D = 0
   int Cor;
   Vetor3D CM;    // Centro de massa
   //
   double Epsilon;
   double Sigma;
   double Mi;

public:
   __fastcall Face(
      int nV, Vertice **pV, int nA, Aresta **pA,
      double e, double s, double m, int nCor );
   __fastcall Face( const Face &f );
   __fastcall ~Face( void );
   int __fastcall operator==( const Face &f ) const;
   int __fastcall operator!=( const Face &f ) const;
   bool __fastcall BackFace( const Vetor3D &p ) const;
   bool __fastcall PertenceAFace( const Vetor3D &p ) const;
   int __fastcall Contains( const Vertice &v, double tol ) const;
   int __fastcall Coplanar( const Face &A, double Alfa ) const;
   bool __fastcall Intersection(Vetor3D &v,
         const Vetor3D &v1, const Vetor3D &v2);
 Face * __fastcall MergeFace( const Face &A ) const;
   void __fastcall MergeTriangulo( const Face &T );
};

class Triangulo : public Face
{
public:
   __fastcall Triangulo(
      Vertice **pV, Aresta **pA,
      double e, double s, double m, int nCor );
   __fastcall ~Triangulo( void );
};

class Solido : public TCollectionItem
{
public:
   int NumVertices;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   Vertice **pVertice;
   int NumArestas;
   Aresta **pAresta;
   int NumFaces;
   Face **pFace;
   int Cor;
   Vetor3D CM;    // Centro de massa

public:
   __fastcall Solido( TCollection *TCol, int n,
      Vertice **pV, int nCor );
   __fastcall Solido( TCollection *TCol,
      int nV, Vertice **pV,
      int nA, Aresta **pA,
      int nF, Face **pF,
      int nCor );
   virtual __fastcall ~Solido( void );
   void __fastcall CreateFaces( void );
};

class SolidoCollection : public TCollection
{
public:
   __fastcall SolidoCollection( TMetaClass *classid );
   __fastcall ~SolidoCollection( void );
};
//---------------------------------------------------
   #if defined( __SOLIDO_CPP__ )
         SolidoCollection *colSolido;
      #else
         extern SolidoCollection *colSolido;
   #endif
//--------------------------------------------------
#endif


//-------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop

#define __SOLIDO_CPP__

#include "Solido.h"

#include <math.h>

//-------------------------------------------------
#pragma package(smart_init)

//-----------------------------------------------
static const double pi = 4.0 * atan( 1.0 );
static double Minimo = 1.0e-10;
//---------------------------------------------------
__fastcall Vertice::Vertice( const Vetor3D &p1,
 int nCor )
:
p( p1 ),
Cor( nCor )
{
}
//--------------------------------------------------
__fastcall Vertice::Vertice( const Vertice &v )
:
p( v.p ),
Cor( v.Cor )
{
}
//-----------------------------------------------
__fastcall Vertice::~Vertice( void )
{
}
//-----------------------------------------------
int __fastcall Vertice::operator==( const
         Vertice &v ) const
{
   int Result;

   Result =
      p.x == v.p.x &&
      p.y == v.p.y &&
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      p.z == v.p.z;

   return Result;
}
//---------------------------------------------------
__fastcall Aresta::Aresta(
   const Vertice &v11, const Vertice &v21,
    int vis, int nCor )
:
v1( v11 ),
v2( v21 ),
Visivel( vis ),
Cor( nCor ),
CM( ( v11.p + v21.p ) / 2.0 ),
Nf(),
SignCM( 1 )
{
}
//----------------------------------------------------
__fastcall Aresta::Aresta( const Aresta &a )
:
v1( a.v1 ),
v2( a.v2 ),
Visivel( a.Visivel ),
Cor( a.Cor ),
CM( a.CM ),
Nf( a.Nf ),
SignCM( a.SignCM )
{
}
//---------------------------------------------------
__fastcall Aresta::~Aresta( void )
{
}
//----------------------------------------------------
int __fastcall Aresta::operator==( const Aresta &a ) const
{
   int Result;

   Result =
      ( v1 == a.v1 && v2 == a.v2 ) ||
      ( v1 == a.v2 && v2 == a.v1 );

   return Result;
}
//-------------------------------------------------
__fastcall Face::Face(
 int nV, Vertice **pV, int nA, Aresta **pA,
 double Eps1, double Sig1, double Mi1, int nCor )
:
NumVertices( nV ),
NumArestas( nA ),
Cor( nCor ),
Epsilon( Eps1 ),
Sigma( Sig1 ),
Mi( Mi1 )
{
 register int i;
 Vetor3D A( pV[1]->p - pV[0]->p );
 Vetor3D B( pV[2]->p - pV[1]->p );
 Normal = A % B;
   Normal.Normaliza();
   //
   D = - Normal * pV[0]->p;
   //
   pVertice = new Vertice *[NumVertices];
   CM = Vetor3D( 0.0, 0.0, 0.0 );
   for( i = 0; i < NumVertices; i++ )
   {
      pVertice[i] = new Vertice( *pV[i] );
      CM += pVertice[i]->p;
   }
   CM /= NumVertices;
   //
   // Copia as Arestas
   //
   Vetor3D Dir;
   pAresta = new Aresta *[NumArestas];
   for( i = 0; i < NumArestas; i++ )
   {
      pAresta[i] = new Aresta( *pA[i] );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      Dir = pAresta[i]->v2.p - pAresta[i]->v1.p;
      // Normal à aresta no plano da face
      pAresta[i]->Nf = Normal % Dir;
      pAresta[i]->Nf.Normaliza();
      pAresta[i]->SignCM =
SIGN( pAresta[i]->Nf * ( CM - pAresta[i]->v1.p ) );
   }
}
//-----------------------------------------------
__fastcall Face::Face( const Face &f )
:
NumVertices( f.NumVertices ),
NumArestas( f.NumArestas ),
Normal( f.Normal ),
D( f.D ),
Cor( f.Cor ),
CM( f.CM ),
Epsilon( f.Epsilon ),
Sigma( f.Sigma ),
Mi( f.Mi )
{
   register int i;
   pVertice = new Vertice *[NumVertices];
   for( i = 0; i < NumVertices; i++ )
   {
      pVertice[i] = new Vertice( *f.pVertice[i] );
   }
   pAresta = new Aresta *[NumArestas];
   for( i = 0; i < NumArestas; i++ )
   {
      pAresta[i] = new Aresta( *f.pAresta[i] );
   }
}
//-------------------------------------------------
__fastcall Face::~Face( void )
{
   if( pVertice != NULL )
   {
      register int i;
      for( i = 0; i < NumVertices; i++ )
      {
         if( pVertice[i] != NULL )
         {
            delete pVertice[i];
         }
      }
      delete [] pVertice;
      pVertice = NULL;
   }
}
//--------------------------------------------------
int __fastcall Face::operator==( const Face &f ) const
{
   register int i;
   register int j;
   int achou;
   int Result;

   Result = true;
   i = 0;
   achou = true;
   while( i < f.NumArestas && achou )
   {
      j = 0;
      achou = false;
      while( j < NumArestas && !achou )
      {
         achou = *f.pAresta[i] == *pAresta[j];
         j++;
      }
      Result &= achou;
      i++;
   }
   return Result;
}
//---------------------------------------------------
int __fastcall Face::operator!=( const Face &f ) const
{
   return !( *this == f );
}
//----------------------------------------------------
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
bool __fastcall Face::BackFace( const Vetor3D &p ) const
{
   bool ret;
   Vetor3D DeltaP( pVertice[0]->p - p );
   ret = DeltaP * Normal >= 0.0;
   return ret;
}
//------------------------------------------------
//
bool __fastcall Face::PertenceAFace( const Vetor3D &p ) const
{
 bool Result;
 register int i;

 Result = true;
 i = 0;
 while( i < NumArestas && Result )
 {
int s = SIGN( pAresta[i]->Nf * ( p - pAresta[i]->v1.p ) );
Result &= s == 0 || s == pAresta[i]->SignCM;
i++;
 }

 return Result;
}

bool __fastcall Face::Intersection(Vetor3D &v,
const Vetor3D &v1, const Vetor3D &v2)
{
Vetor3D r = v2 - v1;
double d;
double w1 = Normal * r ;
double w2 = Normal * v1;
double t; //parametro da reta
if( w1 < Minimo ) return false;
d = Normal * pVertice[0]->p;
t = (d - w2) / w1;
if( t < 0 || t > 1) return false;
v = t*r + v1;
return true;

}
//------------------------------------------------------
//
int __fastcall Face::Contains( const Vertice &v, double tol ) const
{
 int Result;
 Vetor3D p0 = pVertice[1]->p - pVertice[0]->p;
 Vetor3D p1 = pVertice[2]->p - pVertice[0]->p;
 Vetor3D p2 = v.p           - pVertice[0]->p;
 double Triplo = p2 * ( p1 % p0 );
 Triplo = fabs( Triplo );
 Result = Triplo < tol;
 return Result;
}
//--------------------------------------------------------------
int __fastcall Face::Coplanar( const Face &A, double Alfa ) const
{
 int Result;
 double cosAlfa = cos( Alfa );
 double cosBeta = Normal * A.Normal;
   if( cosBeta < 0.0 )
   {
      cosBeta = -cosBeta;
   }
   if( cosBeta >= cosAlfa ) // cos( epsilon ) ~~== 1
   {
#if defined( UNDEFINED )
      register int i;
      register int j;
      i = 0;
      int achou = false;
      while( i < NumVertices && !achou )
      {
         j = 0;
         while( j < A.NumVertices && !achou )
         {
            achou = *pVertice[i] == *A.pVertice[j];
            j++;
         }
         i++;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
      }
      Result = achou;
#endif
      //
      Result = this->Contains( *A.pVertice[0], 1.0e-3 );
   }
   else
   {
      Result = false;
   }
   return Result;
}
//---------------------------------------------------
//
Face * __fastcall Face::MergeFace( const Face &A ) const
{
#if defined( UNDEFINED )
   register int i;
   register int j;
   int achou;

   //
   Vertice **pV = new Vertice *[NumVertices+A.NumVertices];
   //
   for( i = 0; i < NumVertices; i++ )
   {
      pV[i] = new Vertice( *pVertice[i] );
   }
   int TotalVertices = NumVertices;
   //
   for( i = 0; i < A.NumVertices; i++ )
   {
      j = 0;
      achou = false;
      while( j < NumVertices && !achou )
      {
         achou = *A.pVertice[i] == *pVertice[j];
         j++;
      }
      if( !achou )
      {
         pV[TotalVertices] = new Vertice( *A.pVertice[i] );
         TotalVertices++;
      }
   }
   //
   //
   Face *NovaFace = new Face( TotalVertices, pV );
   //
   //    Apaga a face velha ( objeto se suicida )
   //
   delete this;
   //
   //    Desaloca memoria
   //
   for( i = 0; i < TotalVertices; i++ )
   {
      delete pV[i];
   }
   delete [] pV;

   return NovaFace;
#endif
   return NULL;
}
//-------------------------------------------------
void __fastcall Face::MergeTriangulo( const Face &T )
{
   register int i;
   register int j;
   register int achou;
   int nVerticesEmComum;
   //
   Vertice *v;
   nVerticesEmComum = 0;
   for( i = 0; i < T.NumVertices; i++ )
   {
      j = 0;
      achou = false;
      while( j < NumVertices && !achou )
      {
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
         achou = *T.pVertice[i] == *pVertice[j];
         j++;
      }
      if( !achou )
      {
         v = T.pVertice[i];
      }
      else
      {
         nVerticesEmComum++;
      }
   }
   if( nVerticesEmComum < 2 )
   {
      Application->MessageBox(
"Face mal formada - Triangulo não possui aresta em comum.",
"Erro no DXF", MB_OK );
   }
   else if( nVerticesEmComum > 2 )
   {
      Application->MessageBox(
      "Face mal formada - Triangulo já incluído.",
      "Erro no DXF", MB_OK );
      return;
//      throw ESolidoException(
//         "Face mal formada - Triangulo já incluído." );
   }
   //
   int PosIns;
   Vetor3D v1 = pVertice[1]->p - pVertice[0]->p;
   Vetor3D v2 = pVertice[2]->p - pVertice[0]->p;
   Vetor3D v3 = v1 % v2;
   Vetor3D Ref = v3;      // Direcao de referencia
   v2 = v->p - pVertice[0]->p;
   i = 1;
   achou = false;
   while( i < NumVertices && !achou )
   {
      v1 = pVertice[i]->p - pVertice[0]->p;
      v3 = v1 % v2;
      achou = v3 * Ref < 0;
      if( !achou )
      {
         i++;
      }
   }
   PosIns = i;
   Vertice **pV2 = new Vertice *[NumVertices+1];
   for( i = 0; i < PosIns; i++ )
   {
      pV2[i] = pVertice[i];
   }
   pV2[PosIns] = new Vertice( *v );
   NumVertices++;
   for( i = PosIns+1; i < NumVertices; i++ )
   {
      pV2[i] = pVertice[i-1];
   }
   delete [] pVertice;
   pVertice = pV2;
   //
   Aresta **pA2;
   int aj;
   for( i = 0; i < T.NumArestas; i++ )
   {
      j = 0;
      achou = false;
      while( j < NumArestas && !achou )
      {
         achou = *T.pAresta[i] == *pAresta[j];
         if( !achou )
         {
            j++;
         }
      }
      if( !achou )
      {
         //
         //    Acrescenta
         //
         if( !T.pAresta[i]->Visivel )
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
        {
           Application->MessageBox(
   "Atributo de Aresta não confere - deveria ser visível.",
   "Erro no DXF", MB_OK );
        }
        pA2 = new Aresta *[NumArestas+1];
        for( j = 0; j < NumArestas; j++ )
        {
           pA2[j] = pAresta[j];
        }
        pA2[NumArestas] = new Aresta( *T.pAresta[i] );
        NumArestas++;
        delete [] pAresta;
        pAresta = pA2;
      }
      else
      {
        //
        //    Remove
        //
        aj = j;
        if( pAresta[j]->Visivel || T.pAresta[i]->Visivel )
        {
           Application->MessageBox(
"Atributo de Aresta não confere - deveria ser invisível.",
"Erro no DXF", MB_OK );
       }
        pA2 = new Aresta *[NumArestas-1];
        for( j = 0; j < aj; j++ )
        {
           pA2[j] = pAresta[j];
        }
        delete pAresta[aj];
        for( j = aj+1; j < NumArestas; j++ )
        {
           pA2[j-1] = pAresta[j];
        }
        NumArestas--;
        delete [] pAresta;
        pAresta = pA2;
      }
   }
}
//--------------------------------------------
__fastcall Triangulo::Triangulo(
   Vertice **pV, Aresta **pA, double e,
   double s, double m, int nCor )
:
Face( 3, pV, 3, pA, e, s, m, nCor )
{
}
//--------------------------------------------
__fastcall Triangulo::~Triangulo( void )
{
}
//--------------------------------------------
__fastcall Solido::Solido( TCollection *TCol,
int nV, Vertice **pV, int nCor )
:
TCollectionItem( TCol ),
NumVertices( nV ),
Cor( nCor )
{
   register int i;
   pVertice = new Vertice *[NumVertices];
   CM = Vetor3D( 0.0, 0.0, 0.0 );
   for( i = 0; i < NumVertices; i++ )
   {
      pVertice[i] = new Vertice( *pV[i] );
      CM += pVertice[i]->p;
   }
   CM /= NumVertices;
   CreateFaces();
}
//---------------------------------------------
__fastcall Solido::Solido( TCollection *TCol,
   int nV, Vertice **pV,
   int nA, Aresta **pA,
   int nF, Face **pF,
   int nCor )
:
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
TCollectionItem( TCol ),
NumVertices( nV ),
NumArestas( nA ),
NumFaces( nF ),
Cor( nCor )
{
   register int i;
   pVertice = new Vertice *[NumVertices];
   Vetor3D CM( 0.0, 0.0, 0.0 );
   for( i = 0; i < NumVertices; i++ )
   {
      pVertice[i] = new Vertice( *pV[i] );
      CM += pVertice[i]->p;
   }
   CM /= NumVertices;
   pAresta = new Aresta *[NumArestas];
   for( i = 0; i < NumArestas; i++ )
   {
      pAresta[i] = new Aresta( *pA[i] );
   }
   pFace = new Face *[NumFaces];
   for( i = 0; i < NumFaces; i++ )
   {
      pFace[i] = new Face( *pF[i] );
   }
//#if defined( UNDEFINED )
   Vetor3D Aux1( 0.0, 0.0, 0.0 );
   double DotProd;
   for( i = 0; i < NumFaces; i++ )
   {
      Aux1 = pFace[i]->CM - CM;
      DotProd = Aux1 * pFace[i]->Normal;
      if( DotProd < 0.0 )
      {
         pFace[i]->Normal *= -1;
throw ESolidoException( "Face com normal mal-orientada." );
      }
   }
//#endif
}
//--------------------------------------------------
__fastcall Solido::~Solido( void )
{
   register int i;
   //
   // Destroi as faces
   //
   if( pFace != NULL )
   {
      for( i = 0; i < NumFaces; i++ )
      {
         delete pFace[i];
         pFace[i] = NULL;
      }
      delete [] pFace;
      pFace = NULL;
   }
   //
   // Destroi os Vertices
   //
   if( pVertice != NULL )
   {
      for( i = 0; i < NumVertices; i++ )
      {
         delete pVertice[i];
         pVertice[i] = NULL;
      }
      delete [] pVertice;
      pVertice = NULL;
   }
}
//------------------------------------------
void __fastcall Solido::CreateFaces( void )
{
#if defined( UNDEFINED )
  int InitialNumFaces =
      NumVertices * ( NumVertices - 1 ) * ( NumVertices - 2 ) / 6;
   Triangulo **Triangs = new Triangulo *[InitialNumFaces];
   Vertice **TriVertices = new Vertice *[3];
   register int i;
   register int j;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   register int k;
   register int l;
   l = 0;
   for( i = 0; i < NumVertices-2; i++ )
   {
      for( j = i+1; j < NumVertices-1; j++ )
      {
         for( k = j+1; k < NumVertices; k++ )
         {
            TriVertices[0] = pVertice[i];
            TriVertices[1] = pVertice[j];
            TriVertices[2] = pVertice[k];
            Triangs[l] = new Triangulo( TriVertices );
            l++;
         }
      }
   }
   //
   pFace = new Face *[InitialNumFaces];
   i = 0;
   NumFaces = 0;
   int NumTriangsLeft = InitialNumFaces;
   while( NumTriangsLeft > 0 )
   {
      //    Acha o primeiro triangulo nao usado
      j = 0;
      while( Triangs[j] == NULL && j < InitialNumFaces )
      {
         j++;
      }
      if( j < InitialNumFaces )
      {
         //    Cria mais uma face
         pFace[i] = new Face( *Triangs[j] );
         NumFaces++;
         delete Triangs[j];
         Triangs[j] = NULL;
         j++;
         NumTriangsLeft--;
      }
      while( j < InitialNumFaces )
      {
         while( Triangs[j] == NULL && j < InitialNumFaces )
         {
            j++;
         }
         if( j < InitialNumFaces &&
             pFace[i]->Coplanar( *Triangs[j], 0.1 * pi / 180.0 ) )
         {
            // Acrescenta esse triangulo na face atual
            pFace[i] = pFace[i]->MergeFace( *Triangs[j] );
            // Um triangulo a menos
            delete Triangs[j];
            Triangs[j] = NULL;
            NumTriangsLeft--;
         }
         j++;
      }
      i++;
   }
   //
   delete [] Triangs;
   delete [] TriVertices;
#endif
}
//-----------------------------------------------------
__fastcall SolidoCollection::SolidoCollection( TMetaClass *classid )
:
TCollection( classid )
{
}
//---------------------------------------------------
__fastcall SolidoCollection::~SolidoCollection( void )
{
}
//---------------------------------------------------

//--------------------------------------------------
#ifndef SolidoDxfH
#define SolidoDxfH
//--------------------------------------------------
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
#include "Dxf.h"
#include "Solido.h"
//-------------------------------------------------
class SolidoDxfCollection : public SolidoCollection
{
private:
public:
   dxf DxfReader;
public:
   __fastcall SolidoDxfCollection(
      TMetaClass *classid, const AnsiString &DxfFilename );
   __fastcall ~SolidoDxfCollection( void );
};
//---------------------------------------------------------
#endif

//--------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop

#define __SOLIDODXF_CPP__

#include "SolidoDxf.h"

//-------------------------------------------------------
#pragma package(smart_init)
//-------------------------------------------------------
__fastcall SolidoDxfCollection::SolidoDxfCollection(
   TMetaClass *classid, const AnsiString &DxfFileName )
:
SolidoCollection( classid )
{
   DxfReader.ReadDxfFile( this, DxfFileName );
}
//--------------------------------------------------------
__fastcall SolidoDxfCollection::~SolidoDxfCollection( void )
{
}
//--------------------------------------------------------


//--------------------------------------------------------
#ifndef TIFieldH
#define TIFieldH
//--------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <gl\gl.h>
#include "fViewdxf.h"
//---------------------------------------------------------
class TIField : public TForm
{
__published: // IDE-managed Components
void __fastcall FormCreate(TObject *Sender);
void __fastcall FormPaint(TObject *Sender);
void __fastcall FormDestroy(TObject *Sender);
private: // User declarations

 HWND hWnd;
   HDC hDC;
public: // User declarations
  void __fastcall SetPixelFormatDescriptor( void )   ;
void Setup(int x1, int y1, int x2, int y2, int z);
__fastcall TIField(TComponent* Owner);
};
//---------------------------------------------------------
extern PACKAGE TIField *IField;
//---------------------------------------------------------
#endif

//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "TIField.h"
#include <gl/glu.h>
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TIField *IField;
//---------------------------------------------------------
__fastcall TIField::TIField(TComponent* Owner)
: TForm(Owner)
{
hWnd = NULL;
}
//----------------------------------------------------------
void __fastcall TIField::FormCreate(TObject *Sender)
{
 if(hWnd) return;
 hWnd = Handle;
 hDC = GetDeviceContext( hWnd );
}
//---------------------------------------------------------

void __fastcall TIField::SetPixelFormatDescriptor( void )
{
 PIXELFORMATDESCRIPTOR pfd = {
sizeof(PIXELFORMATDESCRIPTOR),
1,
PFD_DRAW_TO_WINDOW |
PFD_SUPPORT_OPENGL |
PFD_DOUBLEBUFFER,
PFD_TYPE_RGBA,
24,
0, 0, 0, 0, 0, 0,
0, 0,
0, 0, 0, 0, 0,
32,
0,
0,
PFD_MAIN_PLANE,
0,
0, 0, 0
 };
 int PixelFormat = ChoosePixelFormat( hDC, &pfd );
 SetPixelFormat( hDC, PixelFormat, &pfd );
}

void TIField::Setup(int x1, int y1, int x2, int y2, int z)
{
 SetPixelFormatDescriptor();
 int width = ClientWidth;
 int height = ClientHeight;

 if( wglMakeCurrent( hDC, frmTracado->hRC ) == false )
 {
ShowMessage( "Could not MakeCurrent" );
return;
 }
 glViewport( 0, 0, width, height);
 glMatrixMode( GL_MODELVIEW );
 glPushMatrix();
 glLoadIdentity();

 glMatrixMode( GL_PROJECTION );
 glPushMatrix();
 glLoadIdentity();

 glOrtho( x1, x2, y1, y2, -100 , 100);
 Paint();

 glMatrixMode( GL_MODELVIEW );
 glPopMatrix();

 glMatrixMode( GL_PROJECTION );
 glPopMatrix();
}

void __fastcall TIField::FormPaint(TObject *Sender)
{
 frmTracado->RenderScene();
 SwapBuffers( hDC );
}
//---------------------------------------------------
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
void __fastcall TIField::FormDestroy(TObject *Sender)
{
 ReleaseDC( hWnd, hDC );
 hWnd = NULL;
}
//--------------------------------------------------

//--------------------------------------------------
#ifndef TIFieldAttH
#define TIFieldAttH
//--------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
//-------------------------------------------------
class TIFieldAtt : public TForm
{
__published: // IDE-managed Components
TGroupBox *GroupBox1;
TEdit *Z;
TEdit *X;
TEdit *Width;
TEdit *Y;
TEdit *Height;
TLabel *Label1;
TLabel *Label2;
TLabel *Label3;
TLabel *Label4;
TLabel *Label5;
TBitBtn *BitBtn1;
TBitBtn *BitBtn2;
TLabel *Label6;
TLabel *Label7;
TLabel *Label8;
void __fastcall BitBtn1Click(TObject *Sender);
void __fastcall WidthKeyUp(TObject *Sender, WORD
&Key, TShiftState Shift);
private: // User declarations
public: // User declarations
double fZ;
double fXmin;
double fXmax;
double fYmin;
double fYmax;
__fastcall TIFieldAtt(TComponent* Owner);
};
//------------------------------------------------
extern PACKAGE TIFieldAtt *IFieldAtt;
//------------------------------------------------
#endif

//------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include "TIFieldAtt.h"
//------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TIFieldAtt *IFieldAtt;
//------------------------------------------------

#define NEARESTPOW(x) (pow(2,ceil((log(x)/log(2)))))


__fastcall TIFieldAtt::TIFieldAtt(TComponent* Owner)
: TForm(Owner)
{
}
//------------------------------------------------
void __fastcall TIFieldAtt::BitBtn1Click(TObject *Sender)
{
fZ = atof(Z->Text.c_str());
fXmin = atof(X->Text.c_str()) -
            NEARESTPOW(atof(Width->Text.c_str()))/2;
fXmax = atof(X->Text.c_str()) +
            NEARESTPOW(atof(Width->Text.c_str()))/2 - 1;
fYmin = atof(Y->Text.c_str()) -
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
            NEARESTPOW(atof(Height->Text.c_str()))/2;
fYmax = atof(Y->Text.c_str()) +
            NEARESTPOW(atof(Height->Text.c_str()))/2 - 1;
}
//----------------------------------------------------
void __fastcall TIFieldAtt::WidthKeyUp(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
Height->Text = Width->Text;
}
//----------------------------------------------------

//-----------------------------------------------------
#include <vcl\vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>

#include "filehdr.h"
#include "specular.h"
#include "Dlg_AntennaManager.h"
#include "Main.h"
#include "Dlg_Hierarchy.h"
#include "Class_PropagationModel.h"
#include "Class_CellBuilder.h"

#pragma hdrstop

#include "Utilities.h"


extern Specular SpecPts;

// --------------------------------------------------|
char *DecToDegMinSec(double dec)
{
static char DecTxt[20];
double Deg,Min,Sec;

Deg = dec > 0 ? floor(dec) : ceil(dec);
dec = fabs(dec - Deg)*60;
Min = floor(dec);
Sec = (dec - Min)*60.;
sprintf(DecTxt,"%3.0f°%2.0f'%2.2f\"",Deg,Min,Sec);
return(DecTxt);
}
// ------------------------------------------------|
void OutOfMemory(int errn)
{
char txt[20];
sprintf(txt,"Error %d",errn);
Application->MessageBox("Out of memory.
Program execution suspended.",txt,MB_OK);
exit(0);
}
// --------------------------------------------|
// cuts off the path of a filename
void GetRealFileName(char *fullname,char *fname)
{
char *pos = strrchr(fullname,'\\');  // find last <\>
strcpy(fname,pos+1);
}

double Length2mToPt(int m)
{
double d = m*9.0*tfh.nPx/(1000000.0*(tfh.ArqLongE-tfh.ArqLongW));
return(d*d);
}

double DistancePtToKm(int Pt)
{
return (Pt*1000.0*(tfh.ArqLongE-tfh.ArqLongW)/9.0/tfh.nPx);
}

double DistancePtToM(double Pt)
{
return (Pt*1000000.0*(tfh.ArqLongE-tfh.ArqLongW)/9.0/tfh.nPx);
}

int DistanceKmToPt(double km)
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
{
return(km*9.0*tfh.nPx/(1000.0*(tfh.ArqLongE-tfh.ArqLongW)));
}

int GetHorzDistToAntPt(int x,int y)
{
return ((int)sqrt(((double)x - CellBuilder->WrkSite->x)*((double)x -
CellBuilder->WrkSite->x) + ((double)y -
CellBuilder->WrkSite->y)*((double)y -
CellBuilder->WrkSite->y)));
}

double GetHorzDistToAntM(int x,int y)
{
return (DistancePtToM(GetHorzDistToAntPt(x,y)));
}

// sqrt(x2 + y2)
float GetHorzDistToStartpointM(float x,float y)
{
switch (SpecPts.Track)
       {
case 1: return (GetHorzDistToAntM(x,y));
case 2: return (sqrt((SpecPts.Pt1.x - x)*(SpecPts.Pt1.x - x) +
       (SpecPts.Pt1.y - y)*(SpecPts.Pt1.y - y)));
       case 3: return (GetHorzDistToAntM(x,y));
       case 4: return (sqrt((SpecPts.Pt2.x - x)*
(SpecPts.Pt2.x - x) + (SpecPts.Pt2.y - x)*(SpecPts.Pt2.y - x)));
       }
return (-1.);
}

inline float GetTotalDistanceToAntenna(float x,float y)
{
return (sqrt((x - CellBuilder->WrkSite->x)*
       (x - CellBuilder->WrkSite->x) +
       (y - CellBuilder->WrkSite->y)*
       (y - CellBuilder->WrkSite->y)));
}

void ErrorMsg(char *text,char *title)
{
Application->MessageBox(text,title,MB_OK|MB_ICONEXCLAMATION);
}


void FatalMsg(char *text)
{
Application->MessageBox(text,"Fatal Error",MB_OK|MB_ICONSTOP);
exit(0);
}

void DecToDegMinSec(AnsiString &dmsString,float d)
{
char DmsTxt[20];
double Deg,Min,Sec;

Deg = d > 0 ? floor(d) : ceil(d);
d = fabs(d - Deg)*60;
Min = floor(d);
byte b[] = {0,13,22,15,17,1,16,0,17,2,0,11};
Sec = (d - Min)*60.;
sprintf(DmsTxt,"%3.0f°%2.0f'%2.2f\"",Deg,Min,Sec);
dmsString = DmsTxt;
}

void Draw90DegText(TCanvas *C,int x,int y,
      const AnsiString &s)
{
HFONT hfnt, hfntPrev;

/* Allocate memory for a LOGFONT structure. */
PLOGFONT plf = new LOGFONT;

/* Specify a font typeface name and weight. */
lstrcpy(plf->lfFaceName, "Arial");
plf->lfWeight = FW_NORMAL;
plf->lfHeight = 10;

// Draw the text:
plf->lfEscapement = 900;  // angle
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
hfnt     = CreateFontIndirect(plf);
hfntPrev = SelectObject(C->Handle,hfnt);
TextOut(C->Handle,x,y,s.c_str(), s.Length());
SelectObject(C->Handle,hfntPrev);
DeleteObject(hfnt);

delete plf;
}

int IntDivisionRoundToNearest(int nom,int denom)
{
float f = (float)nom/(float)denom;
return (nom/denom) + (f - floor(f) < 0.5 ? 0 : 1);
}


void Regression(const double x[],const double y[],
        const int size,double *k, double *d)
{
double xsum = 0.,ysum = 0.,xxsum = 0.,xysum = 0.;

for (int i = 0;i < size;i++)
    {
    xsum  += x[i];
    ysum  += y[i];
    xysum += x[i]*y[i];
    xxsum += x[i]*x[i];
    }
*k = (xysum - (xsum*ysum)/(double)size) /
     (xxsum - (xsum*xsum)/(double)size);
*d = (ysum - *k*xsum)/(double)size;
}

//    POINT == POINT
bool operator==(POINT p1,POINT p2)
{
return (p1.x == p2.x && p1.y == p2.y);
}

//    POINT != POINT
bool operator!=(POINT p1,POINT p2)
{
return (!(p1 == p2));
}
//    POINT constructor TPOINT()
POINT TPOINT(int x,int y)
{
POINT p = {x,y};
return p;
}

//---------------------------------------|
float GetAreaEquivalent(int num_pts)
{
int   baseline_pt = sqrt((double)num_pts);
float baseline_km = DistancePtToKm(baseline_pt);
return baseline_km*baseline_km;
}
// ----------------------------------|
bool HandleMissingFile(String msg,String title,String &fname)
{
if (!FileExists(fname))
   {
msg += String("\n") + fname + String("\nBrowse for file?");
if (IDYES == Application->MessageBox(msg.c_str(),
     title.c_str(),MB_YESNO))
      {
 auto_ptr<TOpenDialog> od(new TOpenDialog(Application));
 od->FileName    = fname;
 od->Filter      = "Data File (*.*)|*.*";
 od->DefaultExt  = "*.*";
 od->Title       = title;
 od->Options << ofFileMustExist;
 if (od->Execute())
         {
         fname = od->FileName;
         return true;
         }
      }
   return false;
   }
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
return true;
}

// ----------------------------------|
void SpectralColor(float Int,byte &r,byte &g,byte &b)
{
#if 1
// linear scale
if (Int < 0.)
   { r = 0; g = 0; b = 255; }
else if (Int < 0.25)
   { r = 0; g = Int*4.*255; b = 255; }
else if (Int < 0.5)
   { r = 0; g = 255; b = 255-(Int-0.25)*4.*255; }
else if (Int < 0.75)
   { r = (Int-0.5)*4.*255; g = 255; b = 0; }
else if (Int < 1.)
   { r = 255; g = 255-(Int-0.75)*4*255; b = 0; }
else
   { r = 255; g = 0; b = 0; }
// logarithmic scale:
#else
Int = log10(Int + 1.)/0.30103; // /log10(2.)

if (Int < 0.25)
   { r = 0; g = Int*4.*255; b = 255; }
else if (Int < 0.5)
   { r = 0; g = 255; b = 255-(Int-0.25)*4.*255; }
else if (Int < 0.75)
   { r = (Int-0.5)*4.*255; g = 255; b = 0; }
else
   { r = 255; g = 255-(Int-0.75)*4*255; b = 0; }
#endif
}

//-----------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
//-------------------------------------------
class FileObject{
public:
FILE * fptr;
FileObject( const char *name, const char *mode );
~FileObject();
int printf( char *fmt, ... ) const;
int scanf( char *fmt, ... ) const;
size_t fread( void *ptr, size_t size, size_t n ) const;
int fend( void ) const;
private:
bool opened;
};
//------------------------------------------------------

//------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
#include "FileIO.h"
//-----------------------------------------------------
FileObject::FileObject( const char *name, const char *mode )
{
   fptr = fopen( name, mode );
   opened = fptr != NULL;
}
//-------------------------------------------------------
FileObject::~FileObject()
{
   if( opened )
   {
      fclose( fptr );
   }
}
//-------------------------------------------------
int FileObject::printf( char *fmt, ... ) const
{
   va_list( marker );
   int ret;
   va_start( marker, fmt );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   ret = vfprintf( fptr, fmt, marker );
   va_end( marker );
   return ret;
}
//---------------------------------------------------
int FileObject::scanf( char *fmt, ... ) const
{
   va_list( marker );
   int ret;
   va_start( marker, fmt );
   ret = vfscanf( fptr, fmt, marker );
   va_end( marker );
   return ret;
}
//----------------------------------------------------------
size_t FileObject::fread( void *ptr, size_t size, size_t n ) const
{
   size_t ret;
   ret = ::fread( ptr, size, n, fptr );
   return ret;
}
//--------------------------------------------------
int FileObject::fend( void ) const
{
   int ret;
   if( opened )
   {
      ret = feof( fptr ) || ferror( fptr );
   }
   else
   {
      ret = true;
   }
   return ret;
}
//----------------------------------------------------------
```

B.2
Códigos correspondentes ao método FDTD

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "CylinderPML.h"
#include "fonte.h"
#include "malha.h"
#include "material.h"
 static double PI   = acos(-1);
  static double cc   = 2.9979245e+008;
  static double muz  = 4.0*PI*1.0e-007;
  static double epsz = 1.0/(pow(cc,2)*muz);
  static double etaz = sqrt(muz/epsz);
void main()
{
 double **ex;
 double **ey;
 double **hz;
     double **exbcf;
 double **eybcf;
       double **hzxbcf;  // Split Field
   double **hzybcf;
     double **exbcb;
 double **eybcb;
   double **hzxbcb;  // Split Field
   double **hzybcb;
    double **exbcl;
 double **eybcl;
       double **hzxbcl;  // Split Field
   double **hzybcl;
     double **exbcr;
 double **eybcr;
       double **hzxbcr;  // Split Field
   double **hzybcr;
   double ca[2], cb[2], da[2],db[2];
      double eaf, haf;
  int i,j,n;
  ex=zeros(grid.ie,grid.jb);
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
  exbcf=zeros(grid.iefbc,grid.jebc);
  exbcb=zeros(grid.iefbc,grid.jbbc);
  exbcl=zeros(grid.iebc,grid.jb);
  exbcr=zeros(grid.iebc,grid.jb);
  ey=zeros(grid.ib,grid.je);
  eybcf=zeros(grid.ibfbc,grid.jebc);
  eybcb=zeros(grid.ibfbc,grid.jebc);
  eybcl=zeros(grid.iebc,grid.je);
  eybcr=zeros(grid.ibbc,grid.je);
  hz=zeros(grid.ie,grid.je);
  hzxbcf=zeros(grid.iefbc,grid.jebc);
  hzxbcb=zeros(grid.iefbc,grid.jebc);
  hzxbcl=zeros(grid.iebc,grid.je);
  hzxbcr=zeros(grid.iebc,grid.je);
  hzybcf=zeros(grid.iefbc,grid.jebc);
  hzybcb=zeros(grid.iefbc,grid.jebc);
  hzybcl=zeros(grid.iebc,grid.je);
  hzybcr=zeros(grid.iebc,grid.je);
   fonte pSGauss(3.0e-3,  15, 25, 5.0e+9, 160.0e-12);
     double *source;
     source=zeros(grid.nmax);
     pSGauss.pulso_gaussiano(source);
for(i=0; i<media; i++){
    eaf=pSGauss.dt*sig[i]/(2.0*epsz*eps[i]);
  ca[i]=(1.0-eaf)/(1.0+eaf);
  cb[i]=pSGauss.dt/epsz/eps[i]/pSGauss.dx/(1.0+eaf);
    haf=pSGauss.dt*sim[i]/(2.0*muz*mur[i]);
      da[i]=(1.0-haf)/(1.0+haf);
      db[i]=pSGauss.dt/muz/mur[i]/pSGauss.dx/(1.0+haf);
    }

double **caex,**cbex;
double **caey,**cbey;
double **dahz,**dbhz;

    caex = zeros(grid.ie,grid.jb);
    cbex = zeros(grid.ie,grid.jb);
    caey = zeros(grid.ib,grid.je);
    cbey = zeros(grid.ib,grid.je);
    dahz = zeros(grid.ie,grid.je);
    dbhz = zeros(grid.ie,grid.je);
    for(i=0;i<grid.ie;i++){
for(j=0;j<grid.jb;j++){
caex[i][j] = ca[0];
cbex[i][j] = cb[0];
}
}
    for(i=0;i<grid.ib;i++){
for(j=0;j<grid.je;j++){
caey[i][j] = ca[0];
cbey[i][j] = cb[0];
        }
}
    for(i=0;i<grid.ie;i++){
for(j=0;j<grid.jb;j++){
dahz[i][j] = da[0];
dbhz[i][j] = db[0];
        }
}
 double diam = 20.0;
 double rad  = (diam/2.0);
 double icenter = (4*grid.ie/5);
 double jcenter = (grid.je/2);
 double dist2;

 for(i=1;i<=grid.ie;i++){
 for(j=1;j<=grid.je;j++){
dist2 = pow((i+0.5-icenter),2) + pow((j-jcenter),2);
//cout<<"\n dist2 = "<<dist2;
if(dist2<=pow(rad,2)){
caex[i-1][j-1]=ca[1];
//cout<<"\n caex["<<i-1<<"]["<<j-1<<" ] = "<<caex[i-1][j-1];
cbex[i-1][j-1]=cb[1];
}
dist2 = pow((i-icenter),2) + pow((j+0.5-jcenter),2);
if(dist2<=pow(rad,2)){
caey[i-1][j-1]=ca[1];
cbey[i-1][j-1]=cb[1];
        }
 }
 }
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
//================================================
 double delbc, sigmam, bcfactor;
 int orderbc=2;
 double rmax=1.0e-7;
 double y1,y2,x1,x2;
 double ca1,cb1,da1, db1;
 double sigmay,sigmays;
 double sigmax,sigmaxs;

 delbc   = grid.iebc*pSGauss.dx;
 sigmam  =-log(rmax)*(orderbc+1)/(2*etaz*delbc);
 bcfactor= sigmam/(pSGauss.dx*(orderbc+1)*
          (pow(delbc,orderbc)));

 double **caexbcf,**cbexbcf;
 caexbcf=zeros(grid.iefbc, grid.jebc);
 cbexbcf=zeros(grid.iefbc, grid.jebc);
 double **caexbcb,**cbexbcb;
 caexbcb=zeros(grid.iefbc, grid.jbbc);
 cbexbcb=zeros(grid.iefbc, grid.jbbc);
 double **caexbcl,**cbexbcl;
 caexbcl=zeros(grid.iebc, grid.jb);
 cbexbcl=zeros(grid.iebc, grid.jb);
 double **caexbcr,**cbexbcr;
 caexbcr=zeros(grid.iebc, grid.jb);
 cbexbcr=zeros(grid.iebc, grid.jb);
 double **caeybcf,**cbeybcf;
 caeybcf=zeros(grid.ibfbc, grid.jebc);
 cbeybcf=zeros(grid.ibfbc, grid.jebc);
 double **caeybcb,**cbeybcb;
 caeybcb=zeros(grid.ibfbc, grid.jebc);
 cbeybcb=zeros(grid.ibfbc, grid.jebc);
 double **caeybcl,**cbeybcl;
 caeybcl=zeros(grid.iebc, grid.je);
 cbeybcl=zeros(grid.iebc, grid.je);
 double **caeybcr,**cbeybcr;
 caeybcr=zeros(grid.ibbc, grid.je);
 cbeybcr=zeros(grid.ibbc, grid.je);
 double **dahzxbcf,**dbhzxbcf;
 dahzxbcf=zeros(grid.iefbc, grid.jebc);
 dbhzxbcf=zeros(grid.iefbc, grid.jebc);
 double **dahzxbcb,**dbhzxbcb;
 dahzxbcb=zeros(grid.iefbc, grid.jebc);
 dbhzxbcb=zeros(grid.iefbc, grid.jebc);
 double **dahzxbcl,**dbhzxbcl;
 dahzxbcl=zeros(grid.iebc, grid.je);
 dbhzxbcl=zeros(grid.iebc, grid.je);
 double **dahzxbcr,**dbhzxbcr;
 dahzxbcr=zeros(grid.iebc, grid.je);
 dbhzxbcr=zeros(grid.iebc, grid.je);
 double **dahzybcf,**dbhzybcf;
 dahzybcf=zeros(grid.iefbc, grid.jebc);
 dbhzybcf=zeros(grid.iefbc, grid.jebc);
 double **dahzybcb,**dbhzybcb;
 dahzybcb=zeros(grid.iefbc, grid.jebc);
 dbhzybcb=zeros(grid.iefbc, grid.jebc);
 double **dahzybcl,**dbhzybcl;
 dahzybcl=zeros(grid.iebc, grid.je);
 dbhzybcl=zeros(grid.iebc, grid.je);
 double **dahzybcr,**dbhzybcr;
 dahzybcr=zeros(grid.iebc, grid.je);
  dbhzybcr=zeros(grid.iebc, grid.je);
 for(i=0; i<grid.iefbc; i++){
   caexbcf[i][0] = 1.0;
 cbexbcf[i][0] = 0.0;
 }
 for(j=1; j<grid.jebc; j++){
 y1 = (grid.jebc - (j+1) + 1.5)*pSGauss.dx;
 y2 = (grid.jebc - (j+1) + 0.5)*pSGauss.dx;
 sigmay = bcfactor*(
              pow(y1,(orderbc+1)) -
                      pow(y2,(orderbc+1))
                     );
  ca1 = exp(-sigmay*pSGauss.dt/epsz);
cb1 = (1.0 - ca1)/(sigmay*pSGauss.dx);
for(i=0;i<grid.iefbc;i++){
caexbcf[i][j] = ca1;
        cbexbcf[i][j] = cb1;

    }
 }
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
 sigmay = bcfactor*pow((0.5*pSGauss.dx),(orderbc+1));
 ca1 = exp(-sigmay*pSGauss.dt/epsz);
 cb1 = (1-ca1)/(sigmay*pSGauss.dx);
 for(i=0; i<grid.ie; i++){
 caex[i][0] = ca1;
 cbex[i][0] = cb1;
 }
 for(i=0; i<grid.iebc; i++){
 caexbcl[i][0] = ca1;
     cbexbcl[i][0] = cb1;
 caexbcr[i][0] = ca1;
 cbexbcr[i][0] = cb1;
 }
for(j=0;j<grid.jebc;j++){
y1=(grid.jebc-(j+1)+1)*pSGauss.dx;
y2=(grid.jebc-(j+1))*pSGauss.dx;
sigmay=bcfactor*(
               pow(y1,(orderbc+1))-
   pow(y2,(orderbc+1))
  );
sigmays=sigmay*(muz/epsz);
da1=exp(-sigmays*pSGauss.dt/muz);
db1=(1-da1)/(sigmays*pSGauss.dx);
for(i=0;i<grid.iefbc;i++){
dahzybcf[i][j]=da1;
dbhzybcf[i][j]=db1;
}
for(i=0;i<grid.ibfbc;i++){
caeybcf[i][j]=ca[0];
cbeybcf[i][j]=cb[0];
}
    for(i=0;i<grid.iefbc;i++){
dahzxbcf[i][j]=da[0];
dbhzxbcf[i][j]=db[0];
}
}
  for(i=0;i<grid.iefbc;i++){
caexbcb[i][grid.jbbc-1]=1.0;
cbexbcb[i][grid.jbbc-1]=0.0;
  }
  for(j=1;j<grid.jebc;j++){
      y1 = ((j+1)-0.5)*pSGauss.dx;
  y2 = ((j+1)-1.5)*pSGauss.dx;
      sigmay = bcfactor*(
                 pow(y1,(orderbc+1)) -
                 pow(y2,(orderbc+1))
);
      ca1 = exp(-sigmay*pSGauss.dt/epsz);
  cb1 = (1-ca1)/(sigmay*pSGauss.dx);
  for(i=0;i<grid.iefbc;i++){
  caexbcb[i][j]=ca1;
  cbexbcb[i][j]=cb1;
      }
  }
  sigmay = bcfactor*pow((0.5*pSGauss.dx),(orderbc+1));
  ca1=exp(-sigmay*pSGauss.dt/epsz);
  cb1=(1-ca1)/(sigmay*pSGauss.dx);
  for(i=0;i<grid.ie;i++){
  caex[i][grid.jb-1] = ca1;
  cbex[i][grid.jb-1] = cb1;
  }
  for(i=0;i<grid.iebc;i++){
  caexbcl[i][grid.jb-1]=ca1;
  cbexbcl[i][grid.jb-1]=cb1;
  caexbcr[i][grid.jb-1]=ca1;
      cbexbcr[i][grid.jb-1]=cb1;
  }
  for(j=0;j<grid.jebc;j++){
  y1 = (j+1)*pSGauss.dx;
  y2 = ((j+1)-1)*pSGauss.dx;
  sigmay = bcfactor*(
                 pow(y1,(orderbc+1)) -
 pow(y2,(orderbc+1))
);
     sigmays = sigmay*(muz/epsz);

     da1 = exp(-sigmays*pSGauss.dt/muz);
 db1 = (1-da1)/(sigmays*pSGauss.dx);
 for(i=0;i<grid.iefbc;i++){
        dahzybcb[i][j]=da1;
        dbhzybcb[i][j]=db1;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
     }
 for(i=0;i<grid.ibfbc;i++){
caeybcb[i][j]=ca[0];
        cbeybcb[i][j]=cb[0];
     }
 for(i=0;i<grid.iefbc;i++){
        dahzxbcb[i][j]=da[0];
        dbhzxbcb[i][j]=db[0];
     }
  }
  for(j=0;j<grid.je;j++){
  caeybcl[0][j] = 1.0;
  cbeybcl[0][j] = 0.0;
  }
  for(i=1;i<grid.iebc;i++){
     x1 = (grid.iebc - (i+1) + 1.5)*pSGauss.dx;
 x2 = (grid.iebc - (i+1) + 0.5)*pSGauss.dx;
 sigmax = bcfactor*(
              pow(x1,(orderbc+1)) -
pow(x2,(orderbc+1))
   );
 ca1 = exp(-sigmax*pSGauss.dt/epsz);
     cb1 = (1 - ca1)/(sigmax*pSGauss.dx);
     for(j=0;j<grid.je;j++){
 caeybcl[i][j]=ca1;
         cbeybcl[i][j]=cb1;
     }
 for(j=0;j<grid.jebc;j++){
 caeybcf[i][j]=ca1;
 cbeybcf[i][j]=cb1;
 caeybcb[i][j]=ca1;
 cbeybcb[i][j]=cb1;
 }
  }
  sigmax = bcfactor*(pow((0.5*pSGauss.dx),(orderbc+1)));
  ca1 = exp(-sigmax*pSGauss.dt/epsz);
  cb1 = (1 - ca1)/(sigmax*pSGauss.dx);
  for(j=0;j<grid.je;j++){
  caey[0][j]=ca1;
  cbey[0][j]=cb1;
  }
  for(j=0;j<grid.jebc;j++){
  caeybcf[grid.ibbc-1][j]=ca1;
  cbeybcf[grid.ibbc-1][j]=cb1;
      caeybcb[grid.ibbc-1][j]=ca1;
  cbeybcb[grid.ibbc-1][j]=cb1;
  }
  for(i=0;i<grid.iebc;i++){
  x1 = (grid.iebc-(i+1)+1)*pSGauss.dx;
  x2 = (grid.iebc-(i+1))*pSGauss.dx;
  sigmax = bcfactor*(
              pow(x1,(orderbc+1)) -
pow(x2,(orderbc+1))
);
  sigmaxs=sigmax*(muz/epsz);
  da1 = exp(-sigmaxs*pSGauss.dt/muz);
  db1 = (1-da1)/(sigmaxs*pSGauss.dx);
  for(j=0;j<grid.je;j++){
      dahzxbcl[i][j]=da1;
  dbhzxbcl[i][j]=db1;
      }
      for(j=0;j<grid.jebc;j++){
         dahzxbcf[i][j]=da1;
         dbhzxbcf[i][j]=db1;
         dahzxbcb[i][j]=da1;
         dbhzxbcb[i][j]=db1;
      }
  for(j=1;j<grid.je;j++){
          caexbcl[i][j]=ca[0];
  cbexbcl[i][j]=cb[0];
      }
  for(j=0;j<grid.je;j++){
  dahzybcl[i][j]=da[0];
  dbhzybcl[i][j]=db[0];
  }
  }
 for(j=0;j<grid.je;j++){
  caeybcr[grid.ibbc-1][j] = 1.0;
  cbeybcr[grid.ibbc-1][j] = 0.0;
  }
  for(i=1;i<grid.iebc;i++){
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
     x1 = ((i+1)-0.5)*pSGauss.dx;
     x2 = ((i+1)-1.5)*pSGauss.dx;
 sigmax = bcfactor*(
               pow(x1,(orderbc+1))-
                       pow(x2,(orderbc+1))
);
 ca1=exp(-sigmax*pSGauss.dt/epsz);
 cb1=(1-ca1)/(sigmax*pSGauss.dx);
     for(j=0;j<grid.je;j++){
 caeybcr[i][j]=ca1;
 cbeybcr[i][j]=cb1;
 }
     for(j=0;j<grid.jebc;j++){
       caeybcf[i+grid.iebc+grid.ie-1][j]=ca1;
       cbeybcf[i+grid.iebc+grid.ie-1][j]=cb1;
       caeybcb[i+grid.iebc+grid.ie-1][j]=ca1;
       cbeybcb[i+grid.iebc+grid.ie-1][j]=cb1;
 }
  }
  sigmax = bcfactor*pow((0.5*pSGauss.dx),(orderbc+1));
  ca1 = exp(-sigmax*pSGauss.dt/epsz);
  cb1 = (1-ca1)/(sigmax*pSGauss.dx);
  for(j=0;j<grid.je;j++){
  caey[grid.ib-1][j]=ca1;
  cbey[grid.ib-1][j]=cb1;
  }
  for(j=0;j<grid.jebc;j++){
  caeybcf[grid.iebc+grid.ib-1][j]=ca1;
  cbeybcf[grid.iebc+grid.ib-1][j]=cb1;
      caeybcb[grid.iebc+grid.ib-1][j]=ca1;
  cbeybcb[grid.iebc+grid.ib-1][j]=cb1;
  }
  for(i=0;i<grid.iebc;i++){
  x1=(i+1)*pSGauss.dx;
  x2=((i+1)-1)*pSGauss.dx;
  sigmax  = bcfactor*(
                pow(x1,(orderbc+1))-
  pow(x2,(orderbc+1))
  );
      sigmaxs = sigmax*(muz/epsz);
  da1=exp(-sigmaxs*pSGauss.dt/muz);
  db1=(1-da1)/(sigmaxs*pSGauss.dx);
      for(j=0;j<grid.je;j++){
  dahzxbcr[i][j] = da1;
          dbhzxbcr[i][j] = db1;
      }
      for(j=0;j<grid.jebc;j++){
     dahzxbcf[i+grid.ie+grid.iebc-1][j]=da1;
         dbhzxbcf[i+grid.ie+grid.iebc-1][j]=db1;
         dahzxbcb[i+grid.ie+grid.iebc-1][j]=da1;
         dbhzxbcb[i+grid.ie+grid.iebc-1][j]=db1;
  }
  for(j=1;j<grid.je;j++){
       caexbcr[i][j]=ca[0];
         cbexbcr[i][j]=cb[0];
     }
  for(j=0;j<grid.je;j++){
         dahzybcr[i][j]=da[0];
         dbhzybcr[i][j]=db[0];
  }
  }
for(n=0;n<grid.nmax;n++){
   for(i=0;i<grid.ie;i++)
   for(j=1;j<grid.je;j++)
ex[i][j]=caex[i][j]*ex[i][j]+cbex[i][j]*(hz[i][j]-hz[i][j-1]);
   for(i=1;i<grid.ie;i++)
     for(j=0;j<grid.je;j++)
ey[i][j]=caey[i][j]*ey[i][j]+cbey[i][j]*(hz[i-1][j]-hz[i][j]);
  for(i=0;i<grid.iefbc;i++)
  for(j=1;j<grid.jebc;j++)
  exbcf[i][j]=caexbcf[i][j]*exbcf[i][j]-
              cbexbcf[i][j]*(
             hzxbcf[i][j-1] + hzybcf[i][j-1]-
             hzxbcf[i][j]   - hzybcf[i][j]
             );
  for(i=0;i<grid.ie;i++)
     ex[i][0]=caex[i][0]*ex[i][0]-
                 cbex[i][0]*(
           hzxbcf[i+grid.iebc][grid.jebc-1] +
   hzybcf[i+grid.iebc][grid.jebc-1] -
               hz[i][0]
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
             );
  for(i=0;i<grid.iefbc;i++)
  for(j=1;j<grid.jebc-1;j++)
          exbcb[i][j]=caexbcb[i][j]*exbcb[i][j] -
              cbexbcb[i][j]*(
         hzxbcb[i][j-1] + hzybcb[i][j-1] -
 hzxbcb[i][j] - hzybcb[i][j]
 );
  for(i=0;i<grid.ie;i++)
ex[i][grid.jb-1]=caex[i][grid.jb-1]*ex[i][grid.jb-1] -
                cbex[i][grid.jb-1]*(
             hz[i][grid.jb-2] -
         hzxbcb[i+grid.iebc][0] -
         hzybcb[i+grid.iebc][0]
         );
  for(i=0;i<grid.iebc;i++)
  for(j=1;j<grid.je;j++)
  exbcl[i][j]=caexbcl[i][j]*exbcl[i][j] -
              cbexbcl[i][j]*(
            hzxbcl[i][j-1]+hzybcl[i][j-1] -
hzxbcl[i][j] - hzybcl[i][j]
);
  for(i=0;i<grid.iebc;i++)
  exbcl[i][0]=caexbcl[i][0]*exbcl[i][0] -
              cbexbcl[i][0]*(
    hzxbcf[i][grid.jebc-1]+hzybcf[i][grid.jebc-1] -
    hzxbcl[i][0] - hzybcl[i][0]
 );
  for(i=0;i<grid.iebc;i++)
exbcl[i][grid.jb-1] = caexbcl[i][grid.jb-1]*exbcl[i][grid.jb-1] -
      cbexbcl[i][grid.jb-1]*(
      hzxbcl[i][grid.je-1]+hzybcl[i][grid.je-1]-
      hzxbcb[i][0]-hzybcb[i][0]
                  );
  for(i=0;i<grid.iebc;i++)
  for(j=1;j<grid.je;j++)
  exbcr[i][j]=caexbcr[i][j]*exbcr[i][j] -
              cbexbcr[i][j]*(
        hzxbcr[i][j-1]+hzybcr[i][j-1] -
hzxbcr[i][j]-hzybcr[i][j]
);
  for(i=0;i<grid.iebc;i++)
  exbcr[i][0]=caexbcr[i][0]*exbcr[i][0] -
                cbexbcr[i][0]*(
hzxbcf[i+grid.iebc+grid.ie][grid.jebc-1] +
hzybcf[i+grid.iebc+grid.ie][grid.jebc-1] -
hzxbcr[i][0]-hzybcr[i][0]
      );
for(i=0;i<grid.iebc;i++)
exbcr[i][grid.jb-1]=caexbcr[i][grid.jb-1]*exbcr[i][grid.jb-1] -
    cbexbcr[i][grid.jb-1]*(
    hzxbcr[i][grid.je-1]+hzybcr[i][grid.je-1] -
hzxbcb[i+grid.iebc+grid.ie][0]-
    hzybcb[i+grid.iebc+grid.ie][0]
          );
 for(i=1;i<grid.iefbc;i++)
 for(j=0;j<grid.jebc;j++)
        eybcf[i][j]=caeybcf[i][j]*eybcf[i][j] -
            cbeybcf[i][j]*(
       hzxbcf[i][j] + hzybcf[i][j] -
       hzxbcf[i-1][j] - hzybcf[i-1][j]
   );
  for(i=1;i<grid.iefbc;i++)
  for(j=0;j<grid.jebc;j++)
         eybcb[i][j]=caeybcb[i][j]*eybcb[i][j] -
             cbeybcb[i][j]*(
       hzxbcb[i][j]+hzybcb[i][j] -
   hzxbcb[i-1][j]-hzybcb[i-1][j]
   );
  for(i=1;i<grid.iebc;i++)
  for(j=0;j<grid.je;j++)
  eybcl[i][j]=caeybcl[i][j]*eybcl[i][j] -
              cbeybcl[i][j]*(
            hzxbcl[i][j]+hzybcl[i][j] -
hzxbcl[i-1][j]-hzybcl[i-1][j]
    );
  for(j=0;j<grid.je;j++)
  ey[0][j]=caey[0][j]*ey[0][j] -
           cbey[0][j]*(
hz[0][j]-hzxbcl[grid.iebc-1][j]-hzybcl[grid.iebc-1][j]
  );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
  for(i=1;i<grid.iebc;i++)
  for(j=0;j<grid.je;j++)
  eybcr[i][j] = caeybcr[i][j]*eybcr[i][j]-
                cbeybcr[i][j]*(
      hzxbcr[i][j]+hzybcr[i][j] -
  hzxbcr[i-1][j]-hzybcr[i-1][j]
  );
  for(j=0;j<grid.je;j++)
 ey[grid.ib-1][j]=caey[grid.ib-1][j]*ey[grid.ib-1][j]-
              cbey[grid.ib-1][j]*(
     hzxbcr[0][j]+hzybcr[0][j] - hz[grid.ie-1][j]
);
  for(i=0;i<grid.ie;i++)
  for(j=0;j<grid.je;j++)
  hz[i][j]=dahz[i][j]*hz[i][j]+
           dbhz[i][j]*(
   ex[i][j+1]-ex[i][j]+
   ey[i][j]-ey[i+1][j]
  );
  hz[pSGauss.is-1][pSGauss.js-1] = source[n];
  for(i=0;i<grid.iefbc;i++)
  for(j=0;j<grid.jebc;j++)
  hzxbcf[i][j]=dahzxbcf[i][j]*hzxbcf[i][j] -
  dbhzxbcf[i][j]*(
          eybcf[i+1][j]-eybcf[i][j]
  );
  for(i=0;i<grid.iefbc;i++)
  for(j=0;j<grid.jebc;j++)
hzxbcb[i][j]=dahzxbcb[i][j]*hzxbcb[i][j] -
       dbhzxbcb[i][j]*(
      eybcb[i+1][j]-eybcb[i][j]
  );
 for(i=0;i<grid.iebc-1;i++)
  for(j=0;j<grid.je;j++)
 hzxbcl[i][j]=dahzxbcl[i][j]*hzxbcl[i][j] -
         dbhzxbcl[i][j]*(
               eybcl[i+1][j]-eybcl[i][j]
  );
for(j=0;j<grid.je;j++)
hzxbcl[grid.iebc-1][j]=dahzxbcl[grid.iebc-1][j]*
    hzxbcl[grid.iebc-1][j] -
    dbhzxbcl[grid.iebc-1][j]*(
 ey[0][j]-eybcl[grid.iebc-1][j]
 );
for(i=1;i<grid.iebc;i++)
  for(j=0;j<grid.je;j++)
  hzxbcr[i][j]=dahzxbcr[i][j]*hzxbcr[i][j] -
              dbhzxbcr[i][j]*(
                eybcr[i+1][j]-eybcr[i][j]
 );
for(j=0;j<grid.je;j++)
 hzxbcr[0][j]=dahzxbcr[0][j]*hzxbcr[0][j] -
           dbhzxbcr[0][j]*(
           eybcr[1][j]-ey[grid.ib-1][j]
 );
  for(i=0;i<grid.iefbc;i++)
  for(j=0;j<grid.jebc-1;j++)
hzybcf[i][j]=dahzybcf[i][j]*hzybcf[i][j] -
      dbhzybcf[i][j]*(
            exbcf[i][j] - exbcf[i][j+1]
);
for(i=0;i<grid.iebc;i++)
hzybcf[i][grid.jebc-1]=dahzybcf[i][grid.jebc-1]*
hzybcf[i][grid.jebc-1] -
      dbhzybcf[i][grid.jebc-1]*(
      exbcf[i][grid.jebc-1]-exbcl[i][0]
              );
for(i=grid.iebc;i<grid.iebc+grid.ie;i++)
hzybcf[i][grid.jebc-1]= dahzybcf[i][grid.jebc-1]*
hzybcf[i][grid.jebc-1] -
           dbhzybcf[i][grid.jebc-1]*(
           exbcf[i][grid.jebc-1]-ex[i-grid.iebc][0]
                );
for(i=grid.iebc+grid.ie;i<grid.iefbc;i++)
hzybcf[i][grid.jebc-1]=dahzybcf[i][grid.jebc-1]*
hzybcf[i][grid.jebc-1] -
    dbhzybcf[i][grid.jebc-1]*(
             exbcf[i][grid.jebc-1] -
 exbcr[i-grid.iebc+grid.ie][0]
);
for(i=0;i<grid.iefbc;i++)
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
for(j=1;j<grid.jebc;j++)
 hzybcb[i][j]=dahzybcb[i][j]*hzybcb[i][j] -
       dbhzybcb[i][j]*(
       exbcb[i][j]-exbcb[i][j+1]
 );
for(i=0;i<grid.iebc;i++)
hzybcb[i][0]=dahzybcb[i][0]*hzybcb[i][0] -
       dbhzybcb[i][0]*(
       exbcl[i][grid.jb-1]-exbcb[i][1]
 );
for(i=grid.iebc;i<grid.iebc+grid.ie;i++)
hzybcb[i][0]=dahzybcb[i][0]*hzybcb[i][0] -
            dbhzybcb[i][0]*(
 ex[i-grid.iebc][grid.jb-1]-exbcb[i][1]
  );
for(i=grid.iebc+grid.ie;i<grid.iefbc;i++)
hzybcb[i][0]=dahzybcb[i][0]*hzybcb[i][0] -
        dbhzybcb[i][0]*(
exbcr[i-grid.iebc+grid.ie][grid.jb-1]-exbcb[i][1]
  );
for(i=0;i<grid.iebc;i++)
 for(j=0;j<grid.je;j++)
hzybcl[i][j]=dahzybcl[i][j]*hzybcl[i][j] -
          dbhzybcl[i][j]*(
        exbcl[i][j]-exbcl[i][j+1]
  );
for(i=0;i<grid.iebc;i++)
 for(j=0;j<grid.je;j++)
  hzybcr[i][j]=dahzybcr[i][j]*hzybcr[i][j] -
         dbhzybcr[i][j]*(
         exbcr[i][j]-exbcr[i][j+1]
);
if(n==299){
  FILE *fp1;
  fp1=fopen("ex-cpp-n300.txt","wt");
  for(i=0;i<grid.ie;i++)
  for(j=0;j<grid.jb;j++){
  fprintf(fp1,"\n %i  %i  %3.10f",i,j,ex[i][j]);
  }
  fclose(fp1);
}
}
}


#include "fonte.h"
const unsigned int diametro = 6;


#ifndef MALHA_H
#define MALHA_H
class malha{
  public:
      int nmax;
      int ie;
      int je;
      int ib;
      int jb;
      int iebc;
      int jebc;
      int ibbc;
      int jbbc;
      int iefbc;
      int jefbc;
      int ibfbc;
      int jbfbc;
  public:
malha(int nmax1, int ie1, int je1, int iebc1,int jebc1);
  private:
};
#endif


#include "malha.h"
malha::malha(int nmax1, int ie1, int je1, int iebc1,int jebc1)
:
nmax(nmax1),
ie(ie1),
je(je1),
iebc(iebc1),
jebc(jebc1)
{
   ib = ie1 + 1;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   jb = je1 + 1;
   ibbc = iebc1 + 1;
   jbbc = jebc1 + 1;
   iefbc = ie1 + 2*iebc1;
   jefbc = je1 + 2*jebc1;
   ibfbc = iefbc + 1;
   jbfbc = jefbc + 1;
}
//------------

#ifndef FONTE_H
#define FONTE_H
#include <math.h>
#include "matrix.h"
#include "malha.h"
class fonte{
  public:
     double freq;
     double lambda;
     double omega;
  public:
     unsigned int is;
     unsigned int js;
     double rtau;
 double dx;
     double dt;
     double tau;
     double delay;
  public:
fonte(double dx1, int is1, int js1,
double freq1, double rtau1);
     void pulso_gaussiano(double *vetor);
     void pulsoDuplo(double *vetor);
     void dipolo(double *vetor);
};
#endif

#include <math.h>
#include "fonte.h"
#include "malha.h"
#include "matrix.h"
  static double PI = acos(-1);
  static double cc = 2.99792458e+8;
// construtor
fonte::fonte(double dx1, int is1, int js1,
double freq1, double rtau1)
:
dx(dx1),
is(is1),
js(js1),
freq(freq1),
rtau(rtau1)
{
    lambda= cc/freq;
omega = 2*PI*freq;
    dt    = dx/(2.0*cc);  // 5.00346e-012
    tau   = rtau/dt;
    delay = 3*tau;         // 9593.36
}
void fonte::pulso_gaussiano(double *vetor){
 int n;
   for(n=1;n<7.0*tau;n++)
   vetor[n-1]=sin(omega*(n-delay)*dt)*exp(
pow((n-delay),2)/pow(tau,2)
   )
           );
}
#ifndef MATERIAL_H
#define MATERIAL_H
  static int media=2;
  static double eps[2]={1.0, 1.0};
  static double sig[2]={0.0, 1.0e+7};
  static double mur[2]={1.0, 1.0};
  static double sim[2]={0.0, 0.0};
class material{
  public:
  public:
     material();
};
\begin{verbatim}
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
#ifndef MATRIX_H
#define MATRIX_H
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

class matrix{
  public:
void impressao(double **matriz, int n,int m);
void impressao(double *vetor, int n);
void salvardados(const char *nomarq,
double *vetor, int n);
void salvardados(const char *nomarq,
double **matriz, int n, int m);
  private:
};
//-- Inicialização de Vetores ------------
      double *zeros(int n);
      double **zeros(int n, int m);
#endif
```

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "matrix.h"
double *zeros(int n){
double *v;
v = (double *)malloc((size_t)((n+2)*sizeof(double)));
    if(!v) {
      fprintf(stderr,"Runtime error \n");
  exit(1);
}
v+=1;
for(int i=0;i<n;i++)
  v[i]=0.0;
return v;
}
double **zeros(int n, int m){
  double **matriz;
  int i,j;
  matriz = (double **)malloc((size_t)
  ((n+1)*sizeof(double *)));
  if(!matriz) {
      fprintf(stderr,"Runtime error \n");
  exit(1);
  }
  matriz += 1;
  matriz[0] = (double *)malloc((size_t)
  ((n*m+1)*sizeof(double)));
  if(!matriz) {
      fprintf(stderr,"Runtime error \n");
  exit(1);
  }
  matriz[0] +=1;

  for(i=1;i<=n;i++)
  matriz[i] = matriz[i-1] + m;
  for(i=0;i<n;i++)
    for(j=0;j<m;j++)
   matriz[i][j]=0.0;
  return matriz;
}
void matrix::impressao(double **matriz, int n,int m){
  for(int i=0;i<n;i++){
  for(int j=0;j<m;j++){
cout<<"\tM["<<i<<","<<j<<"] = "<<matriz[i][j]<<"   ";

  }
  }
  cout<<"\n\n\n"<<endl;
}
void matrix::impressao(double *vetor, int n){
  char p;
for(int k=0;k<n;k++){
  cout<< "\n A["<<k<<"] = "<<vetor[k];
  if((k+1)%30==0) cin>>p;
}
  cout<<"\n\n\n"<<endl;
}
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
void matrix::salvardados(const char *nomarq,
double *vetor, int n){
FILE *fp;
fp = fopen(nomarq,"wt");
for(int k=0;k<n;k++){
          fprintf(fp,"\n %7.17f",vetor[k]);
    }
fclose(fp);
}
void matrix::salvardados(const char *nomarq,
double **matriz, int n, int m){
FILE *fp;
    int k,l;
fp = fopen(nomarq,"wt");
for(k=0;k<n;k++){
for(l=0;l<m;l++)
          fprintf(fp,"\n %3.17f",matriz[k][l]);
    }
fclose(fp);
}
```

```
#include <iostream>
#include <math.h>
#include "constantes.h"
#include "material.h"
#include "malha.h"
#include "fonte.h"
#include "matrix.h"

int main(){
malha *grid = new malha(10,41,17,16,0.0020);
malha(10,41,17,16,0.0020);
grid(thickness_upml, ie, je, ke, dx)
double dt   = grid->dx*sqrt(epsr*mur)/(2.0*cc);
int nmax    = 100;
int is,js,ks;
is=(int)(grid->ih_tot/2)-1;
    js=(int)(grid->jh_tot/2)-1;
    ks=(int)(grid->kh_tot/2)-1;
double rtau = 50.0e-12;
fonte *source = new fonte(is,js,ks,rtau,dt);
       double ***ex;    double ***hx;
   double ***ey;    double ***hy;
   double ***ez;    double ***hz;
   double ***dx;    double ***bx;
   double ***dy;    double ***by;
   double ***dz;    double ***bz;

// matrix m;
// m.impressao(ex,grid->ie_tot,grid->jh_tot,grid->kh_tot);


ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
dx=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
bx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
dy=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
by=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
dz=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
bz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
double ***C1ex, ***C2ex, ***C3ex, ***C4ex, ***C5ex, ***C6ex;
double ***C1ey, ***C2ey, ***C3ey, ***C4ey, ***C5ey, ***C6ey;
double ***C1ez, ***C2ez, ***C3ez, ***C4ez, ***C5ez, ***C6ez;
double ***D1hx, ***D2hx, ***D3hx, ***D4hx, ***D5hx, ***D6hx;
double ***D1hy, ***D2hy, ***D3hy, ***D4hy, ***D5hy, ***D6hy;
double ***D1hz, ***D2hz, ***D3hz, ***D4hz, ***D5hz, ***D6hz;
C1ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C1ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
C1ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
C2ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C2ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
C2ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
C3ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C3ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
C3ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
C4ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C4ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
C4ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
C5ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C5ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
C5ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
C6ex=zeros(grid->ie_tot,grid->jh_tot,grid->kh_tot);
C6ey=zeros(grid->ih_tot,grid->je_tot,grid->kh_tot);
C6ez=zeros(grid->ih_tot,grid->jh_tot,grid->ke_tot);
D1hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D1hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
D1hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
D2hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D2hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);

D2hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
D3hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D3hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
D3hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
D4hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D4hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
D4hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
D5hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D5hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
D5hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);
D6hx=zeros(grid->ih_tot,grid->je_tot,grid->ke_tot);
D6hy=zeros(grid->ie_tot,grid->jh_tot,grid->ke_tot);
D6hz=zeros(grid->ie_tot,grid->je_tot,grid->kh_tot);

       double C1 = 1.0;
       double D1 = 1.0;
   double C2 = source->dt;
double D2 = source->dt;
   double C3 = 1.0;
double D3 = 1.0;
   double C4 = 1.0/2.0/epsr/epsr/epsz/epsz;
double D4 = 1.0/2.0/epsr/epsz/mur/muz;
   double C5 = 2.0*epsr*epsz;
double D5 = 2.0*epsr*epsz;
   double C6 = C5;
double D6 = D5;
       int i,j,k;
       for(i=0;i<grid->ie_tot;i++)
for(j=grid->jh_bc - 1;j<grid->jh_tot - grid->upml;j++)
   for(k=0;k<grid->kh_tot;k++){
                   C1ex[i][j][k] = C1;
   C2ex[i][j][k] = C2;
   }
       for(i=0;i<grid->ie_tot;i++)
   for(j=0;j<grid->jh_tot;j++)
for(k=grid->kh_bc - 1;k<grid->kh_tot - grid->upml;k++){
                   C3ex[i][j][k] = C3;
   C4ex[i][j][k] = C4;
   }
for(i=grid->ih_bc - 1;i<grid->ie_tot - grid->upml;i++)
   for(j=0;j<grid->jh_tot;j++)
   for(k=0;k<grid->kh_tot;k++){
                   C5ex[i][j][k] = C5;
   C6ex[i][j][k] = C6;
   }
for(i=0;i<grid->ih_tot;i++)
   for(j=0;j<grid->je_tot;j++)
for(k=grid->kh_bc - 1;k<grid->kh_tot - grid->upml;k++){
                   C1ey[i][j][k] = C1;
   C2ey[i][j][k] = C2;
   }
for(i=grid->ih_bc - 1;i<grid->ih_tot - grid->upml;i++)
   for(j=0;j<grid->je_tot;j++)
   for(k=0;k<grid->kh_tot;k++){
                   C3ey[i][j][k] = C3;
   C4ey[i][j][k] = C4;
   }
       for(i=0;i<grid->ih_tot;i++)
for(j=grid->jh_bc - 1;j<grid->je_tot - grid->upml;j++)
   for(k=0;k<grid->kh_tot;k++){
                   C5ey[i][j][k] = C5;
   C6ey[i][j][k] = C6;
   }
for(i=grid->ih_bc - 1;i<grid->ih_tot - grid->upml;i++)
   for(j=0;j<grid->jh_tot;j++)
   for(k=0;k<grid->ke_tot;k++){
                   C1ez[i][j][k] = C1;
   C2ez[i][j][k] = C2;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   }
       for(i=0;i<grid->ih_tot;i++)
for(j=grid->jh_bc - 1;j<grid->jh_tot - grid->upml;j++)
   for(k=0;k<grid->ke_tot;k++){
                  C3ez[i][j][k] = C3;
   C4ez[i][j][k] = C4;
   }
       for(i=0;i<grid->ih_tot;i++)
   for(j=0;j<grid->jh_tot;j++)
for(k=grid->kh_bc - 1;k<grid->ke_tot  - grid->upml;k++){
                  C5ez[i][j][k] = C5;
   C6ez[i][j][k] = C6;
   }
for(i=0;i<grid->ih_tot;i++)
for(j=grid->jh_bc - 1;j<grid->je_tot - grid->upml;j++)
   for(k=0;k<grid->ke_tot;k++){
                  D1hx[i][j][k] = D1;
   D2hx[i][j][k] = D2;
   }
for(i=0;i<grid->ih_tot;i++)
   for(j=0;j<grid->je_tot;j++)
for(k=grid->kh_bc - 1;k<grid->ke_tot - grid->upml;k++){
                  D3hx[i][j][k] = D3;
   D4hx[i][j][k] = D4;
   }
for(i=grid->ih_bc - 1;i<grid->ih_tot - grid->upml;i++)
   for(j=0;j<grid->je_tot;j++)
for(k=0;k<grid->ke_tot;k++){
                  D5hx[i][j][k] = D5;
   D6hx[i][j][k] = D6;
   }
      for(i=0;i<grid->ie_tot;i++)
   for(j=0;j<grid->jh_tot;j++)
for(k=grid->kh_bc - 1;k<grid->ke_tot - grid->upml;k++){
                  D1hy[i][j][k] = D1;
   D2hy[i][j][k] = D2;
   }
for(i=grid->ih_bc - 1;i<grid->ie_tot - grid->upml;i++)
   for(j=0;j<grid->jh_tot;j++)
   for(k=0;k<grid->ke_tot;k++){
                  D3hy[i][j][k] = D3;
   D4hy[i][j][k] = D4;
   }
       for(i=0;i<grid->ie_tot;i++)
for(j=grid->jh_bc - 1;j<grid->jh_tot - grid->upml;j++)
   for(k=0;k<grid->ke_tot;k++){
                  D5hy[i][j][k] = D5;
   D6hy[i][j][k] = D6;
   }
for(i=grid->ih_bc - 1;i<grid->ie_tot - grid->upml;i++)
   for(j=0;j<grid->je_tot;j++)
   for(k=0;k<grid->kh_tot;k++){
                  D1hz[i][j][k] = D1;
   D2hz[i][j][k] = D2;
   }
       for(i=0;i<grid->ie_tot;i++)
for(j=grid->jh_bc - 1;j<grid->je_tot - grid->upml;j++)
   for(k=0;k<grid->kh_tot;k++){
                  D3hz[i][j][k] = D3;
   D4hz[i][j][k] = D4;
   }
       for(i=0;i<grid->ie_tot;i++)
   for(j=0;j<grid->je_tot;j++)
for(k=grid->kh_bc - 1;k<grid->kh_tot  - grid->upml;k++){
                  D5hz[i][j][k] = D5;
   D6hz[i][j][k] = D6;
   }
double rmax     = exp(-16);
int orderbc     = 4;
double delbc    = grid->upml*grid->dx;
double sigmam   = -log(rmax)*(orderbc+1.0)/(2.0*eta*delbc);
double sigfactor= sigmam/(grid->dx*(pow(delbc,orderbc))*(orderbc+1));
int kmax = 1;
double kfactor  = (kmax-1.0)/grid->dx/(orderbc+1)/(pow(delbc,orderbc));

double x1,x2,sigma,ki,facm,facp;
        for(i=0;i<grid->upml;i++){
           x1 = (grid->upml-(i+1)+1)*grid->dx;
   x2 = (grid->upml-(i+1))*grid->dx;
sigma= sigfactor*( pow(x1,(orderbc+1)) - pow(x2,(orderbc+1)) );
ki   = 1+kfactor*( pow(x1,(orderbc+1)) - pow(x2,(orderbc+1)) );
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   facm = 2*epsr*epsz*ki - sigma*source->dt;
   facp = 2*epsr*epsz*ki + sigma*source->dt;

           for(j=0;j<grid->jh_tot;j++){
   for(k=0;k<grid->kh_tot;k++){
     C5ex[i][j][k] = facp;
         C5ex[grid->ie_tot-(i+2)+1][j][k] = facp;
     C6ex[i][j][k] = facm;
         C6ex[grid->ie_tot-(i+2)+1][j][k] = facm;
   }
   }
   for(j=0;j<grid->je_tot;j++){
   for(k=0;k<grid->kh_tot;k++){
D1hz[i][j][k]= facm/facp;
D1hz[grid->ie_tot-(i+2)+1][j][k] = facm/facp;
D2hz[i][j][k]= 2.0*epsr*epsz*source->dt/facp;
D2hz[grid->ie_tot-(i+2)+1][j][k] = 2.0*epsr*epsz*source->dt/facp;
   }
   }
           for(j=0;j<grid->jh_tot;j++){
   for(k=0;k<grid->ke_tot;k++){
 D3hy[i][j][k]= facm/facp;
D3hy[grid->ie_tot-(i+2)+1][j][k] = facm/facp;
D4hy[i][j][k]=1.0/facp/mur/muz;
D4hy[grid->ie_tot-(i+2)+1][j][k] = 1.0/facp/mur/muz;
   }
   }
           x1 = (grid->upml-(i+1)+1.5)*grid->dx;
   x2 = (grid->upml-(i+1)+0.5)*grid->dx;
sigma = sigfactor*( pow(x1,(orderbc+1)) - pow(x2,(orderbc+1)) );
ki    = 1+kfactor*( pow(x1,(orderbc+1)) - pow(x2,(orderbc+1)) );
facm = 2*epsr*epsz*ki - sigma*source->dt;
facp = 2*epsr*epsz*ki + sigma*source->dt;
           for(j=0;j<grid->jh_tot;j++){
   for(k=0;k<grid->ke_tot;k++){
C1ez[i][j][k] = facm/facp;
C1ez[grid->ih_tot-(i+2)+1][j][k] = facm/facp;
C2ez[i][j][k] = 2.0*epsr*epsz*source->dt/facp;
C2ez[grid->ih_tot-(i+2)+1][j][k] = 2.0*epsr*epsz*source->dt/facp;
   }
   }
           for(j=0;j<grid->je_tot;j++){
   for(k=0;k<grid->kh_tot;k++){
     C3ey[i][j][k]= facm/facp;
C3ey[grid->ih_tot-(i+2)+1][j][k] = facm/facp;
C4ey[i][j][k]= 1.0/facp/epsr/epsz;
C4ey[grid->ih_tot-(i+2)+1][j][k] = 1.0/facp/epsr/epsz;
     }
   }
           for(j=0;j<grid->je_tot;j++){
   for(k=0;k<grid->ke_tot;k++){
 D5hx[i][j][k]= facp;
D5hx[grid->ih_tot-(i+2)+1][j][k] = facp;
D6hx[i][j][k]=facm;
D6hx[grid->ih_tot-(i+2)+1][j][k] = facm;
   }
   }
}
        for(j=0;j<grid->jh_tot;j++){
   for(k=0;k<grid->ke_tot;k++){
    C1ez[0][j][k] = -1.0;
                    C1ez[grid->ih_tot-1][j][k] = -1.0;
                    C2ez[0][j][k] = 0.0;
                    C2ez[grid->ih_tot-1][j][k] = 0.0;
   }
}
for(j=0;j<grid->je_tot;j++){
   for(k=0;k<grid->kh_tot;k++){
C3ey[0][j][k] = -1.0;
                    C3ey[grid->ih_tot-1][j][k] = -1.0;
                    C4ey[0][j][k] = 0.0;
                    C4ey[grid->ih_tot-1][j][k] = 0.0;
   }
}
delbc    = grid->upml*grid->dx;
sigmam   = -log(rmax)*epsr*epsz*cc*(orderbc+1.0)/(2.0*delbc);
sigfactor= sigmam/(grid->dx*(pow(delbc,orderbc))*(orderbc+1));
kmax = 1;
kfactor  = (kmax-1.0)/grid->dx/(orderbc+1)/(pow(delbc,orderbc));
double y1,y2;
for(j=0;j<grid->upml;j++){
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
y1 = (grid->upml-(j+1)+1)*grid->dx;
            y2 = (grid->upml-(j+1))*grid->dx;
sigma= sigfactor*( pow(y1,(orderbc+1)) - pow(y2,(orderbc+1)) );
ki   = 1+kfactor*( pow(y1,(orderbc+1)) - pow(y2,(orderbc+1)) );
    facm = 2*epsr*epsz*ki - sigma*source->dt;
    facp = 2*epsr*epsz*ki + sigma*source->dt;
            for(i=0;i<grid->ih_tot;i++){
   for(k=0;k<grid->kh_tot;k++){
     C5ey[i][j][k] = facp;
         C5ey[i][grid->je_tot-(j+2)+1][k] = facp;
     C6ey[i][j][k] = facm;
         C6ey[i][grid->je_tot-(j+2)+1][k] = facm;
   }
   }
   for(i=0;i<grid->ih_tot;i++){
   for(k=0;k<grid->ke_tot;k++){
D1hx[i][j][k]= facm/facp;
D1hx[i][grid->je_tot-(j+2)+1][k] = facm/facp;
D2hx[i][j][k]= 2.0*epsr*epsz*source->dt/facp;
D2hx[i][grid->je_tot-(j+2)+1][k] = 2.0*epsr*epsz*source->dt/facp;
   }
   }
           for(i=0;i<grid->ie_tot;i++){
   for(k=0;k<grid->kh_tot;k++){
D3hz[i][j][k]= facm/facp;
D3hz[i][grid->je_tot-(j+2)+1][k] = facm/facp;
D4hz[i][j][k]=1.0/facp/mur/muz;
D4hz[i][grid->je_tot-(j+2)+1][k] = 1.0/facp/mur/muz;
   }
   }
y1 = (grid->upml-(j+1)+1.5)*grid->dx;
y2 = (grid->upml-(j+1)+0.5)*grid->dx;
sigma = sigfactor*( pow(y1,(orderbc+1)) - pow(y2,(orderbc+1)) );
ki    = 1+kfactor*( pow(y1,(orderbc+1)) - pow(y2,(orderbc+1)) );
facm = 2*epsr*epsz*ki - sigma*source->dt;
facp = 2*epsr*epsz*ki + sigma*source->dt;
   for(i=0;i<grid->ie_tot;i++){
   for(k=0;k<grid->kh_tot;k++){
    C1ex[i][j][k] = facm/facp;
   C1ex[i][grid->jh_tot-(j+2)+1][k] = facm/facp;
   C2ex[i][j][k] = 2.0*epsr*epsz*source->dt/facp;
C2ex[i][grid->jh_tot-(j+2)+1][k] = 2.0*epsr*epsz*source->dt/facp;
   }
   }
           for(i=0;i<grid->ih_tot;i++){
   for(k=0;k<grid->ke_tot;k++){
C3ez[i][j][k]= facm/facp;
C3ez[i][grid->jh_tot-(j+2)+1][k] = facm/facp;
C4ez[i][j][k]= 1.0/facp/epsr/epsz;
C4ez[i][grid->jh_tot-(j+2)+1][k] = 1.0/facp/epsr/epsz;
     }
   }
           for(i=0;i<grid->ie_tot;i++){
   for(k=0;k<grid->ke_tot;k++){
D5hy[i][j][k]= facp;
D5hy[i][grid->jh_tot-(j+2)+1][k] = facp;
D6hy[i][j][k]=facm;
D6hy[i][grid->jh_tot-(j+2)+1][k] = facm;
   }
   }
}
        for(i=0;i<grid->ie_tot;i++){
   for(k=0;k<grid->kh_tot;k++){
C1ex[i][0][k] = -1.0;
C1ex[i][grid->jh_tot-1][k] = -1.0;
C2ex[i][0][k] =  0.0;
C2ex[i][grid->jh_tot-1][k] =  0.0;
   }
}
for(i=0;i<grid->ih_tot;i++){
   for(k=0;k<grid->ke_tot;k++){
C3ez[i][0][k] = -1.0;
C3ez[i][grid->jh_tot-1][k] = -1.0;
C4ez[i][0][k] =  0.0;
C4ez[i][grid->jh_tot-1][k] =  0.0;
   }
}
delbc    = grid->upml*grid->dx;
sigmam   = -log(rmax)*epsr*epsz*cc*(orderbc+1.0)/
            (2.0*delbc);
sigfactor= sigmam/(grid->dx*(pow(delbc,orderbc))*
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
            (orderbc+1));
kmax = 1;
kfactor  = (kmax-1)/grid->dx/(orderbc+1)
             /(pow(delbc,orderbc));

double z1,z2;

for(k=0;k<grid->upml;k++){
z1 = (grid->upml-(k+1)+1)*grid->dx;
z2 = (grid->upml-(k+1))*grid->dx;
sigma= sigfactor*( pow(z1,(orderbc+1))
         - pow(z2,(orderbc+1)) );
ki   = 1+kfactor*( pow(z1,(orderbc+1))
         - pow(z2,(orderbc+1)) );
facm = 2*epsr*epsz*ki - sigma*source->dt;
facp = 2*epsr*epsz*ki + sigma*source->dt;
for(i=0;i<grid->ih_tot;i++){
for(j=0;j<grid->jh_tot;j++){
   C5ez[i][j][k] = facp;
   C5ez[i][j][grid->ke_tot-(k+2)+1] = facp;
   C6ez[i][j][k] = facm;
   C6ez[i][j][grid->ke_tot-(k+2)+1] = facm;
}
}
for(i=0;i<grid->ie_tot;i++){
for(j=0;j<grid->jh_tot;j++){
D1hy[i][j][k]= facm/facp;
D1hy[i][j][grid->ke_tot-(k+2)+1] = facm/facp;
D2hy[i][j][k]= 2.0*epsr*epsz*source->dt/facp;
D2hy[i][j][grid->ke_tot-(k+2)+1] =
            2.0*epsr*epsz*source->dt/facp;
}
}

for(i=0;i<grid->ih_tot;i++){
for(j=0;j<grid->je_tot;j++){
D3hx[i][j][k]= facm/facp;
D3hx[i][j][grid->ke_tot-(k+2)+1] = facm/facp;
D4hx[i][j][k]=1.0/facp/mur/muz;
D4hx[i][j][grid->ke_tot-(k+2)+1] =
                 1.0/facp/mur/muz;
}
}

z1 = (grid->upml-(k+1)+1.5)*grid->dx;
z2 = (grid->upml-(k+1)+0.5)*grid->dx;
sigma = sigfactor*( pow(z1,(orderbc+1)) - pow(z2,(orderbc+1)) );
ki    = 1+kfactor*( pow(z1,(orderbc+1)) - pow(z2,(orderbc+1)) );
facm = 2*epsr*epsz*ki - sigma*source->dt;
facp = 2*epsr*epsz*ki + sigma*source->dt;
for(i=0;i<grid->ih_tot;i++){
   for(j=0;j<grid->je_tot;j++){
C1ey[i][j][k] = facm/facp;
C1ey[i][j][grid->kh_tot-(k+2)+1] = facm/facp;
C2ey[i][j][k] = 2.0*epsr*epsz*source->dt/facp;
C2ey[i][j][grid->kh_tot-(k+2)+1] = 2.0*epsr*epsz*source->dt/facp;
   }
}
for(i=0;i<grid->ie_tot;i++){
for(j=0;j<grid->jh_tot;j++){
C3ex[i][j][k]= facm/facp;
C3ex[i][j][grid->kh_tot-(k+2)+1] = facm/facp;
C4ex[i][j][k]= 1.0/facp/epsr/epsz;
C4ex[i][j][grid->kh_tot-(k+2)+1] = 1.0/facp/epsr/epsz;
     }
}
for(i=0;i<grid->ie_tot;i++){
for(j=0;j<grid->je_tot;j++){
D5hz[i][j][k]= facp;
D5hz[i][j][grid->kh_tot-(k+2)+1] = facp;
D6hz[i][j][k]=facm;
D6hz[i][j][grid->kh_tot-(k+2)+1] = facm;
}
}
}
        for(i=0;i<grid->ih_tot;i++){
   for(j=0;j<grid->je_tot;j++){
C1ey[i][j][0] = -1.0;
C1ey[i][j][grid->kh_tot-1] = -1.0;
C2ey[i][j][0] =  0.0;
C2ey[i][j][grid->kh_tot-1] =  0.0;
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   }
}
for(i=0;i<grid->ie_tot;i++){
   for(j=0;j<grid->jh_tot;j++){
C3ex[i][j][0] = -1.0;
C3ex[i][j][grid->kh_tot-1] = -1.0;
C4ex[i][j][0] =  0.0;
C4ex[i][j][grid->kh_tot-1] =  0.0;
   }
}
double ***bstore;
bstore=zeros(grid->ih_tot,
grid->jh_tot,grid->kh_tot);
double ***dstore;
dstore=zeros(grid->ih_tot,
grid->jh_tot,grid->kh_tot);
 int c;
 for(int n=1;n<=nmax;n++){
//-----------------------------------------
for(i=0;i<grid->ih_tot;i++)
for(j=0;j<grid->je_tot;j++)
 for(k=0;k<grid->ke_tot;k++)
  bstore[i][j][k]=bx[i][j][k];
for(i=1;i<grid->ie_tot;i++)
 for(j=0;j<grid->je_tot;j++)
 for(k=0;k<grid->ke_tot;k++){
bx[i][j][k] = D1hx[i][j][k]*bx[i][j][k] -
      D2hx[i][j][k]*(
(ez[i][j+1][k] -  ez[i][j][k]) -
(ey[i][j][k+1] -  ey[i][j][k])
  )/grid->dx;
}
for(i=1;i<grid->ie_tot;i++)
for(j=0;j<grid->je_tot;j++)
  for(k=0;k<grid->ke_tot;k++){
 hx[i][j][k] = D3hx[i][j][k]*hx[i][j][k] +
   D4hx[i][j][k]*(
  D5hx[i][j][k]*bx[i][j][k] -
  D6hx[i][j][k]*bstore[i][j][k]
  );
}
//----------------------------------------
for(i=0;i<grid->ie_tot;i++)
for(j=0;j<grid->jh_tot;j++)
for(k=0;k<grid->ke_tot;k++)
   bstore[i][j][k]=by[i][j][k];
 for(i=0;i<grid->ie_tot;i++)
  for(j=1;j<grid->je_tot;j++)
 for(k=0;k<grid->ke_tot;k++){
   by[i][j][k] = D1hy[i][j][k]*by[i][j][k] -
       D2hy[i][j][k]*(
   (ex[i][j][k+1] -  ex[i][j][k]) -
 (ez[i+1][j][k] -  ez[i][j][k])
  )/grid->dx;
}
for(i=0;i<grid->ie_tot;i++)
  for(j=1;j<grid->je_tot;j++)
 for(k=0;k<grid->ke_tot;k++){
hy[i][j][k] = D3hy[i][j][k]*hy[i][j][k] +
D4hy[i][j][k]*(
 D5hy[i][j][k]*by[i][j][k] -
 D6hy[i][j][k]*bstore[i][j][k]
    );
}
 for(i=0;i<grid->ie_tot;i++)
 for(j=0;j<grid->je_tot;j++)
 for(k=0;k<grid->kh_tot;k++)
     bstore[i][j][k]=bz[i][j][k];
 for(i=0;i<grid->ie_tot;i++)
    for(j=0;j<grid->je_tot;j++)
 for(k=1;k<grid->ke_tot;k++){
  bz[i][j][k] = D1hz[i][j][k]*bz[i][j][k] -
      D2hz[i][j][k]*(
 (ey[i+1][j][k] - ey[i][j][k]) -
 (ex[i][j+1][k] - ex[i][j][k])
 )/grid->dx;
  }
 for(i=0;i<grid->ie_tot;i++)
  for(j=0;j<grid->je_tot;j++)
for(k=1;k<grid->ke_tot;k++){
  hz[i][j][k] = D3hz[i][j][k]*hz[i][j][k] +
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
   D4hz[i][j][k]*(
D5hz[i][j][k]*bz[i][j][k] -
D6hz[i][j][k]*bstore[i][j][k]
          );
 }
//-----------------------------------------------
  for(i=0;i<grid->ie_tot;i++)
    for(j=0;j<grid->jh_tot;j++)
  for(k=0;k<grid->kh_tot;k++)
        dstore[i][j][k]=dx[i][j][k];
    for(i=0;i<grid->ie_tot;i++)
     for(j=1;j<grid->je_tot;j++)
 for(k=1;k<grid->ke_tot;k++){
       dx[i][j][k] = C1ex[i][j][k]*dx[i][j][k] +
             C2ex[i][j][k]*(
             (hz[i][j][k] -  hz[i][j-1][k]) -
               (hy[i][j][k] -  hy[i][j][k-1])
              )/grid->dx;
  }
          for(i=0;i<grid->ie_tot;i++)
            for(j=1;j<grid->je_tot;j++)
  for(k=1;k<grid->ke_tot;k++){
          ex[i][j][k] = C3ex[i][j][k]*ex[i][j][k] +
             C4ex[i][j][k]*(
                       C5ex[i][j][k]*dx[i][j][k] -
                       C6ex[i][j][k]*dstore[i][j][k]
               );
  }
  for(i=0;i<grid->ih_tot;i++)
          for(j=0;j<grid->je_tot;j++)
  for(k=0;k<grid->kh_tot;k++)
             dstore[i][j][k]=dy[i][j][k];


          for(i=1;i<grid->ie_tot;i++)
            for(j=0;j<grid->je_tot;j++)
  for(k=1;k<grid->ke_tot;k++){
       dy[i][j][k] = C1ey[i][j][k]*dy[i][j][k] +
             C2ey[i][j][k]*(
                     (hx[i][j][k] -  hx[i][j][k-1]) -
                     (hz[i][j][k] -  hz[i-1][j][k])
               )/grid->dx;
  }
          for(i=1;i<grid->ie_tot;i++)
            for(j=0;j<grid->je_tot;j++)
  for(k=1;k<grid->ke_tot;k++){
       ey[i][j][k] = C3ey[i][j][k]*ey[i][j][k] +
             C4ey[i][j][k]*(
                 C5ey[i][j][k]*dy[i][j][k] -
                 C6ey[i][j][k]*dstore[i][j][k]
               );
  }
  for(i=0;i<grid->ih_tot;i++)
          for(j=0;j<grid->jh_tot;j++)
  for(k=0;k<grid->ke_tot;k++)
             dstore[i][j][k]=dz[i][j][k];
          for(i=1;i<grid->ie_tot;i++)
            for(j=1;j<grid->je_tot;j++)
  for(k=0;k<grid->ke_tot;k++){
     dz[i][j][k] = C1ez[i][j][k]*dz[i][j][k] +
             C2ez[i][j][k]*(
                     (hy[i][j][k] -  hy[i-1][j][k]) -
                     (hx[i][j][k] -  hx[i][j-1][k])
               )/grid->dx;
  }

dz[is][js][ks]=dz[is][js][ks]+source->
                 J0*(n-source->ndelay)*exp(-(
pow((n-source->ndelay),2)/
pow(source->tau,2))
);
dz[is][js][ks+1]=dz[is][js][ks+1]+
     source->J0*(n-source->ndelay)*exp(-(
pow((n-source->ndelay),2)/
pow(source->tau,2))
);
  for(i=1;i<grid->ie_tot;i++)
            for(j=1;j<grid->je_tot;j++)
  for(k=0;k<grid->ke_tot;k++){
          ez[i][j][k] = C3ez[i][j][k]*ez[i][j][k] +
             C4ez[i][j][k]*(
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
                          C5ez[i][j][k]*dz[i][j][k] -
                          C6ez[i][j][k]*dstore[i][j][k]
               );
  }
if(n==nmax){
FILE *fp1,*fp2;
fp1=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           bx_cpp_nmax.txt","wt");

fp2=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           hx_cpp_nmax.txt","wt");

fprintf(fp1,"\n  n    D1hx      bstore     D2hx       ez[j+1]
                ez       ey[k+1]    ey      bx");
fprintf(fp2,"\n  n    D3hx      hx         D4hx
                D5hx       bx       D6hx       bstore");
c=1;
for(i=1;i<grid->ie_tot;i++){
      for(j=0;j<grid->je_tot;j++){
for(k=0;k<grid->ke_tot;k++){

if(bstore[i][j][k] || bx[i][j][k] || ez[i][j+1][k] ||
ez[i][j][k] || ey[i][j][k+1] || ey[i][j][k]){
    printf(fp1,"\n%i  %g   %g   %g   %g   %g   %g   %g
    %g",c,D1hx[i][j][k],bstore[i][j][k],D2hx[i][j][k],
ez[i][j+1][k],ez[i][j][k],ey[i][j][k+1],ey[i][j][k],bx[i][j][k]);
}
if(hx[i][j][k] || bx[i][j][k] || bstore[i][j][k]){
fprintf(fp2,"\n%i  %g   %g   %g   %g   %g
%g",c,D3hx[i][j][k],hx[i][j][k],D4hx[i][j][k],
      D5hx[i][j][k],bx[i][j][k],
D6hx[i][j][k],bstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp1);
fclose(fp2);
//=====================================================
FILE *fp3,*fp4;
fp3=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           by_cpp_nmax.txt","wt");
fp4=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           hy_cpp_nmax.txt","wt");

fprintf(fp3,"\n  n    D1hy      bstore    D2hy      ex[k+1]
ex      ez[i+1]      ez       by[i][j][k]");
fprintf(fp4,"\n  n    D3hy      hy        D4hy      D5hy
by      D6hy         bstore");

c=1;
for(i=0;i<grid->ie_tot;i++){
      for(j=1;j<grid->je_tot;j++){
for(k=0;k<grid->ke_tot;k++){

if(bstore[i][j][k] || by[i][j][k]
            || ex[i][j][k+1] || ex[i][j][k] ||
ez[i+1][j][k] || ez[i][j][k]){
 fprintf(fp3,"\n%i  %g   %g   %g   %g   %g   %g   %g
 %g",c,D1hy[i][j][k],bstore[i][j][k],D2hy[i][j][k],
 ex[i][j][k+1],ex[i][j][k],
 ez[i+1][j][k],ez[i][j][k],by[i][j][k]);
}
if(hy[i][j][k] || by[i][j][k] || bstore[i][j][k]){
fprintf(fp4,"\n%i  %g   %g   %g   %g   %g   %g
%g",c,D3hy[i][j][k],hy[i][j][k],D4hy[i][j][k],D5hy[i][j][k],
by[i][j][k],D6hy[i][j][k],bstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp3);
fclose(fp4);
//=======================================================
FILE *fp5,*fp6;
fp5=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           bz_cpp_nmax.txt","wt");
fp6=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           hz_cpp_nmax.txt","wt");
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
fprintf(fp5,"\n   n   i   j   k    D1hz
            bstore     D2hz     ey[i+1]

ey      ex[j+1]      ex      bz");

fprintf(fp6,"\n   n   i   j   k
             D3hz    hz         D4hz     D5hz

bz      D6hz         bstore");

c=1;
for(i=0;i<grid->ie_tot;i++){
      for(j=0;j<grid->je_tot;j++){
for(k=1;k<grid->ke_tot;k++){

if(bstore[i][j][k] || bz[i][j][k] || ey[i+1][j][k] ||
ey[i][j][k] || ex[i][j+1][k]
|| ex[i][j][k]){
fprintf(fp5,"\n%i  %i   %i   %i
            %g   %g   %g   %g   %g   %g   %g
%g",c,i,j,k,D1hz[i][j][k],bstore[i][j][k],D2hz[i][j][k],
ey[i+1][j][k],ey[i][j][k],
ex[i][j+1][k],ex[i][j][k],bz[i][j][k]);
}
if(hz[i][j][k] || bz[i][j][k] || bstore[i][j][k]){
fprintf(fp6,"\n%i  %i   %i   %i
                 %g   %g   %g   %g   %g   %g
%g",c,i,j,k,D3hz[i][j][k],hz[i][j][k],D4hz[i][j][k],
D5hz[i][j][k],bz[i][j][k],
D6hz[i][j][k],bstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp5);
fclose(fp6);
//============================================================
FILE *fp7,*fp8;
fp7=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
            dx_cpp_nmax.txt","wt");
fp8=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
            ex_cpp_nmax.txt","wt");
fprintf(fp7,"\n  n   C1ex    dstore
             C2ex      hz        hz[j-1]     hy
hy[k-1]      dx");
fprintf(fp8,"\n  n   C3ex    ex          C4ex
                C5ex      dx          C6ex
dstore");
c=1;
for(i=0;i<grid->ie_tot;i++){
      for(j=1;j<grid->je_tot;j++){
for(k=1;k<grid->ke_tot;k++){
if(dstore[i][j][k] || dx[i][j][k] || hz[i][j-1][k]
                  || hz[i][j][k] || hy[i][j][k]
|| hy[i][j][k-1]){
fprintf(fp7,"\n%i  %g   %g   %g   %g   %g   %g   %g
%g",c,C1ex[i][j][k],dstore[i][j][k],C2ex[i][j][k],
           hz[i][j][k],hz[i][j-1][k],
hy[i][j][k],hy[i][j][k-1],dx[i][j][k]);
}
if(ex[i][j][k] || dx[i][j][k] || dstore[i][j][k]){
fprintf(fp8,"\n%i  %g   %g   %g   %g   %g   %g
%g",c,C3ex[i][j][k],ex[i][j][k],
          C4ex[i][j][k],C5ex[i][j][k],
dx[i][j][k],C6ex[i][j][k],

dstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp7);
fclose(fp8);
//================================================
FILE *fp9,*fp10;
fp9=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
          dy_cpp_nmax.txt","wt");
fp10=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
          ey_cpp_nmax.txt","wt");
```

PUC-Rio - Certificação Digital Nº 0310507/CB

```
fprintf(fp9, "\n  n    C1ey     dstore     C2ey     hx
  hx[k-1]    hz
hz[i-1]      dy");
fprintf(fp10,"\n  n    C3ey     ey         C4ey
         C5ey     dy
C6ey    dstore");
c=1;
for(i=1;i<grid->ie_tot;i++){
for(j=0;j<grid->je_tot;j++){
for(k=1;k<grid->ke_tot;k++){
if(dstore[i][j][k] || dy[i][j][k] || hx[i][j][k]
               || hx[i][j][k-1] ||
hz[i][j][k] || hz[i-1][j][k]){
fprintf(fp9,"\n%i  %g   %g   %g   %g   %g   %g   %g
%g",c,C1ey[i][j][k],dstore[i][j][k],C2ey[i][j][k],
hx[i][j][k],hx[i][j][k-1],
hz[i][j][k],hz[i-1][j][k],dy[i][j][k]);
}
if(ey[i][j][k] || dy[i][j][k] || dstore[i][j][k]){
fprintf(fp10,"\n%i  %g   %g   %g   %g   %g   %g
%g",c,C3ey[i][j][k],ey[i][j][k],C4ey[i][j][k],
C5ey[i][j][k],dy[i][j][k],
C6ey[i][j][k],dstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp9);
fclose(fp10);
//=================================================
FILE *fp11,*fp12;
fp11=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           dz_cpp_nmax.txt","wt");
fp12=fopen("C:\\TEMPs\\TEMP - CPP\\C_UPML\\TXT\\
           ez_cpp_nmax.txt","wt");
fprintf(fp11,"\n n   C1ez    dstore    C2ez
            hy       hy[i-1]
hx      hx[j-1]     dz");
fprintf(fp12,"\n n   C3ez    ez        C4ez     C5ez     dz
C6ez    dstore");
c=1;
for(i=1;i<grid->ie_tot;i++){
for(j=1;j<grid->je_tot;j++){
for(k=0;k<grid->ke_tot;k++){
if(dstore[i][j][k] || dz[i][j][k] || hy[i-1][j][k]
 || hy[i][j][k] ||
hx[i][j][k] || hx[i][j-1][k]){
fprintf(fp11,"\n%i  %g   %g   %g   %g   %g   %g   %g
%g",c,C1ez[i][j][k],dstore[i][j][k],C2ez[i][j][k],
hy[i][j][k],hy[i-1][j][k],
hx[i][j][k],hx[i][j-1][k],dz[i][j][k]);
}
if(ez[i][j][k] || dz[i][j][k] || dstore[i][j][k]){
fprintf(fp12,"\n%i  %g   %g   %g   %g   %g   %g
%g",c,C3ez[i][j][k],ez[i][j][k],C4ez[i][j][k],
C5ez[i][j][k],dz[i][j][k],
C6ez[i][j][k],dstore[i][j][k]);
}
c++;
   }
  }
 }
fclose(fp11);
fclose(fp12);
   } // end
  }
 return 0;
}
```

PUC-Rio - Certificação Digital Nº 0310507/CB

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo