HEITOR HONDA FEDERICO

PROBLEMAS INVERSOS: MODELAGEM DE PROPAGAÇÃO DE ONDAS SONORAS E DIFUSÃO POR CELLULAR AUTOMATA APLICADO AO DESENVOLVIMENTO DE UM DETECTOR DE BARREIRAS REFLEXIVAS

Trabalho de formatura apresentado à
Escola Politécnica da Universidade de
São Paulo para a obtenção do título de
Engenheiro Mecatrônico

Aprovado
10,0
Kawano

Nota final
9,5 (nove e cinco)

22/01/04

São Paulo
2003

HEITOR HONDA FEDERICO

PROBLEMAS INVERSOS: MODELAGEM DE PROPAGAÇÃO DE ONDAS SONORAS E DIFUSÃO POR CELLULAR AUTOMATA APLICADO AO DESENVOLVIMENTO DE UM DETECTOR DE BARREIRAS REFLEXIVAS

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Engenheiro Mecatrônico

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Alexandre Kawano

São Paulo
2003

À minha avó "batcham", que sempre acreditou em mim,
até mesmo quando eu duvidava.

AGRADECIMENTOS

Ao Prof. Dr. Alexandre Kawano, amigo e orientador deste trabalho, por ter me dado oportunidade de trabalhar em um projeto tão interessante quanto desafiador, pelo incentivo e pela paciência quase milenar.

À minha família, pelas condições que proporcionaram todo o desenrolar deste trabalho por noites a fio.

Aos meus amigos Parra e Roberto, pelas noites e noites de discussão sobre o tema.

Ao meu amigo Douglas, pela força nos momentos finais.

Aos meus amigos politécnicos Sydnei, Mário, Chen, Sérgio, Rodrigo, Fábio, Daniel, Eduardo, pelo incentivo e pela camaradagem.

RESUMO

O objetivo deste trabalho é a modelagem e a implementação da propagação de ondas sonoras, difusão e um detector de barreiras reflexivas. Para tanto, utilizamos uma ferramenta matemática chamada *cellular automata* que visa, por meio da aplicação de regras locais, a determinação do comportamento de um sistema no tempo. Realizamos o levantamento bibliográfico necessário, não só *celular automata* como também de outras ferramentas necessárias para o desenvolvimento do sistema. A linguagem de programação utilizada foi o *C++* e as saídas gráficas foram feitas com arquivos de imagem e OpenGL. Com essas informações, implementamos um programa para simular a propagação de ondas e difusão em um dado meio. A partir deste programa desenvolvemos sistemas de manipulação de índices de refração e determinação de barreiras e corpos reflexivos através de algoritmos genéticos. Em tudo o que desenvolvemos utilizou-se tecnologia de *software* livre, não havendo, portanto, nenhum tipo de gasto em licenças ou restrições de uso do programa resultante, inclusive usando a mesma licensa para os produtos desse trabalho.

ABSTRACT

The primary goal of this project is to develop a sound wave model, a diffusion model, and a barrier detector. Both phisical models are based on a mathematical tool called *cellular automata*, which states that we are able to model complex behavior by simulating simple local rules in a simple system, instead of attempting to solve the problem at once. Throughout this monography, you will find theoretical information on *cellular automata*, *genetic algorithms* and *inverse problems* necessary to develop the models contained here. The programming language used was *C++* and the graphical outputs were obtained through *ppm* files and *OpenGL* based applications written by the author. While this project was being carried out, only free software was used. Therefore, there were no expenses with software licenses. All programs contained here are licensed by the *Free Software Foundation - GPL.*

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

Capítulo 1

Introdução e objetivos

Um dos grandes problemas de engenharia é a análise de sistemas com comportamentos complexos cujas soluções analíticas, quando existem, apresentam-se de uma forma extremamente complicada e de uma grande dificuldade de implementação em computador. Um desses problemas é a simulação do comportamento de ondas, que tem sua descrição na forma de uma equação simples, mas que apresenta para cada tipo novo de condição de fronteira uma solução não necessariamente simples. Outro problema parecido encontra-se no processo de difusão. Nele, essa modelagem se torna ainda mais complexa pois não conseguimos ainda contar com certas propriedades como a reversibilidade das ondas.

Neste trabalho propomos resolver a equação da onda e a equação de difusão de uma maneira diferente da tradicional. Ao invés de controlarmos todo o sistema e termos problemas para implementar as equações conforme desejamos, queremos, aplicando regras locais e promovendo interações locais dentro de uma malha, convergir para o resultado macroscópico da propagação de uma onda.

Isso se faz extremamente importante nos dias de hoje em termos de engenharia, uma vez que cada vez mais sensores baseados em sons são utilizados para inspeção de equipamentos. A análise de propagação de ondas e vibração constituí uma excelente maneira de se realizar inspeções e estudos de uma maneira não destrutiva, e que responda perguntas de engenharia com alguma precisão.

Através da simulação de ondas sonoras podemos também auxiliar o projeto acústico de uma sala, um teatro, um anfiteatro ou até mesmo instalações de fábricas e parques. Podemos executar inspeções, desde as mais delicadas estruturas como levantamento de superfícies, até grandes estruturas como pontes e estações submersas, que hoje contam com inspeções visuais. Por isso é intenção também aqui o desenvolvi-

mento de um detector de barreiras reflexivas baseado nos modelos obtidos.

1.1 Objetivos

Neste trabalho pretendemos realizar a:

1. modelagem matemática da propagação de ondas sonoras, difusão e resolução de problemas inversos;
2. validação matemática dos modelos obtidos;
3. implementação computacional dos modelos físicos;
4. solução dos problemas inversos no detector de barreiras;

Colocamos também como objetivos secundários:

1. obtenção de um modelo mais genérico de ondas, uma vez que a matemática é toda a mesma;
2. programação de uma saída gráfica em OpenGL;
3. uso somente de programas livres.

Capítulo 2

Revisão teórica: autômatos celulares

2.1 Introdução

Neste capítulo faremos uma breve descrição do que é e como funcionam os autômatos celulares, em latim *cellular automata*. Explicaremos, brevemente, um pouco do histórico dessa ciência, desenvolvendo ao final alguns exemplos de autômatos celulares simples.

2.2 Histórico

O estudo de autômatos celulares é uma ciência recente na matemática, datando do final de 1940. Seu pioneiro incontestável foi *John von Neumann*, o mesmo pesquisador que se envolveu nas pesquisas dos primeiros computadores digitais [4], seu trabalho mais conhecido. Apesar disso, seu trabalho com o modelo de autômato celular constituí também o primeiro modelo de computação para grandes sistemas.

Inicialmente *von Neumann* tencionava imitar o funcionamento biológico do cérebro humano para construir uma máquina que resolvesse problemas complexos [5]. No entanto, a idéia era muito mais ambiciosa do que a resolução de problemas.

Ele pensava que uma máquina, que conseguisse seguir os mesmo padrões de um cérebro, conteria também mecanismos de auto controle e reparos. Essa máquina portanto conseguiria se construir sozinha a partir de algum material disponível. Considerando então o problema de um ponto de vista mais formal, ele queria uma máquina que fosse capaz de se auto replicar. Este foi, portanto, um dos primeiros estudos sobre vida artificial, um ser vivo que pudesse ser programado e evoluísse dentro de uma máquina.

Ele construiu então um sistema que tinha uma grade de células discretas, com es-

tados internos bem definidos e propôs que o sistema evoluí-se em passos discretos no tempo seguindo regras que eram válidas para toda a grade. Essas regras envolviam somente as próprias células e suas vizinhas mais imediatas. Esse sistema inteiramente discreto inventado por *von Neumann* são agora chamados *CA - Cellulares automatas.*

O primeiro sistema construído por *von Neumann* foi uma grade bidimensional com alguns milhares de células. Cada célula continha 29 estados possíveis e a regra de evolução consistia em somar o valor da célula com as das vizinhas imediatas. Esse sistema nunca pode ser implementado devido sua alta complexidade [4]. Mesmo assim, *von Neumann* conseguiu programar estruturas replicantes com capacidades de gerar novas estruturas iguais. Isso não pode ser considerado nem sequer uma forma de vida primitiva, mas uma máquina capaz de construir máquinas de complexidade igual e maior do que a sua própria é um resultado muito interessante.

Muitos outros cientistas deram continuidade as pesquisas de *von Neumann* e hoje autômatos celulares podem e são aplicados para resolver uma grande variedade de problemas. O estudo de vida artificial se desenvolveu todo como um outro campo de pesquisa, mas não existe motivo para que autômatos celulares não sejam mais explorados mesmo nesse campo da ciência.

2.3 Idéia básica

A idéia básica do funcionamento dos autômatos celulares vem da natureza. Partimos do princípio que qualquer sistema, por mais complexo que seja sempre funciona a partir de interações locais entre partículas. Como resultado final vemos por exemplo um ser humano formado. Se analisarmos o desenvolvimento desse indivíduo e relembrarmos deste a formação do ovo até o nascimento, percebemos que o ser humano é formado por interações locais entre células que se diferenciam a partir de proteínas que *excitam* as células.

Um desmoronamento de terra depois de uma grande chuva também pode ser observado dessa forma. Num mundo macro observamos uma grande massa de terra deslizando um barranco, mas podemos observar o sistemas como o movimento sincronizado entre uma série de partículas, governados por leis locais de aderência que resulta no movimento do corpo de terra.

A idéia do autômato celular é simular esse comportamento. Através da modelagem de grandes grades de partículas bem definidas e regras de iterações locais, busca-se o comportamento complexo de um deslizamento de terra por exemplo. No caso desse

trabalho, busca-se o comportamento de ondas aplicando princípio de Huygens.

2.4 Cellular automata unidimensional

Podemos definir o autômato celular então como sendo uma ferramenta matemática para sistemas discretos com transições discretas. Dado um sistema unidimensional, temos como forma geral para a transição de um autômato celular a equação 2.1:

$$x_i^{t+1} = f(x_{i-r}^t, \ldots, x_i^t, \ldots, x_{it+r}^t) \qquad f : Z_k^{2r+1} \rightarrow Z_k, \tag{2.1}$$

sendo x_i^t denota o estado do ponto i no instantes $t+1$, f representa a regra de transição definida para o *automata* e r é um inteiro maior que zero que especifica o raio de influência dos vizinhos no estado futuro do ponto i e Z_k representa o domínio no qual o *automata* atua. O mais simples *automata* tem $r = 1$ e $k = 2$, descrito por Wolfram [3] como o *elementar*. As regras de um *celllular automata* podem ser também descritas através de uma tabela de regras. Analisemos um exemplo com um *automata* elementar.

Temos portanto o seguinte domínio a ser analisado:

$$\boxed{x_{i-1}^t, x_i^t, x_{i+1}^t}$$

com uma definição de regras dada por 2.2:

$$f : 0, 1^3 \rightarrow \{0, 1\}, \tag{2.2}$$

sendo x_i^t o valor do local i em um instante t. Neste caso, a tabela de regras tem 2^3 tuplas de 3. A tabela é definida então para cada operação:

Tabela 2.1: Tabela de regras para *autômatos celulares* elementar

tupla	f
000	a_0
001	a_1
010	a_2
011	a_3
100	a_4
101	a_5
110	a_6
111	a_7

sendo

$$f(xyz) = a_i$$

A definição das regras de transição serão feitas ainda mais adiante no trabalho, levando em conta as equações que regem os movimentos das ondas. Conforme já executado em um trabalho anterior, uma das opções seria tratar o problema da propagação como um problema de difusão. Neste caso, uma aplicação das regras, dado que seja matematicamente possível, seria utilizando as *Cadeias de Markov*.

2.4.1 Regras Locais

A primeira curiosidade sobre autômatos unidimensionais é que as regras locais recebem nomes conforme o valor que recebem. Por exemplo:

$$\underbrace{111}_{0}\ \underbrace{110}_{0}\ \underbrace{101}_{0}\ \underbrace{100}_{0}\ \underbrace{011}_{0}\ \underbrace{010}_{1}\ \underbrace{001}_{0}\ \underbrace{000}_{0}$$

Repare que os números da transição formam um número em binário 00000100 que é igual ao número 4 em base 10. Essa regra é chamada regra 4.

No caso dos autômatos celulares com uma única dimensão existe ainda mais uma consideração a ser feita. Segundo Wolfram [5], duas regras não essenciais devem ser impostas a essas regras para discussão da sua legalidade.

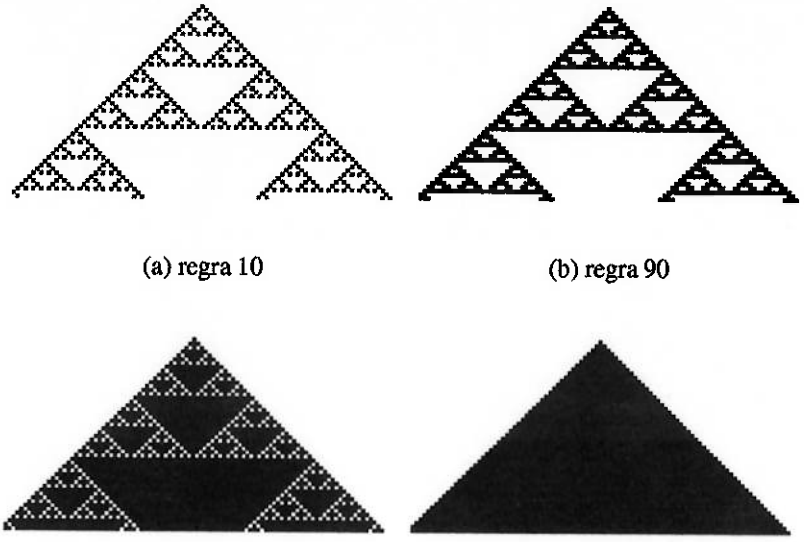
1. O estado 000 deve permanecer inalterado, ou seja 000;

2. As regras precisam ter simetria entre as mesmas, ou seja, o resultado de 001 deve ser o mesmo de 100.

Essas regras implicam em algumas conseqüências conforme apontado em [5] e [7]. Regras legais devem ter a forma $a_1, a_2, \ldots a_i \ldots, a_{n-1}, 0$.

Podemos portanto, no caso do autômato celular elementar determinar 32 regras locais de evolução legais. Na figura 2.1 podemos observar algumas dessas regras se desenvolvendo no tempo.

Figura 2.1: Cellular automata unidimensional: regras legais

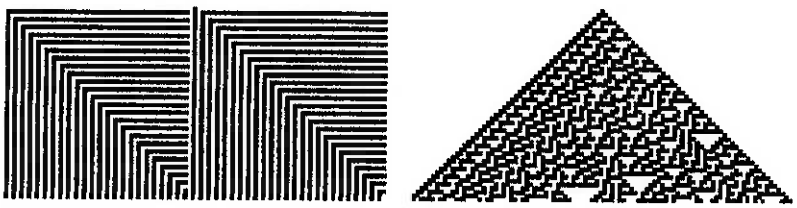


(a) regra 10 (b) regra 90

(c) regra 102 (d) regra 254

Conforme pode se observar, através de regras simples consegue-se comportamentos complexos. A questão agora é utilizar esses comportamentos complexos para aplicações práticas e resolução de problemas. A figura 2.2 demonstra algumas regras ilegais.

Figura 2.2: Cellular automata unidimensional: regras ilegais



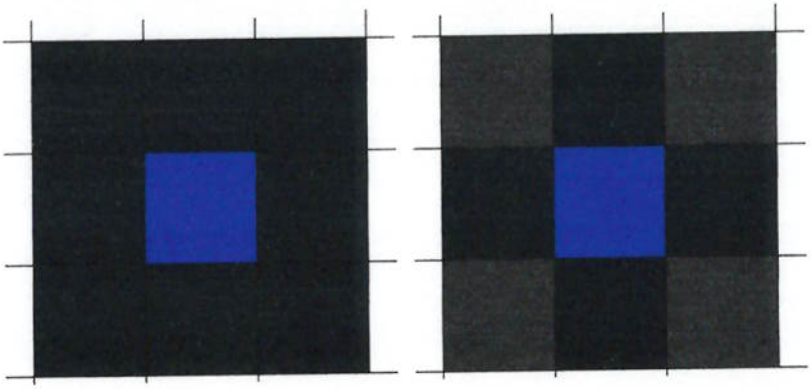
(a) regra 13 (b) regra 30

2.5 Autômatos celulares bidimensionais

A idéia dos autômatos celulares não mudam quando passamos para problemas com mais de uma dimensão. O problema passa a ser a maneira como enxergamos o espaço.

Existem duas maneiras básicas em que o espaço pode ser enxergado. Neste trabalho utilizamos o mapeamento de vizinhas de *von Neumann*. O mapeamento de vizinhas de Moore também é bastante conhecido [6].

Figura 2.3: Vizinhanças



(a) Moore (b) Neumann

Capítulo 3

Revisão teórica: ondas sonoras

3.1 Introdução

Neste capítulo apresentamos fundamentos teóricos do equacionamento e solução de ondas e será feito o relacionamento dessa matemática com ondas sonoras. O capítulo começa com uma abordagem geral do que é uma onda e sua matemática, passando então para ondas sonoras e seu equacionamento específico.

A resolução da matemática apresentada aqui consiste na maneira mais tradicional de resolução e simulação de ondas.

3.2 A equação da onda: ondas progressivas

3.2.1 Forma geral

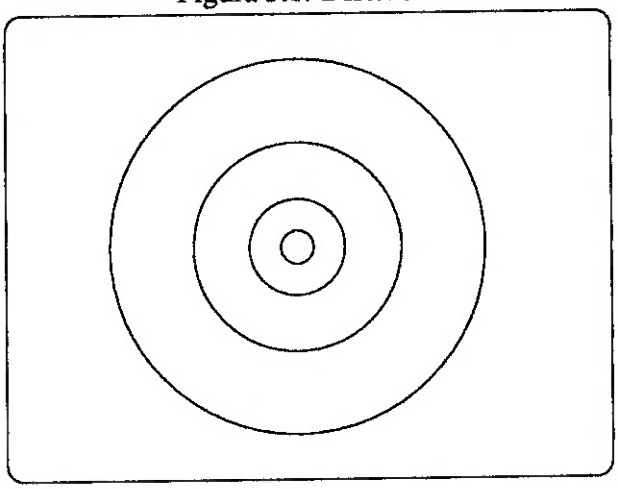
Consideremos um meio em equilíbrio onde é introduzido um distúrbio ϕ que se propaga no meio com velocidade constante c [11]. Essa grandeza ϕ no caso ainda não vai ser definida, bastando saber que ela existe e representa um distúrbio. Como ela se propaga, o valor de ϕ depende então do lugar no meio onde ela se encontra e em que instante ela esta sendo medida. Se tiramos uma fotografia quando $t = 0$, podemos observar o distúrbio e teremos $\phi = f(x)$. Caso a onda se propague sem mudanças, esse formato nunca muda, só muda a posição no espaço que vai para ct. Quando mudamos a origem de onde fazemos as medições portanto em um ponto $x = X + ct$, a equação do distúrbio mudaria para:

$$\phi = f(x - ct) \tag{3.1}$$

e para propagação negativa

$$\phi = f(x + ct) \tag{3.2}$$

Figura 3.1: Distúrbio



Essas equações (3.1 e 3.2) são a forma mais geral de equação de uma onda se movendo com velocidade constante [11].

3.2.2 Ondas harmônicas

Ondas harmônicas são um tipo de onda onde a forma do distúrbio segue a seguinte forma (3.3):

$$\phi_{t=0} = A\cos(mx) \tag{3.3}$$

Temos portanto, de acordo com 3.1

$$\phi_{t=0} = A\cos m(x - ct) \tag{3.4}$$

Podemos agora determinar $\lambda = 2\pi/m$ resultando na equação 3.5, onde λ é o comprimento de onda, ou seja, a distância que a onda leva sem se repetir:

$$\phi_{t=0} = A\cos\frac{2\pi}{\lambda}(x - ct) \tag{3.5}$$

Temos agora condições portanto de determinar o período τ que a onda leva para percorrer um comprimento de onda:

$$\tau = \frac{\lambda}{c} \tag{3.6}$$

A freqüência n é o número de comprimentos de onda que passam por unidade de tempo, que podemos concluir que é:

$$\tau = \frac{\lambda}{c} \tag{3.7}$$

3.2.3 Equação de onda

A onda pode ainda ser escrita na forma diferencial. Segundo [11] [8] [9] existe uma equação que reúne todas as soluções propostas que é:

$$\nabla^2\phi = \frac{1}{c^2}\frac{\partial^2\phi}{\partial t^2} \tag{3.8}$$

que também pode ser escrita como:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2\phi}{\partial t^2} \tag{3.9}$$

Incluindo a excitação, acabamos com a seguinte equação:

$$F = \frac{1}{c^2}\frac{\partial^2\phi}{\partial t^2} - \nabla^2\phi \tag{3.10}$$

A solução dessa equação diferencial resulta na onda da maneira como conhecemos.

3.3 Ondas sonoras

A matemática vista na seção anterior é válida para qualquer tipo de onda linear em um meio. Som é uma onda como outra qualquer e portanto também segue a mesma matemática. A definição de onda sonora não passa de uma percepção do ser humano.

Quando se fala em ondas mecânicas, transmitidas por vibração, todas as ondas que estão entre 20Hz e 20000Hz são percebidas pelo ouvido humano. A essa sensação é dado o nome de som. No entanto, como qualquer outra onda, som segue a mesma matemática e tudo o que é feito para modelar um onda mecânica, também pode ser aplicado para som. A diferença é que, por exemplo, no caso do ar, a grandeza ϕ da equação 3.8 é uma pequena diferença de pressão que se propaga.

No caso de sólidos, a onda é elástica, ou seja, se propaga por deformação, mas a matemática envolvendo a propagação continua sendo a mesma [8].

Tendo isto em mente, toda a matemática desenvolvida neste trabalho visa a modelagem de ondas de uma forma geral.

Capítulo 4

Revisão teórica: Difusão

4.1 Introdução

Nesta parte do trabalho faremos uma breve descrição dos mecanismos de difusão, como ocorrem e como podem ser modelados matemáticamente da maneira tradicional. Será abordado aqui as leis de *Fick* e também uma breve discussão sobre a utilidade deste fenomeno em ciências de engenharia.

4.2 Por que estudar difusão?

A difusão está presente, ou pode modelar diversos fenômenos diferentes de grande importância para engenharia. O exemplo mais comum e talvez o mais fácil de ser lembrado é o tratamento de materiais para *dopagem* como por exemplo na fabricaçao de circuitos integrados. Mas mesmo em outros campos temos problemas para serem atacados.

Se pensarmos nos problemas físicos temos situações como tratamento de metais por calor, onde as partículas se movem de acordo com o fenômeno da difusão [16], tratamento superficial de metais e até mesmo derramamento de petróleo onde temos difusão de micelas de óleo na água do mar. Mas podemos ir além disso.

Se observarmos esse fenômeno como difusão de estados, onde estados podem representar o que queremos, podemos ter difusão de estados de degradação de uma máquina ou estrutura por exemplo.

Isso mostra como esse fenômeno é mais comum e ocorre com mais frequência do que imaginamos.

4.3 O fenômeno

Oservando a difusão pela ótica da mecânica, perceberemos que nada mais se trata do que pequenas partículas que se movem para pontos vizinhos de sua posição atual [1]

4.3.1 As leis de Fick

Quando estudamos difusão a questão que levantamos é qual a quantidade de matéria encontramos em determinado ponto em um determinado instante. Para isso, primeiro definimos a taxa de transferência de massa chamado fluxo de difusão J [16]. Podemos definir:

$$J = \frac{M}{At} \tag{4.1}$$

Escrevendo 4.1 na forma diferencial temos:

$$J = \frac{1}{A}\frac{dM}{dt} \tag{4.2}$$

Se J é constante no tempo, temos difusão em regime permanente e podemos determinar um gradiente de concentração $\frac{dC}{dx}$. Dessa maneira podemos definir novamente a equação 4.1.

$$J = -D\frac{dC}{dx} \tag{4.3}$$

Essa equação (4.3) é as vezes chamada de *primeira lei de Fick* [16], onde D é o coeficiente de difusão de um determinado meio. Essa lei no entanto só analisa o estado permanente.

Para estudar o comportamento dinâmico da difusão usamos a derivada parcial da equação 4.3.

$$\frac{\partial C}{dt} = \frac{\partial}{\partial x}\left(D\frac{\partial C}{\partial x}\right) \tag{4.4}$$

Se o coeficiente de difusão for constante para todo o meio, podemos simplificar 4.4 para:

$$\frac{\partial C}{dt} = D\left(\frac{\partial^2 C}{\partial x^2}\right) \tag{4.5}$$

[1] Essa idéia irá nortear futuramente o desenvolvimento do modelo usando autômatos celulares.

A equação 4.5 é chamada *segunda lei de Fick* [16]. Passando para um espaço tridimensional e adicionando a excitação temos:

$$F = \frac{\partial C}{dt} - \frac{1}{d}\left(\nabla^2 C\right) \tag{4.6}$$

onde $1/d = D$.

Capítulo 5

Revisão teórica:Algoritmos genéticos

5.1 Introdução

Quando um problema de otimização é abordado, procuramos um máximo ou mínimo global de maneira a minimizar os custos que uma falha ou um resultado não otimizado poderia acarretar ao cliente ou ao investidor. O que acontece é que, geralmente nos deparamos com um problema, seja ele matemático ou não, com uma gama de soluções possíveis tão grande que torna inviável a procura exaustiva, ou seja testar todas as possibilidades.

Por exemplo, digamos que desejamos decidir um plano de inspeção para uma estrutura mecânica que tenha 1000 componentes. A simples decisão de inspecionar ou não já nos gera 2^{1000} possibilidades de planos de inspeção. Mas este problema ainda não se aproxima de um problema real. Aumentemos um pouco a complexidade do exemplo, tomemos agora uma estrutura com 10 componentes, onde num período de dois anos possamos realizar 2 inspeções. Digamos também, que o intervalo mínimo entre as inspeções é de um mês. Temos portanto um problema já mais complicado onde a estrutura nos fornece por si só 2^{10} possibilidades e o tempo, uma combinação de 24 elementos tomados 2 a 2 nós dá mais 66 possibilidades. Temos portanto um total de $66x1024 = 67584$ possibilidades de inspeção.

Já podemos perceber neste exemplo a importância do desenvolvimento de um método que nos possibilite achar qual dessas possibilidades nos dá o melhor resultado sem que tenhamos que analisar cada uma delas. Este problema tem ainda mais um agravante na dificuldade de transformá-lo numa função matemática. Mas mesmo que não tivesse um formula matemática, e sim um comportamento estatístico, o problema ainda tem que ser resolvido.

O algoritmo genético é justamente uma tentativa de obter um método que encontre este ótimo de maneira a não calcular todas as possibilidades seguindo leis biológicas, simulando assim as possibilidades como uma população de indivíduos onde, quanto mais otimizado o indivíduo maior sua chance de sobreviver.

5.1.1 Fatos históricos:

Algoritmos genéticos são um método de busca baseada na mecânica da seleção natural e genética [13] [12]. A partir de uma população de indivíduos aleatórios, seleciona-se o mais apto a sobreviver e expõe-se a população a fenômenos que simulam as ocorrências de crossovers e mutações genéticas. Apesar desse aspecto aleatório, os algoritmos genéticos tendem a achar um ponto ótimo com uma relativa boa performance.

Este tipo de Algoritmo foi primeiro desenvolvido por *John Holland*, alguns de seus colegas e de seus estudantes na Universidade de Michigan. Seus objetivos eram os de :

1. abstrair e explicar os processos de adaptação natural;
2. desenvolver programas que mantivessem as características principais dos mecanismos de sistemas naturais.

Esta abordagem acabou trazendo muitas descobertas tanto para as ciências naturais quanto para a computação.

5.1.2 Nomenclatura

A nomenclatura utilizada nos algoritmos vem parte das ciências biológicas e parte das ciências de computação, justamente por ter em sua origem a intenção de simular em computador o comportamento evolucionário natural. A tabela (5.1) que pode ser encontrada em [14] apresenta um resumo de alguns dos termos mais comuns.

5.2 Modelo do problema: fenótipos e genótipos

A parte talvez mais complicada de qualquer problema é determinar um modelo da situação real. Esses modelos tem em si uma precisão que podem comprometer todo o processo. No caso do algoritmo genético, temos que determinar, em que elementos matemáticos (operações, letras, números ...) e quantos precisaríamos para modelar o

Tabela 5.1: Lista de termos de GA

termo (GA)	correspondente
Cromossomo	Solução
Genes	Parte da Solução
Locus	Posição do Gene
Alelos	valor de um gene
Fenótipo	solução descodificada
Genótipo	solução codificada

problema. Isto é a determinação do genótipo dos cromossomos, do problema codificado.

Mas comparar cromossomos codificados não é o nosso objetivo. Para tanto, transformamos o genótipo no fenótipo, que tem que ser um valor comprável entre os cromossomos para determinar qual deles é o ótimo. No caso de funções matemáticas este problema se simplifica.

O algoritmo genético adotado é baseado principalmente na proposta de [14]:

1. geração de um população inicial;
2. seleção da nova geração;
3. crossover;
4. mutação;

Essas operações executadas em seqüencia faz com que surja, ao acaso, um cromossomo dentro da população mais apto, com uma "fitness"melhor ao meio. Esse cromossomo é o cromossomo ótimo do sistema também. Abordaremos em seguida cada um dos processos explicando melhor o seu funcionamento.

5.2.1 Geração de uma população inicial e seleção da nova geração

O algoritmo genético baseia-se na teoria de Darwin para procurar um cromossomo ótimo. Isso requer que uma população inicial seja gerada para que a mesma posso "evoluir"até o surgimento de um cromossomo ótimo. Como na natureza, a população inicial é gerada com características aleatórias, ou seja, os genótipos iniciais "surgem"ao acaso.

A seleção de uma nova geração é o processo de seleção, entre os cromossomos existentes, de quais cromossomos irão continuar na população e em que proporção. Existem uma série de métodos para se fazer esta seleção, e este é talvez, uma das partes mais importantes do algoritmo genético.

No caso da roda da vida, usamos uma técnica para que possamos gerar um dado ponderado. Ou seja, quando geramos números aleatórios para sorteio, existem probabilidades diferentes para cada cromossomo de passar para a próxima geração Para fazer a seleção da população seguinte, usamos uma técnica chamada de *roleta russa*, pois ela consegue gerar uma população aleatória com respeito as probabilidades de cada um dos cromossomos. O processo da *roleta russa* consiste em 3 passos:

1. calcular a fitness total da população:

$$F = \sum_{k=1}^{popsize} (eval(v_k)) \tag{5.1}$$

2. calcular a probabilidade de seleção p_k para cada cromossomo v_k:

$$p_k = \frac{eval(v_k)}{F}, \text{ para k = 1, 2, ..., popsize} \tag{5.2}$$

3. calcular a probabilidade acumulada q_k :

$$q_k = \sum_{j=1}^{k} (p_j) \tag{5.3}$$

Com esses dados em mão vale lembrar que pela evolução, as características surgem por acaso e os membros da população mais aptos às adversidades do ambiente sobrevivem, enquanto os menos aptos desaparecem. Isso é levado em consideração quando se faz a seleção dos "sobreviventes"para a próxima geração. A seleção pode ser resumida em dois passos:

1. gera-se um número aleatório r no intervalo [0, 1];

2. se r é menor ou igual a q_1, seleciona-se o primeiro cromossomo para a nova população, caso contrário, seleciona-se o *k*ésimo cromossomo tal que $q_{k-1} < r \leq q_k$.

A outra maneira, que obteve mais sucesso neste trabalho, foi a da *sobrevivência do mais forte*. Este método consiste no método de amostragem determinístico, conforme

descrito em [14]. Neste caso, determinamos quem irá para a próxima geração simplesmente determinando quem dentro da população atual é o mais forte. Desta maneira colocamos pelo menos uma cópia dos melhores cromossomos na população seguinte evitando assim também o surgimento de super-cromossomos. Outro fato é que isso implica implicitamente na aplicação do elitismo, conforme descrito em [14], uma vez que o melhor cromossomo sempre passa para a próxima geração obrigatoriamente.

A vantagem deste método é que com ele mantemos uma variedade genética maior por mais tempo, e temos uma população de bom resultados que evoluem juntas. Usando a *Roda da vida*, o surgimento de um super-cromossomo que, sendo um ótimo local domina a população com diversas cópias, faz com que o programa vá para um máximo local e estacione neste máximo.

5.2.2 Crossover

O processo de crossover utilizado em [14] e o usado aqui consiste em, sortear dois cromossomos e um locus e inverter os genes dos dois cromossomos a partir do locus sorteado. Esse processo melhor explicado no esquema do algoritmo escrito abaixo:

$k \leftarrow 0$
while $k \leq popsize$ **do**
  $r_k \leftarrow$ número aleatório entre [0, 1]
  if $r_k \leq$probabilidade de crossover **then**
    seleciona v_k como um cromossomo para crossover
  end if
  $k \leftarrow k + 1$
end while

5.2.3 Quando os genes mutam

O processo de mutação corresponde a chance de um gene mudar seu valor. Em um cromossomo compostos de genes que podem tomar o valor de 0, ou 1, consiste na probabilidade de um gene com 1 se tornar 0 na nova geração, e vice-versa. No caso dessa simulação, todos os genes tem a mesma chance de mutar, portanto, uma probabilidade de mutação de 10% dos genes em uma população com 5 cromossomos de 10 genes, espera-se que 5 genes sofram mutação, mas não necessariamente um em cada cromossomo.

5.3 O espaço amostral

Conforme explicado acima, com algoritmos genéticos, trabalhamos com uma amostra da população total de possibilidades de maneira que a melhor possibilidade venha a eventualmente fazer parte desta amostra Uma técnica que é usada para aumentar a variedade de genes que vão para a seleção da próxima geração, é ao invés de substituir os pais pelos cromossomos gerados pela *mutação* e pelo *crossover*, somá-los a população original. Desta maneira, por ocasião do select, tantos os pais quanto os filhos terão chance de ser selecionados conforme o valor do seu fenótipo. É o chamado *Enlarged Sampling Space*, conforme descrito em [14].

5.4 Cuidados

A aplicação de algoritmos genéticos não pode ser feita de maneira irresponsável a qualquer tipo de problema. Problemas discretos não tem maiores problemas, mas quando se falam de problemas contínuos, a discretização gera um problema pois multiplica a quantidade zeros locais. Isso pode ser resolvido aumentando a discretização do projeto, mas quanto mais isso é feito o número de variáveis para otimizar aumenta. Se o número de variáveis aumenta muito o algoritmo fica lento e uma abordagem com outros métodos, como métodos matemáticos, pode se tornar mais atrativa.

No entanto, GA não deixa de ser uma ferramenta extremamente poderosa, e no caso de *Cellulares Automatas* uma boa opção pela própria natureza do problema.

Capítulo 6

Problemas inversos

Neste trabalho propomos o estudo de problemas inversos. Para tanto, se faz necessário uma revisão não só do modelo (que denominaremos futuramente problema direto), mas também da teoria que envolve problemas inversos. A determinação de parâmetros ótimos de modelos duvidosos é uma operação tão difícil quanto incerta, o que levou muitos teóricos e práticos a se debruçarem sobre o problema. Nesta parte do trabalho é feita uma breve revisão dessa teoria para que o leitor possa entender os resultados descritos na próxima sessão, baseados principalmente no trabalho de [15]. É feita nesta parte do trabalho também a aplicação mais direta dessa teria ao problema analisado.

6.1 O espaço de modelos e o espaço de dados

6.1.1 Espaço de modelos

O primeiro passo no estudo de problemas inversos é a definição de espaço de modelos e espaço de dados. Quando analisamos um sistema físico desenvolvemos um modelo que se aproxima do resultado real baseado em alguns parâmetros. Por exemplo, podemos pensar em uma bala sendo atirada por um canhão. Temos neste caso, para a altura uma equação como:

$$Pos_{final} = Pos_{inicial} + Vel_{inicial} * tempo - 5 * \frac{(tempo)^2}{2} \tag{6.1}$$

Temos aqui um modelo para a movimentação vertical da bala no tempo e alguns valores chaves para o bom funcionamento do modelo. Neste caso, essa equação representa o nosso modelo direto do problema físico e esses valores chaves os parâmetros do modelo.

Temos a mesma coisa para o caso de propagação de ondas, onde no caso temos que a equação 3.8 é a maneira direta de se resolver o problema e o parâmetro do modelo seria a velocidade propagação. A propagação de ondas no entanto pode ter diversos tipos de modelos diretos. Neste trabalho usamos um outro método tanto para a propagação de ondas como difusão usamos o modelo baseado em *autômatos celulares*. No caso de propagação de ondas por exemplo podemos pensar como parâmetros do modelo a velocidade da onda, índice de refração, precisão da malha, precisão da malha de tempo e características de reflexão do modelo.

Neste trabalho, conhecemos o índice de refração, as precisões de malhas e tempos e queremos as características de reflexão do sistema.

Temos então um conjunto com todas as opções de parâmetros possíveis para o modelo, um espaço onde podemos localizar e mapear essas opções a qual chamamos de *espaço de modelos* [15]. Podemos mapear esse espaço de modelos de maneira que conseguimos executar operações matemáticas sobre ele. Esse mapa é chamado M. Para resolver um problema direto, sempre escolhemos um ponto em M. Um ponto em M, denominamos m que é um conjunto de parâmetros do espaço de modelos. Geralmente, m é representado por uma matriz $1 \times \alpha$ com os valores necessários. Como um abuso de linguagem chamamos M de espaço de modelos também.

6.1.2 Espaço de dados

Da mesma maneira que o modelo tem alguns parâmetros que definem o comportamento do sistema, existem parâmetros que dizemos observáveis. Isso é dado por exemplo pela posição final na equação 6.1, ou no caso do modelo de propagação de onda a intensidade do distúrbio em um dado ponto do espaço. A esse conjunto de dados observáveis damos o nome de *espaço de dados* [15].

Dando o mesmo tratamento que demos ao *espaço de modelos* podemos mapear o espaço de dados em um mapa D onde cada ponto d indica uma possível solução.

Como abuso de linguagem chamamos D de espaço de dados também.

6.1.3 Espaço conjunto $M \times D$

Da mesma maneira que [15], definimos aqui um espaço conjunto $X = D \times M$. Dessa maneira, observamos aqui elementos $x = (d, m)$, chamados simplesmente de parâmetros do sistema. Calculamos aqui sempre probabilidades dentro deste espaço *físico*.

6.2 O modelo direto

Temos então dois espaços para analisar , M e D. Para tanto necessitamos que exista uma relação entre M e D de maneira que consigamos dado m, um ponto d. Neste trabalho a função que leva $M \rightarrow D$ é o modelo por *autômatos celulares*. Neste trabalho denominaremos essa 'função' $CA(m)$.

6.3 Estado de conhecimento perfeito e total ignorância

Devido a diversos tipos de incertezas de modelos, dados, parâmetros, os resultados de problemas inversos sempre envolvem probabilidades [15]. Para tanto, é bom definirmos três conceitos que serão utilizados neste trabalho:

1. função densidade de probabilidade;
2. estado de conhecimento perfeito;
3. estado de total ignorância;

6.3.1 FDP:função densidade de probabilidade

Determinemos P como uma probabilidade em X, a função densidade de probabilidade dá o valor de P em um dado ponto de X, x. Essa distribuição de probabilidades em X pode ser escrita como:

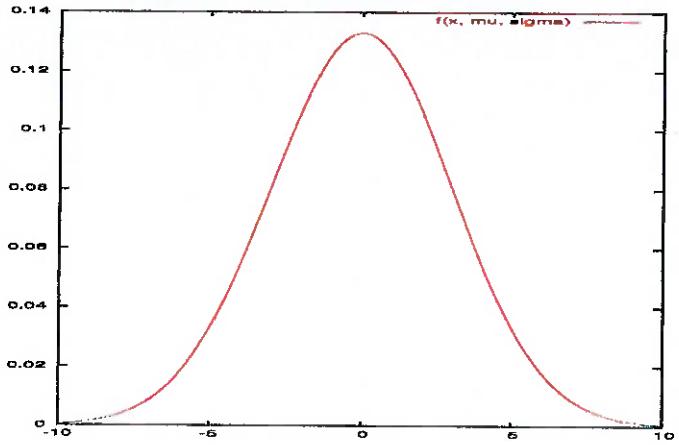
$$P(A) = \int_A f(x)dx \tag{6.2}$$

Uma distribuição conhecida e que é amplamente utilizada neste trabalho é a distribuição normal que pode ser escrita como:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(x-\mu_x)^2}{\sigma^2}\right)} \tag{6.3}$$

O gráfico dele pode ser visto, no caso de $\sigma = 3$ e $\mu = 0$:

Figura 6.1: Distribuição Normal



6.3.2 Estado de conhecimento perfeito

Definimos estado de perfeito conhecimento da mesma maneira que [15]. Ele existe quando temos certeza absoluta de que um dado valor de x é $x = x_0$. Podemos representar essa certeza por uma *f.d.p.* em forma de Delta de Dirac.

$$f(x) = \delta(x - x_0) \tag{6.4}$$

Essa distribuição da probabilidade nula para qualquer ponto $x \neq x_0 = 0$ e tem probabilidade 1 para o ponto $x = x_0$. Esse tipo de distribuição é utilizada quando o erro de algum tipo de dado é desprezível frente a outros erros que podem ocorrer.

6.3.3 Estado de total ignorância

Essa distribuição de probabilidades representa o estado de total ignorância sobre um valor. Para nosso caso, vamos considerar que todas as possibilidades de parâmetro iguais, ou seja a f.d.p. aqui é uma constante. Essa distribuição implica que não temos nenhum conhecimento anterior que nos ajude a determinar a certeza do ponto.

No entando, o estado de total ignorância, ou a total falta de informação sobre um sistema não implica que a distribuição seja constante, dependendo do tipo de modelo abordado.

6.4 Aplicação da teoria Baysiana

Temos então agora que resolver o problema inverso de maneira a determinarmos a probabilidade de um certo conjunto de parâmetros terem gerados os dados observados. Teremos aqui portanto, dados observados através de algum tipo de instrumento, alguma informação sobre os parâmetros além do modelo físico.

6.4.1 Informação a priori

Toda vez que vamos resolver um problema inverso temos alguma ou nenhuma idéia de onde o resultado pode estar. Essa f.d.p. chamamos de f.d.p. a priori.

Quando não temos nenhuma idéia do que pode ser a resposta, a função densidade de probabilidade a priori se torna a função do estado total de ignorância.

6.4.2 Informação a posteriori

Essa informação é dada que se consegue uma vez que analisamos os dados. É como se observássemos os dados que temos, a função a priori e tomássemos uma decisão de uma nova f.d.p. que define a solução. O ponto de maior probabilidade da função a posteriori é a solução que desejamos.

6.4.3 Solução do problema inverso

Temos agora então que determinar uma curva de probabilidades a posteriori para encontrarmos o máximo, nossa solução. Temos então uma seqüencia de dados que foram tirados da amostra com alguma probabilidade e queremos os parâmetros do modelo.

$$f(m|d_{obs}) \tag{6.5}$$

Instrumentação

Temos agora então que realizar medições de amostras tanto iniciais quando para encontrar os parâmetros que queremos. Realizamos o teste então, ou seja, executamos o modelo direto e lemos em um sensor um valor. Esse valor é uma distribuição de probabilidades devido a leitura.

Temos então primeiro uma diferença entre dado real, dado amostrado, e dado observado. Consideramos durante este trabalho que não existe incertezas entre o dado que o sensor leu, ou seja, temos uma distribuição em delta de Dirac, ou seja:

$$d_{observado} = \delta(d - d_{obtido}) \tag{6.6}$$

Consideramos no entanto que existem erros de observação, portanto temos distribuições dos dados observados. Visto isso temos uma certa probabilidade então de o dado observado ser real, que para este trabalho consideramos normal.

Erros do modelo

Quando traçamos qualquer modelo, existem erros inerentes ao modelo. Isso faz com que os nossos resultados tenham mais uma distribuição de acordo com o modelo. Neste caso, consideremos que este erro está embutido na distribuição normal descrita acima.

função a priori

Para este caso consideramos que não existe nenhuma informação anterior conforme o parâmetro. Por isso, nossa função a priori é o estado total de ignorância, ou seja a probabilidade dos parâmetros é uma constante igual para todos os parâmetros.

6.4.4 Desenvolvimento teórico

Consideremos uma função densidade de probabilidade no espaço conjunto $D \times M$ $f(d, m)$. Podemos definir então as probabilidades marginais:

$$f_M(m) = \int_D f(\mathrm{d}, m)d\mathrm{d} \tag{6.7}$$

$$f_D(m) = \int_M f(\mathrm{d}, m)dm \tag{6.8}$$

As probabilidades condicionais podem ser definidas com o teorema de Bayes:

$$f_{D|M} = \frac{f(\mathrm{d}_{obs}, m)}{\int_M f(\mathrm{d}_{obs}, m)dm} \tag{6.9}$$

$$f_{M|D} = \frac{f(\mathrm{d}_{obs}, m)}{\int_D f(\mathrm{d}_{obs}, m)d\mathrm{d}} \tag{6.10}$$

onde temos que , de 6.7,6.8,6.9,6.10:

$$f_{M|D}(m|d_{obs}) = \frac{f_{D|M}(\mathrm{d}_{obs}|m)f_M(m)}{\int_M f_{D|M}(\mathrm{d}_{obs}|m)f_M(m)dm} \tag{6.11}$$

$$f_{D|M}(d|m_0) = \frac{f_{M|D}(m_0|\mathrm{d}) f_D(\mathrm{d})}{\int_D f_{M|D}(m_0|\mathrm{d}) f_D(\mathrm{d}) d\mathrm{d}} \tag{6.12}$$

A equação 6.11 é a solução do nosso problema inverso, onde f_m é a função a priori, a $f(d_{obs}|m)$ são incertezas do modelo gerar aqueles dados, e a a $f(m|d_{obs})$ é a função posteriori.

Com essas considerações em mente, podemos desenvolver para cada problema inverso que tentemos resolver, seja ele qual for (neste caso equação de ondas) aplicando essa teoria caso a caso.

Capítulo 7

Modelagem da propagação

7.1 Introdução

Nesta parte do trabalho abordamos o desenvolvimento teórico de modelos de propagação de onda usando cellular automata e também demonstramos matematicamente, para o caso bidimensional, a validade do método.

7.2 TLM:Transmition Line Matrix

Transmition Line Matrix (TLM) é uma técnica de simulação numérica muito usada a princípio para resolução de problemas de propagação eletromagnética [7]. A idéia principal é obter um modelo discreto para ser resolvido através de métodos numéricos, de maneira que as aproximações se limitam a malha de discretização escolhida. Ou seja, é um modelo que resolve pequenas frações do problema a ser modelado, podendo portanto ser utilizado como regra local em uma malha de autômatos celulares. No caso da propagação de ondas, o TLM se aplica como o princípio de Huygens.

A idéia é considerar cada ponto das frentes de onda como uma nova fonte de onda [10] para a frente de onda seguinte. Dessa maneira podemos considerar o processo como o espalhamento de vários componentes de onda. O resultado final da superposição de efeitos é o que realmente importa, que são as ondas como queremos ver. Mais uma vez, através de regras simples e locais, buscamos um comportamento complexo e difícil de ser simulado.

Uma das maneiras de se determinar as regras locais para a fonte de ondas do TLM é através do movimento sincronizado de partículas [4]. Considerando as distancias relativas entre as partículas em uma malha, vemos que a propagação de deformação

acontece na forma de uma onda.

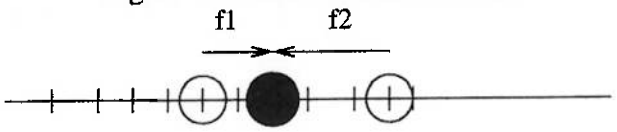
7.3 Caso unidimensional

Inicialmente modelamos o caso unidimensional a partir da literatura usando o modelo físico já mencionado [4]. A propagação unidimensional é mais limitada em termos de aplicação e por isso não foi estudada de maneira mais profunda.

7.3.1 Propagação livre

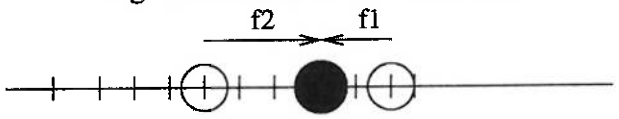
Consideremos uma malha unidimensional com partículas pretas e partículas brancas alternadas como o da figura 7.1

Figura 7.1: Onda unidimensional



Observando a figura 7.1 definimos duas grandezas $f1$ e $f2$ como sendo a distância entre a partícula preta e a partícula branca a esquerda e direita respectivamente. A idéia é que a partícula negra se mova de maneira que f1 e f2 fiquem invertidos (figura 7.2):

Figura 7.2: Onda unidimensional



Na próxima movimentação, temos as partículas brancas se movendo da mesma maneira, de modo que, f1 e f2 se propagam na forma de uma onda unidimensional. Temos então como resultado para uma dada partícula i:

$$\begin{pmatrix} f_1(i+1,t+1) \\ f_2(i-1,t+1) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} f_1(i,t) \\ f_2(i,t) \end{pmatrix} \tag{7.1}$$

Temos então a regra local para um autômato celular que modela ondas unidimensionais. Seguimos o princípio de Huygens, pois cada ponto da frente de onda é uma nova fonte geradora de ondas e percebemos que um distúrbio ocorrido no meio da malha se propaga para os dois lados na forma de duas ondas.

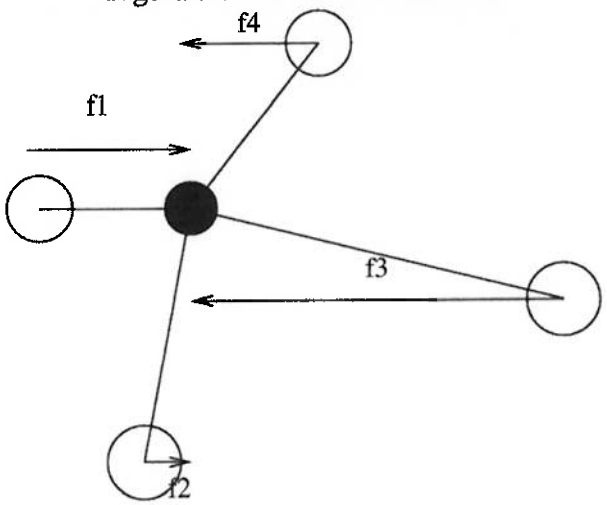
7.4 Caso bidimensional

O caso bidimensional de propagação é muito útil na engenharia uma vez que existem uma série de problemas que podem ser resolvidos a partir dele. Além disso, necessita de uma capacidade computacional bem menor do que a tridimensional e os mecanismos todos já conseguem ser simulados.

7.4.1 Propagação livre

Podemos usar o mesmo raciocínio usado para o caso unidimensional para modelarmos uma onda que se move em uma malha bidimensional. No caso agora teremos para cada partícula preta quatro partículas brancas 7.3

Figura 7.3: Onda bidimensional



Vamos definir agora a separação entre a partícula preta emissora e as partículas brancas.

$$\begin{aligned} f_1(i,j,t) &= x_{i-1,j}(t) - x_{i,j}(t) \\ f_2(i,j,t) &= x_{i,j-1}(t) - x_{i,j}(t) \\ f_3(i,j,t) &= x_{i+1,j}(t) - x_{i,j}(t) \\ f_4(i,j,t) &= x_{i,j+1}(t) - x_{i,j}(t) \end{aligned} \tag{7.2}$$

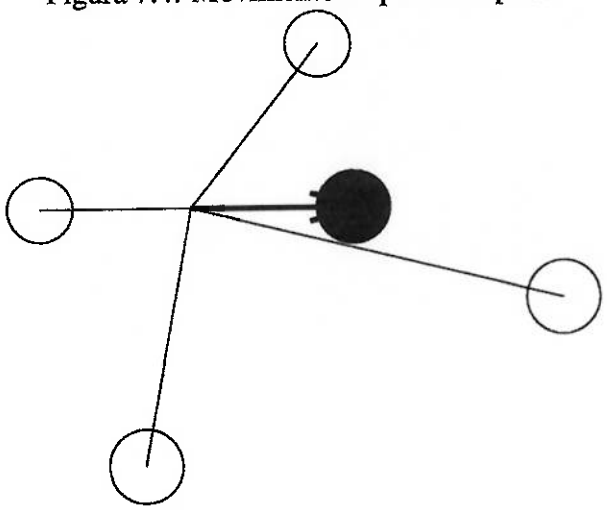
Considerando que a partícula preta está no centro do sistemas de coordenadas podemos definir que o centro de massa X_{CM} das partículas brancas é:

$$X_{CM} = \frac{1}{4}(f_1 + f_2 + f_3 + f_4) \tag{7.3}$$

A regra de evolução continua a mesma, a partícula preta se move para a posição oposta ao centro de massa das partículas brancas (figura 7.4), portanto a nova posição da partícula preta é:

$$x_{i,j}(t+1) = 2X_{CM} = \frac{1}{2}(f_1 + f_2 + f_3 + f_4) \tag{7.4}$$

Figura 7.4: Movimento da partícula preta



Como conseqüência, as distâncias entre as partículas brancas e pretas se alteram e podemos definir:

$$\begin{aligned}
f_1(i+1,j,t+1) &= 2X_{CM} - x_{i+1,j} = \tfrac{1}{2}(f_1 + f_2 + f_3 + f_4) - f3 \\
f_2(i,j+1,t+1) &= 2X_{CM} - x_{i,j+1} = \tfrac{1}{2}(f_1 + f_2 + f_3 + f_4) - f4 \\
f_3(i-1,j,t+1) &= 2X_{CM} - x_{i-1,j} = \tfrac{1}{2}(f_1 + f_2 + f_3 + f_4) - f1 \\
f_4(i,j-1,t+1) &= 2X_{CM} - x_{i,j-1} = \tfrac{1}{2}(f_1 + f_2 + f_3 + f_4) - f2
\end{aligned}$$

Que nos leva a 7.5:

$$\begin{aligned}
f_1(i+1,j,t+1) &= \tfrac{1}{2}(f_1 + f_2 - f_3 + f_4) \\
f_2(i,j+1,t+1) &= \tfrac{1}{2}(f_1 + f_2 + f_3 - f_4) \\
f_3(i-1,j,t+1) &= \tfrac{1}{2}(-f_1 + f_2 + f_3 + f_4) \\
f_4(i,j-1,t+1) &= \tfrac{1}{2}(f_1 - f_2 + f_3 + f_4)
\end{aligned} \tag{7.5}$$

De 7.5 percebemos que existe uma relação entre a propagação de f_is das partículas

brancas em relação ao estado atual da partícula preta (emissora). Temos portanto:

$$\begin{pmatrix} f_1(i+1,j,t+1) \\ f_2(i,j+1,t+1) \\ f_3(i-1,j,t+1) \\ f_4(i,j-1,t+1) \end{pmatrix} = W \begin{pmatrix} f_1(i,j,t) \\ f_2(i,j,t) \\ f_3(i,j,t) \\ f_4(i,j,t) \end{pmatrix} \tag{7.6}$$

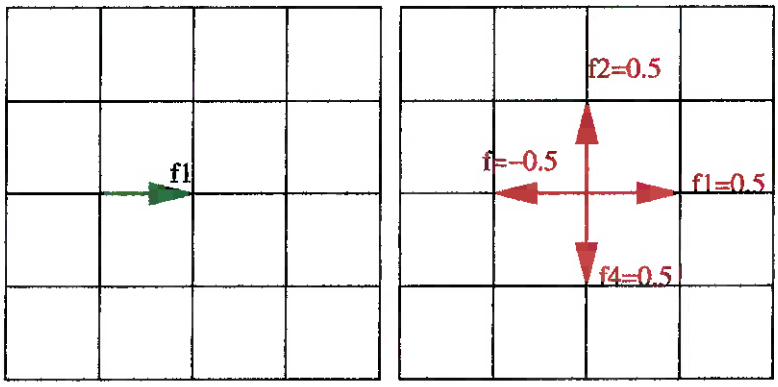
No caso, W é a regra local de mudança que será implementada nos autômatos celulares.

$$W = \frac{1}{2} \begin{pmatrix} 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \end{pmatrix} \tag{7.7}$$

A equação 7.7 implementa então a propagação dos f_is na forma de uma onda, e no caso temos aqui propagação livre, sem perda de energia. É importante notar que eventualmente f_is vão tomar valores fracionários, ou seja as partículas saltarão para fora da malha. Por isso esse não é um autômato celular puro como no caso unidimensional, e sim uma grade de Boltzmann.

Tendo isso em mente, o comportamento que iremos observar neste caso não é mais a posição em si das partículas, mas somente a propagação de f_is em uma grade como na figura abaixo:

Figura 7.5: Dispersão bidimensional em 1 passo

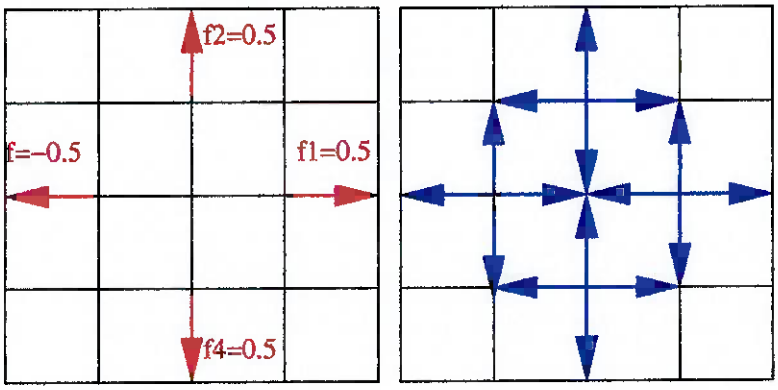


(a) distúrbio (b) primeira dispersão

Podemos perceber que o sinal de f_3 se inverte. Isso acontece pois vai no sentido contrário do sentido original do distúrbio. É importante também perceber que a

propagação como esperado conserva o distúrbio inicial e gera uma frente de onda bidimensional. Cada ponto agora dessa nova frente se torna uma nova fonte de propagação (figura 7.6).

Figura 7.6: Dispersão bidimensional continuação



(a) novas fontes (b) nova frente de onda

7.4.2 Reflexão

Uma propagação, seja onde for não é eterna. Existem sempre fronteiras e condições de contorno a serem consideradas. Neste momento estamos mais interessados em reflexões totais.

A idéia é bem simples: tudo o que chega a um certo ponto, que é reflexivo, retorna a origem com ou sem inversão de fase. Intuitivamente podemos escrever a matriz necessária para que isso seja conseguido:

$$W_{reflexivo} = \begin{pmatrix} 0 & 0 & \pm 1 & 0 \\ 0 & 0 & 0 & \pm 1 \\ \pm 1 & 0 & 0 & 0 \\ 0 & \pm 1 & 0 & 0 \end{pmatrix} \tag{7.8}$$

O sinal se torna positivo ou negativo dependendo se quer ou não inversão de fase e é importante notar que essa definição vale para um ponto, e não para uma fronteira. Com essa definição podemos definir corpos reflexivos no meio da malha sendo analisada, por exemplo.

7.4.3 Absorção

Outra característica que o meio pode ter é o de absorver a energia. Isso não é particularmente útil para a propagação de som, uma vez que a perda de energia é tão pequena devido a velocidade de propagação [8], mas é útil para fronteiras e corpos imersos.

O desenvolvimento é o mesmo da propagação livre com a diferença que definimos um α absorvedor de maneira que:

$$x_{i,j}(t+1) = x_{i,j}(t) + \alpha(X_{CM} - x_{i,j}) \tag{7.9}$$

Com α indo de 0 a 2 onde:

- $\alpha = 0$ a partícula não se move;
- $\alpha = 1$ move para o centro de massa;
- $\alpha = 2$ movimento livre.

Podemos então desenvolver de novo a formulação da matriz W para termos a nova solução:

- para f_1:

$$\begin{aligned}
f_1(t+1) &= x_{i,j}(t+1) - x_{i+1,j} \\
f_1(t+1) &= 2X_{CM} + (1-\alpha)x_{i,j} - x_{i+1,j} \\
f_1(t+1) &= \tfrac{\alpha}{4}(x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) + \\
&\quad + (1-\alpha)x_{i,j} - x_{i+1,j} \\
f_1(t+1) &= \left(\tfrac{\alpha}{4} - 1\right)(x_{i,j-1} - xi,j) + \tfrac{\alpha}{4}(x_{i-1,j} - x_{i,j}) + \\
&\quad + \tfrac{\alpha}{4}(x_{i,j-1} - x_{i,j}) + \tfrac{\alpha}{4}(x_{i,j+1} - x_{i,j})
\end{aligned}$$

  Temos então 7.10:

$$f_1(t+1) = \frac{\alpha}{4}f_1 + \frac{\alpha}{4}f_2 + \left(\frac{\alpha}{4} - 1\right)f_3 + \frac{\alpha}{4}f_4 \tag{7.10}$$

- para f_2:

$$\begin{aligned}
f_2(t+1) &= \tfrac{\alpha}{4}(x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) + \\
&\quad + (1-\alpha)x_{i,j} - x_{i,j+1} \\
f_2(t+1) &= \left(\tfrac{\alpha}{4} - 1\right)(x_{i,j+1} - xi,j) + \tfrac{\alpha}{4}(x_{i+1,j} - x_{i,j}) + \\
&\quad + \tfrac{\alpha}{4}(x_{i,j-1} - x_{i,j}) + \tfrac{\alpha}{4}(x_{i-1,j} - x_{i,j})
\end{aligned}$$

  Temos então 7.11:

$$f_2(t+1) = \frac{\alpha}{4}f_1 + \frac{\alpha}{4}f_2 + \frac{\alpha}{4}f_4 + \left(\frac{\alpha}{4} - 1\right)f_4 \tag{7.11}$$

- para f_3

$$f_3(t+1) = \left(\frac{\alpha}{4} - 1\right) f_1 + \frac{\alpha}{4} f_2 + \frac{\alpha}{4} f_3 + \frac{\alpha}{4} f_4 \tag{7.12}$$

- para f_4

$$f_4(t+1) = \frac{\alpha}{4} f_1 + \left(\frac{\alpha}{4} - 1\right) f_2 + \frac{\alpha}{4} f_3 + \frac{\alpha}{4} f_4 \tag{7.13}$$

Das equações 7.10, 7.11, 7.12 e 7.13 podemos retirar a matriz

$$W_{atenuada} = \begin{pmatrix} \frac{\alpha}{4} & \frac{\alpha}{4} & \left(\frac{\alpha}{4} - 1\right) & \frac{\alpha}{4} \\ \frac{\alpha}{4} & \frac{\alpha}{4} & \frac{\alpha}{4} & \left(\frac{\alpha}{4} - 1\right) \\ \left(\frac{\alpha}{4} - 1\right) & \frac{\alpha}{4} & \frac{\alpha}{4} & \frac{\alpha}{4} \\ \frac{\alpha}{4} & \left(\frac{\alpha}{4} - 1\right) & \frac{\alpha}{4} & \frac{\alpha}{4} \end{pmatrix} \tag{7.14}$$

Temos novamente:

- α=0 $W_{atenuada} = W^{-}_{reflexivo}$
- α=2 $W_{atenuada} = W_{livre}$

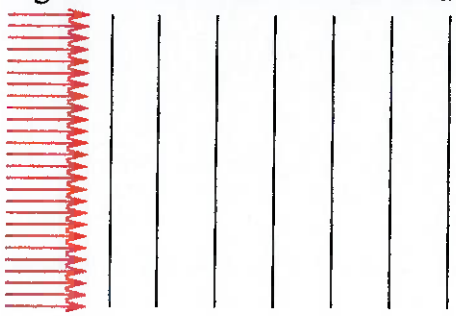
Podemos então determinar uma equação que forneça a matriz atenuada a partir das outras duas conhecidas 7.15:

$$W_{atenuada} = \frac{\alpha}{2} W_{livre} + \left(1 - \frac{\alpha}{2}\right) W^{-1}_{reflexivo} \tag{7.15}$$

7.4.4 Fontes de onda

Determinação de fontes de onda bidimensionais podem ser problemáticas quanto ao formato. Temos que, para construir a fonte saber com que freqüência precisamos perturbar o sistema (entrada em degrau) ou se vamos dar um único estímulo ao sistema (delta de Dirac). Para a formação de fontes de onda mais complexas como uma batedor de ondas em um tanque, necessitamos construir a *imagem* da fonte com fontes de onda pontuais (*princípio de Huygens*). Por exemplo, uma frente de onda reta é obtida com uma estímulos em linha sincronizados, que podem ou não ser em freqüência figura (7.7). As figuras 7.5 e 7.6 ilustram entradas em freqüência.

Figura 7.7: Frente de onda em linha



7.4.5 Mudança de velocidade

Utilizando a formulação de Boltzman e considerando o meio como se fosse um fluido se movimentando, podemos determinar uma outra matriz bastante interessante, onde podemos determinar a velocidade permitida para cada local [4].

O estudo é feito a partir do estudo de difusão, mas no caso, o resultado é tornado específico para que possamos recuperar a matriz 7.7 [4]. A idéia é criar mais um fator f_0 que representaria uma certa quantidade que não se movimenta naquele passo. Isso faria o ajuste de velocidade. A matriz é:

$$W_{velocidade} = \\ = \frac{1}{2n^2} \begin{pmatrix} 2n^2-4 & 2\sqrt{n^2-1} & 2\sqrt{n^2-1} & 2\sqrt{n^2-1} & 2\sqrt{n^2-1} \\ 2\sqrt{n^2-1} & 1 & 1 & 1-2n^2 & 1 \\ 2\sqrt{n^2-1} & 1 & 1 & 1 & 1-2n^2 \\ 2\sqrt{n^2-1} & 1-2n^2 & 1 & 1 & 1 \\ 2\sqrt{n^2-11} & 1-2n^2 & 1 & 1 & \end{pmatrix} \tag{7.16}$$

Essa matriz é valida para qualquer $n \geq 1$, sendo n o índice de refração do meio dado por:

$$n^2 = \frac{v^2}{2c^2} \tag{7.17}$$

onde c é a velocidade do meio livre e v é a velocidade que se quer.

7.4.6 Validação do método

Com toda essa formulação mostrada já é possível implementar ondas para uso de engenharia. É necessário no entanto que se demonstre que a formulação obtida *realmente* representa a mesma onda apresentada no capítulo 3. Isso foi feito através da aplicação

de diferenças finitas e de uma dedução mais matemática através da própria equação da onda levada ao limite. Em ambos os casos se determinou uma relação de dispersão ($\frac{\Delta x}{\Delta t}$) para qual o modelo é válido. Conforme esperado, essa relação é igual nos dois casos.

Limite

A idéia proposta em [4] é que quando $\Delta_x = \Delta_y = \Delta$ e $\Delta t = \tau$, quando levamos Δ para 0, iremos ter uma relação entre Δ e τ. Definindo o vetor $\vec{r} = (x, y)$, podemos escrever:

$$f_i(\vec{r} + \tau \vec{c}_i, t + \tau) = \sum_{j=1}^{4} W_{i,j} f_j(\vec{r}, t) \tag{7.18}$$

onde $W_{i,j}$ é dada pela expressão 7.7. Para que validemos o método, consideremos a seguinte solução para o sistema:

$$f_i(\vec{r}, t) = A_i e^{i(\omega t + \vec{k}\vec{r})} \tag{7.19}$$

Substituindo essa expressão temos que 7.19 é solução desde que w e k respeitem[4]:

$$e^{i\omega\tau} = \frac{1}{2}(\kappa \pm i\sqrt{4 - \kappa^2}) \tag{7.20}$$

sendo $\kappa = \cos(k_1\Delta) + \cos(k_2\Delta)$, onde $\vec{k} = (k_1, k_2)$. A expansão de Taylor da expressão 7.20 usando $k^2 = k_1^2 + k_2^2$ é:

$$i\omega\tau - \frac{\omega^2\tau^2}{2} + O(\tau^3) = -\frac{\Delta^2 k^2}{4} + O(\Delta^4) \pm i\left[\frac{\Delta k}{\sqrt{2}} + O(\Delta^3)\right] \tag{7.21}$$

Segundo [4] temos dois fenômenos aqui presentes, difusão e convecção. Por isso τ deve ter componentes da ordem de Δ e Δ^2. Podemos definir então:

$$\tau = \epsilon\tau 1 + \epsilon^2\tau_2 \tag{7.22}$$

$$\Delta = \epsilon\Delta_1 \tag{7.23}$$

Agora podemos levar ϵ para zero em vez dos termos em tempo e espaço de maneira a descobrirmos a relação de Δ e τ para qual a solução 7.19 é válida. Comparando os membros de mesma ordem da expressão 7.21 temos:

$$\omega = \pm\frac{1}{\sqrt{2}}\frac{\Delta_1}{\tau_1}k \tag{7.24}$$

Que é a relação esperada [4]

$$c_0 = \frac{1}{\sqrt{2}}\lim_{\epsilon\to 0}\frac{\Delta}{\tau} : \tag{7.25}$$

Diferenças finitas

Uma outra maneira de se provar que o sistema é a equação de onda foi proposto por [10], apesar de não muito claro. Ao invés de um desenvolvimento direto fizemos um desenvolvimento em direções, no caso aqui x e y de maneira a encontramos a partir do modelo em autômato celulares a resolução da equação da onda (3.8) com diferenças finitas.

Partindo da expressão 7.7, podemos observar a propagação em cada eixo e depois sobrepor os resultados. Temos então:

$$f_i(\overrightarrow{x} + \Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t + \Delta_t) = W_x f_i(\overrightarrow{x}, \overrightarrow{y}, t) \tag{7.26}$$

$$f_i(\overrightarrow{x}, \overrightarrow{y} + \Delta_{\overrightarrow{y_i}}, t + \Delta_t) = W_y f_i(\overrightarrow{x}, \overrightarrow{y}, t) \tag{7.27}$$

Sendo:

$$W_x = \frac{1}{2}\begin{pmatrix} 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 \end{pmatrix} \tag{7.28}$$

$$W_y = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix} \tag{7.29}$$

Podemos definir agora:

$$f_{i^*}(\overrightarrow{x} + \Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t + \Delta_t) = \frac{1}{2}\sum_{j=1}^{4} f_j(\overrightarrow{x}, \overrightarrow{y}, t) - f_i(\overrightarrow{x}, \overrightarrow{y}, t) \tag{7.30}$$

$$f_{i^*}(\overrightarrow{x}, \overrightarrow{y} + \Delta_{\overrightarrow{y_i}}, t + \Delta_t) = \frac{1}{2}\sum_{j=1}^{4} f_j(\overrightarrow{x}, \overrightarrow{y}, t) - f_i(\overrightarrow{x}, \overrightarrow{y}, t) \tag{7.31}$$

sendo:

$$\overrightarrow{x_i} = -\overrightarrow{x_{i^*}}$$
$$\overrightarrow{y_i} = -\overrightarrow{y_{y^*}}$$

A partir de 7.30 e 7.31 podemos definir:

$$P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t) = \frac{1}{2}\sum_{i=1}^{4} f_i(\overrightarrow{x}, \overrightarrow{y}, t) \tag{7.32}$$

Assim, podemos dizer que $P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t)$ é:

$$P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t) = \frac{1}{2}\sum_{i=1}^{4} f_i(\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t) \tag{7.33}$$

Lembrando agora que estamos analisando os eixos separadamente temos:

$$\begin{aligned} &P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t) = \\ &= \tfrac{1}{2}\textstyle\sum_{j=1}^{2} \left\{ (f_{j,para\,\overrightarrow{x}}\,\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t) + f_{j,para\,\overrightarrow{y}}(\overrightarrow{x}, \overrightarrow{y}, t + \Delta_t) \right\} \end{aligned} \tag{7.34}$$

Aplicando agora 7.26, 7.27, 7.30 e 7.31 sucessivamente temos:

$$I = \begin{cases} f_{j,parax}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = \frac{1}{2}\sum_{k=1}^{4}\left\{f_k(\overrightarrow{x}+\Delta_{\overrightarrow{x_j}}, \overrightarrow{y}, t)\right\} - f_{j^*}(\overrightarrow{x}+\Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t) \\ f_{j,paray}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = \frac{1}{2}\sum_{k=1}^{4}\left\{f_k(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_j}}, t)\right\} - f_{j^*}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_i}}, t) \end{cases}$$

$$I = \begin{cases} f_{j,parax}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = P_{ak}(\overrightarrow{x}+\Delta_{\overrightarrow{x_j}}, \overrightarrow{y}, t) - f_{j^*}(\overrightarrow{x}+\Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t) \\ f_{j,paray}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = P_{ak}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_j}}, t) - f_{j^*}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_i}}, t) \end{cases}$$

Expandindo novamente:

$$II = \begin{cases} f_{j^*}(\overrightarrow{x}+\Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t) = \\ = \frac{1}{2}\sum_{k=1}^{4}\left\{f_k(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t)\right\} - f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \\ f_{j^*}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_i}}, t) = \\ = \frac{1}{2}\sum_{k=1}^{4}\left\{f_k(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t)\right\} - f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \end{cases}$$

$$II = \begin{cases} f_{j^*}(\overrightarrow{x}+\Delta_{\overrightarrow{x_i}}, \overrightarrow{y}, t) = \\ = P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) - f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \\ f_{j^*}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_i}}, t) = \\ = P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) - f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \end{cases}$$

De I e II temos:

$$III = \begin{cases} f_{j,parax}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = P_{ak}(\overrightarrow{x}+\Delta_{\overrightarrow{x_j}}, \overrightarrow{y}, t) - P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) + f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \\ f_{j,paray}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = P_{ak}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_j}}, t) - P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) + f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \end{cases}$$

Usando III e voltando para 7.34:

$$\begin{aligned} P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) = \\ = \frac{1}{2}\sum_{j=1}^{2} & \begin{array}{c} P_{ak}(\overrightarrow{x}+\Delta_{\overrightarrow{x_j}}, \overrightarrow{y}, t) - P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) + f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \\ + \\ P_{ak}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_j}}, t) - P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) + f_j(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) \end{array} \end{aligned} \tag{7.35}$$

Juntando os termos temos:

$$\begin{aligned} & P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t+\Delta_t) + P_{ak}(\overrightarrow{x}, \overrightarrow{y}, t-\Delta_t) = \\ & = \frac{1}{2}\sum_{j=1}^{2}\left\{P_{ak}(\overrightarrow{x}+\Delta_{\overrightarrow{x_j}}, \overrightarrow{y}, t) + P_{ak}(\overrightarrow{x}, \overrightarrow{y}+\Delta_{\overrightarrow{y_j}}, t)\right\} \end{aligned} \tag{7.36}$$

Somando $-2P_{ak}(\vec{x}, \vec{y}, t)$ nos dois lados da equação temos:

$$\begin{aligned}&\tfrac{1}{2}\sum_{j=1}^{2}\left\{P_{ak}(\vec{x}+\Delta_{\vec{x_j}}, \vec{y}, t)+P_{ak}(\vec{x}, \vec{y}+\Delta_{\vec{y_j}}, t)\right\}-2P_{ak}(\vec{x}, \vec{y}, t)= \\ &=P_{ak}(\vec{x}, \vec{y}, t+\Delta_t)-2P_{ak}(\vec{x}, \vec{y}, t)+P_{ak}(\vec{x}, \vec{y}, t-\Delta_t)\end{aligned} \tag{7.37}$$

A equação 7.37 pode ainda ser escrita da forma:

$$\begin{aligned}&\tfrac{1}{2}\left\{\sum_{j=1}^{2}\left\{P_{ak}(\vec{x}+\Delta_{\vec{x_j}}, \vec{y}, t)+P_{ak}(\vec{x}, \vec{y}+\Delta_{\vec{y_j}}, t)\right\}-4P_{ak}(\vec{x}, \vec{y}, t)\right\}= \\ &=P_{ak}(\vec{x}, \vec{y}, t+\Delta_t)-2P_{ak}(\vec{x}, \vec{y}, t)+P_{ak}(\vec{x}, \vec{y}, t-\Delta_t)\end{aligned} \tag{7.38}$$

Lembrando da expressão 3.8, podemos abri-la em diferenças finitas o que resultará na expressão seguinte:

$$\begin{aligned}&\phi_{t+\Delta_t}^{x,y}-2\phi_t^{x,y}+\phi_{t-\Delta_t}^{x,y}= \\ &=\left(c\tfrac{\Delta_t}{\Delta}\right)^2 \begin{matrix}\phi_t^{x-\Delta,y}-2\phi_t^{x,y}+\phi_t^{x+\Delta,y} \\ + \\ \phi_t^{x,y-\Delta}-2\phi_t^{x,y}+\phi_t^{x,y+\Delta}\end{matrix}\end{aligned} \tag{7.39}$$

Comparando as expressões 7.39 e 7.38, podemos concluir que o modelo de autômatos celulares é a equação de onda quando:

$$\left(c\frac{\Delta_t}{\Delta}\right)^2=\frac{1}{2} \tag{7.40}$$

Da expressão 7.40 podemos concluir que para o nosso modelo ser válido:

$$\frac{\Delta}{\Delta_t}=\sqrt{2}c \tag{7.41}$$

Como não poderia deixar de ser, a expressão 7.41 encontrou a mesma solução de 7.24.

7.5 Caso tridimensional

Extrapolando a idéia proposta para o caso unidimensional e bidimensional [4], desenvolvemos um modelo do que seria a propagação de ondas em três dimensões. Esse modelo não foi implementado computacionalmente por sua grande necessidade de memória, mesmo para casos simples, no entanto, seu equacionamento foi desenvolvido no trabalho e está aqui apresentado.

Para o caso com três dimensões, são necessários definir mais duas grandezas refe-

rentes f_5 e f_6 que representam as distâncias no terceiro eixo. Teremos então:

$$\begin{aligned} f_1(i,j,k,t) &= x_{i-1,j,k}(t) - x_{i,j,k}(t) \\ f_2(i,j,k,t) &= x_{i,j-1,k}(t) - x_{i,j,k}(t) \\ f_3(i,j,k,t) &= x_{i+1,j,k}(t) - x_{i,j,k}(t) \\ f_4(i,j,k,t) &= x_{i,j+1,k}(t) - x_{i,j,k}(t) \\ f_5(i,j,k,t) &= x_{i,j,k-1}(t) - x_{i,j,k}(t) \\ f_6(i,j,k,t) &= x_{i,j,k+1}(t) - x_{i,j,k}(t) \end{aligned} \tag{7.42}$$

O centro de massa agora é, considerando que a partícula emissora está no centro:

$$X_{CM} = \frac{1}{6}(f_1 + f_2 + f_3 + f_4 + f_5 + f_6) \tag{7.43}$$

Temos agora, utilizando a mesma regra dos casos unidimensional e bidimensional:

$$\begin{aligned} f_1(i+1,j,k,t+1) &= 2X_{CM} - x_{i+1,j,k} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_3 \\ &= \tfrac{1}{3}[f_1 + f_2 - 2f_3 + f_4 + f_5 + f_6] \end{aligned} \tag{7.44}$$

$$\begin{aligned} f_2(i+1,j,k,t+1) &= 2X_{CM} - x_{i,j+1,k} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_4 \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 - 2f_4 + f_5 + f_6] \end{aligned} \tag{7.45}$$

$$\begin{aligned} f_3(i+1,j,k,t+1) &= 2X_{CM} - x_{i-1,j,k} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_1 \\ &= \tfrac{1}{3}[-2f_1 + f_2 + f_3 + f_4 + f_5 + f_6] \end{aligned} \tag{7.46}$$

$$\begin{aligned} f_4(i+1,j,k,t+1) &= 2X_{CM} - x_{i,j-1,k} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_2 \\ &= \tfrac{1}{3}[f_1 - 2f_2 + f_3 + f_4 + f_5 + f_6] \end{aligned} \tag{7.47}$$

$$\begin{aligned} f_5(i+1,j,k,t+1) &= 2X_{CM} - x_{i,j,k+1} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_6 \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 - 2f_6] \end{aligned} \tag{7.48}$$

$$\begin{aligned} f_6(i+1,j,k,t+1) &= 2X_{CM} - x_{i,j,k-1} \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 + f_5 + f_6] - f_5 \\ &= \tfrac{1}{3}[f_1 + f_2 + f_3 + f_4 - 2f_5 + f_6] \end{aligned} \tag{7.49}$$

Das expressões acima podemos observar que também temos uma matriz de propagação:

$$W_{3D}^{livre} = \frac{1}{3}\begin{pmatrix} 1 & 1 & -2 & 1 & 1 & 1 \\ 1 & 1 & 1 & -2 & 1 & 1 \\ -2 & 1 & 1 & 1 & 1 & 1 \\ 1 & -2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -2 \\ 1 & 1 & 1 & 1 & -2 & 1 \end{pmatrix} \tag{7.50}$$

Com a expressão 7.50 podemos modelar a propagação livre de uma onda qualquer em três dimensões.

Capítulo 8

Modelagem da difusão

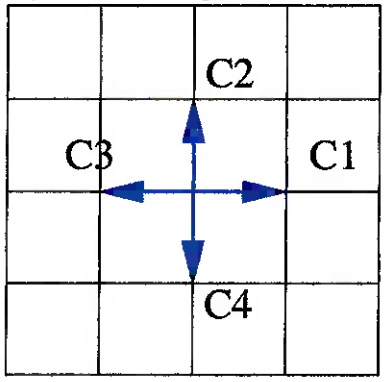
8.1 Introdução

Nesta parte do trabalho abordamos o desenvolvimento teórico da modelagem da difusão por autômatos celulares e também é feita a prova matemática que esse modelo dá o mesmo resultado da lei de Fick, quando levado ao limite. O modelo proposto se encontra em [4]. É abordado aqui o caso bidimensional.

8.2 Microdinâmica

Sabemos que a difusão segue um equação diferencial, a lei de Fick (equação 4.5), mas como já abordado anteriormente, quando observamos o movimento de cada partícula individualmente, temos na verdade *átomos* que pulam para locais vizinhos com uma dada probabilidade. Consideremos um conjunto de vetores de direção como o indicado na figura 8.1:

Figura 8.1: Direções de difusão



Temos então um movimento aleatório de acordo com esse conjunto de vetores da figura 8.1. Tendo isso em mente podemos determinar a regra do autômato celular:

$$n_i\left(\overrightarrow{r}+\lambda\overrightarrow{c_i},t+\tau\right)=\sum_{l=0}^{2d-1}\mu_l(\overrightarrow{r},t)n_{i+l}(\overrightarrow{r},t) \tag{8.1}$$

onde i pertence a $\{1,2,\ldots,2d\}$ e os termos μ_l são valores booleanos que selecionam somente um dos termos do lado direito da equação.

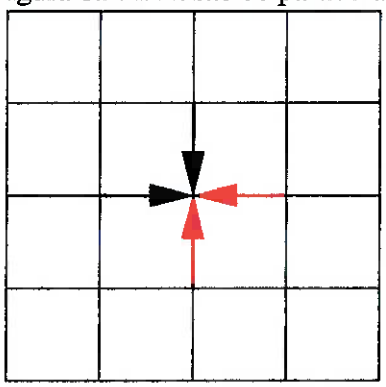
$$\sum_{l=0}^{2d-1}\mu_l=1 \tag{8.2}$$

Podemos então pensar que existem probabilidades p_l, cada uma associada a um μ_l. Dessa maneira usando números aleatórios podemos decidir qual μ_l vai ser igual a 1.

8.2.1 Caso bidimensional

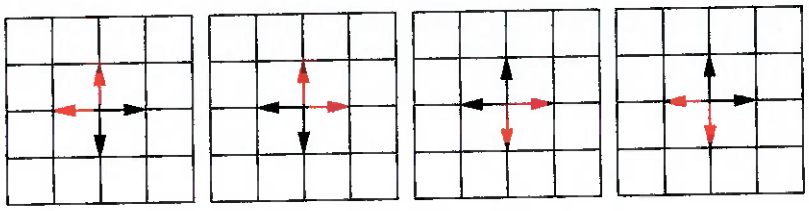
No caso bidimensional, quando as partículas entram em um local existem 4 possíveis comportamentos que podem ocorrer. Dependendo do estado, podemos ter quatro tipos de rotações diferentes. Consideremos a situação da figura 8.2:

Figura 8.2: Difusão de partículas



Temos então as soluções possíveis na figura 8.3:

Figura 8.3: Rotações possíveis



(a) p_0 (b) p_1 (c) p_2 (d) p_3

Podemos observar na primeira figura o efeito de transmissão, ou seja, as partículas continuam no seu próprio caminho, na segunda figura ocorre colisão e rotação horária das direções, na terceira é a reflexão e na ultima é a rotação anti-horária. Para que a difusão seja homogênea, temos que $p_1 = p_3 = p$.

Como μ_l é 1 com probabilidade p_l, temos que o valor médio de μ_l é p_l. Com isso em mente, podemos dizer que, dados N partículas, temos a seguinte regra para difusão:

$$N_i(\overrightarrow{r} + \lambda\overrightarrow{c_i}, t+\tau) - N_i(\overrightarrow{r}, t) = (p_0 - 1)N_i + p_1 N_{i+1} + p_2 N_{i+2} + p_3 N_{i+3} \quad (8.3)$$

Com essa equação conseguimos já determinar um sistema necessário para que simulemos uma difusão em um meio homogêneo:

$$\begin{aligned}
N_1(x+\lambda, y, t+\tau) &= p_0 N_1(x,y) + pN_2(x,y) + p_2 N_3(x,y) + pN_4(x,y) \\
N_2(x, y+\lambda, t+\tau) &= p_0 N_2(x,y) + pN_3(x,y) + p_2 N_4(x,y) + pN_1(x,y) \\
N_3(x-\lambda, y, t+\tau) &= p_0 N_3(x,y) + pN_4(x,y) + p_2 N_1(x,y) + pN_2(x,y) \\
N_2(x, y-\lambda, t+\tau) &= p_0 N_4(x,y) + pN_3(x,y) + p_2 N_4(x,y) + pN_1(x,y)
\end{aligned} \quad (8.4)$$

Das equações 8.4, podemos então determinar a matriz do *TLM* que precisamos para nosso modelo:

$$\begin{pmatrix} p_0 & p & p_2 & p \\ p & p_0 & p & p_2 \\ p_2 & p & p_0 & p \\ p & p_2 & p & p_0 \end{pmatrix} \quad (8.5)$$

Observando essa matriz podemos perceber que quando todos os p_l são iguais é só dividir a

8.2.2 Prova matemática

A prova matemática é similar a executada para a propagação de ondas na seção anterior e pode ser encontrada em [4]. Para este trabalho é suficiente admitir que, o modelo só é verdadeiro quando:

$$D = \frac{\lambda^2}{\tau}\left(\frac{1}{4(p+p_2)} - \frac{1}{4}\right) = \frac{\lambda^2}{\tau}\left(\frac{p+p_0}{4[1-(p+p_0)]}\right) \tag{8.6}$$

É importante perceber que quando todas as probabilidades $p_l = 0.25$, temos, de 8.6, que o modelo é válido quando:

$$D = \frac{\lambda^2}{4\tau} \tag{8.7}$$

Capítulo 9

Materiais e métodos

9.1 Introdução

Um dos aspectos definidos logo no inicio desse projeto foi que não utilizaríamos nenhum tipo de programa que não fosse livre e que não pudéssemos conseguir gratuitamente. Outro regra que norteou o desenvolvimento foi que todo código gerado neste trabalho deveria ser tão portável quanto possível. Para esse fim, algumas considerações quanto aos programas utilizados neste trabalho foram feitas.

Segue neste capítulo uma descrição do material usado e a justificativa de suas escolhas.

9.2 Computador

O computador utilizado foi o computador pessoal do próprio autor, um *pentium 4* com $1.8GHz$ de velocidade com 256 megabytes de memória *RAM*. Instalamos neste micro os programas necessários para o desenvolvimento deste trabalho.

9.3 Linguagem de programação

A linguagem de programação adotada para o projeto foi o *C++* pois existem disponíveis para os sistemas mais conhecidos compiladores desta linguagem. Outro razão pela qual escolhemos *C++* é a existência de um padrão bem definido da linguagem que permite que, caso respeitado, a portabilidade do código é garantida. Neste trabalho foi utilizado o compilador *GNU g++ (gcc) 3.2.2* disponível gratuitamente na *internet* e parte integrante do *Slackware Linux - 9.0.*

Um outro fator que leva a escolha do *C++* é o suporte a orientação objeto que faz com que o código seja altamente modularizado (possibilitando a execução de programas com funções separadas) e a reutilização do código. É intenção que o código desenvolvido se torne uma ferramenta modular independente de maneira que, futuramente, o próprio autor e outras pessoas possam reutilizá-lo para outros fins sem grandes dificuldades.

Como *apoio*, foram utilizados os programas *Scilab* e *Gnuplot*.O *Scilab* tem funcionamento similar ao *Matlab* com a vantagem de ser gratuito e ser distribuído livremente na internet.

9.4 Saídas gráficas

Como saída gráfica determinamos que os programas, deveriam, caso requisitados escrever em um arquivo do tipo *.ppm* (colorido) ou *.pgm* (preto e branco) onde pudessem ser observados os efeitos da propagação sonora. Para visualizações bidimensionais, os programas geram um arquivo com os pontos e seus respectivos valores em um arquivo. Para saídas em uma única dimensão, é gerado um arquivo com pontos de maneira que depois seja transformado em imagem através do programa gnuplot, que gera imagens em diferentes formatos sendo ideal para nosso uso.

Saídas gráficas tridimensionais foi desenvolvido um programa utilizando tecnologia OpenGL.

9.5 Sistema operacional

Para ser feito o desenvolvimento utilizamos um sistema linux, principalmente o *Slackware Linux*, mas também o *Red Hat Linux* que podem ser encontrados gratuitamente na internet também.

O sistema linux tem diversas vantagens para o desenvolvedor no caso deste projeto. Além de oferecer uma grande gama de programas *livres* que são úteis para quem está programando como o compilador *gcc* e o debbuger *gdb*, o linux oferece uma maior estabilidade e segurança no desenvolvimento do código.

Por experiências anteriores com C++, ponteiros *soltos* são diagnosticados no linux com uma maior facilidade do que no windows, mesmo que usando o *gcc-djgpp*, o que ajuda a não ocorrer desastres como o programa correr com sucesso aleatório no sistema, e não correr em um outro.

Outra vantagem é que, para o programador, o sistema traz por padrão uma série de recursos visuais e de programas que proporcionam um conforto e uma eficiência maior.

9.6 Ferramentas de offi ce

Para escrever relatórios utilizamos o LaTeX 2_ε. Isso por que o LaTeX proporciona um maior conforto ao usuário uma vez que tira dele a responsabilidade sobre a formatação do texto. Dentro do LaTeX encontram-se programadas diversas regras de tipologia, de maneira que o usuário sem grandes dificuldades possa desenvolver relatórios com formatação profissional.

Outra grande vantagem é, que devido a ser feito em código, o LaTeX tem duração muito maior do que programas de office comerciais cujos arquivos só duram até o lançamento da próxima versão do programa, quando o usuário se vê obrigado a atualizar seu sistema.

Com LaTeX, o usuário garante seu texto por mais tempo, e, através da geração de documentos em *pdf* com ferramentas gratuitas, garante a distribuição do seu texto.

As apresentações também foram feitas em LaTeX 2_ε.

9.7 Custos

Os custos do projeto se limitam somente as horas trabalhadas do autor e do orientador e o custo em energia e equipamentos (computadores, locais, iluminação, ...). Não houve nenhum custo em *software* e o objetivo de somente se utilizar *software* livre foi atingido.

9.8 Editor de texto

O editor de texto escolhido para a edição dos programas, relatórios e outros *scripts* executados para este projeto foi o *Vim-Vi improved.* Esse editor de texto se mostrou ser, dentro os editores livres (e também de entre proprietários) o que mais apresentava vantagens e conforto para a execução de textos puros.

9.9 Literatura

Para levantamento de literatura utilizamos as bibliotecas da USP (*IFUSP-Instituto de Física da USP, IME-Instituto de Matemática da USP* e *EPUSP - Escola Politécnica da USP*) e também a a internet. Todos os textos considerados relevantes estão relacionados na bibliografia deste documento.

Capítulo 10

Resultados: Modelos físicos

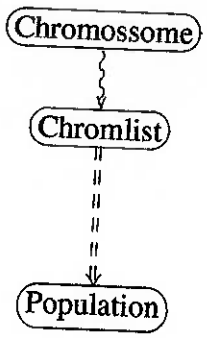
Neste capítulo descrevemos os programas dos modelos físicos e alguns exemplos de execução, resultado final do trabalho. Todos os programas aqui apresentados foram escritos em C++ e compilados, sendo que seu funcionamento foi amplamente testado. Cópias completas do código são encontradas nos apêndices. Neste capítulo apresentamos os programas referentes a modelagem de sistemas de propagação de onda e de difusão e a biblioteca de otimização.

10.1 Algoritmos genéticos

10.1.1 Descrição geral

Para implementarmos o programa em C++, existem uma série de dificuldades que precisam ser contornadas. Uma delas, é que não sabemos de antemão qual o tamanho e o número de cromossomos que serão criados. Isso gera um problema devido a necessidade de alocação de memória principalmente se o tamanho do vetor contendo os genes for muito grande. Outro questão, é o número de filhos que serão criados que é, apesar de estatisticamente controlado, aleatório. Tudo isso nos leva a implementação do algoritmo através de uma lista ligada, onde cada nódo da lista é constituído por um cromossomo com suas próprias características.

Uma segunda característica desde programa é a excessiva quantidade de comentários que possibilitaria a alguém, sabendo do que se trata o assunto, alterar da maneira que achar melhor o código para que se adapte melhor ao problema sendo abordado. Implementamos o programa baseado em 3 estruturas de dados, o Chromossome, o Chromlist e a Population. Dessa maneira reproduzimos o modelo proposto por [14] adotando a seguinte estrutura:



Para manipular a população criamos uma classe chamada *Optimizer* cuja única função é dar acesso simples a rotina e implementar um meio de parada por gerações sem evolução.Essas classes foram todas implementadas em forma de *templates* gerando assim uma ferramenta de otimização. O código completo se encontra nos apêndices e não será abordado aqui em detalhe por não fazer parte do escopo do projeto.

10.1.2 Considerações

O programa que implementa algoritmos genéticos foi feito de maneira que implementase uma ferramenta de otimização e pudesse ser útil não só neste projeto como em outros projetos atuais e futuros. Como exemplo ele foi usado na procura de planos de inspeção ótimos, um outro projeto do orientador. Graças a estrutura em templates e a capacidade de tratar grandes cromossomos (grande número de variáveis de projeto), a utilidade da ferramenta pode ser ampliada. Outro detalhe importante a salientar é que o algoritmo de otimização tem funcionamento independente do sistema à otimizar.

10.2 Descrição - autômatos celulares unidimensionais

Para geração dos exemplos unidimensionais um pequeno programa em *C++* foi escrito. Esse programa está na forma estruturada não sendo portanto de grande utilidade, a não ser meramente ilustrativa. No entanto, este programa serviu como verificação dos dados levantados na bibliografia e foi extremamente útil para o entendimento do funcionamento de autômatos celulares

10.3 Descrição - propagação de ondas bidimensional

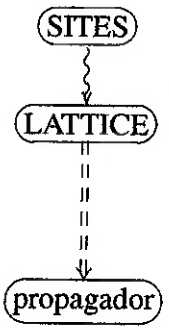
10.3.1 Descrição geral

Implementamos um programa que simula a propagação de ondas em um meio bidimensional retangular de bordas reflexivas. Como requisito de projeto tínhamos a portabilidade do código e a modularização do programa. Como resultado, obtivemos um código em C++, altamente portável, e extremamente modularizado. Isso foi extremamente importante em outra parte do projeto, onde o utilizamos o modelo de propagação praticamente como uma caixa preta.

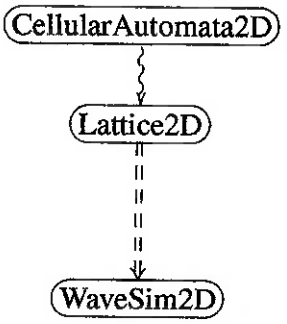
O programa consiste, então, em sua base, de três estruturas:

1. *sites ou locais*: cada ponto da malha a ser modelada;
2. *lattice ou malha*: o conjunto de pontos a ser modelado;
3. *propagador*: implementa as regras locais de iteração.

Temos então a seguinte estrutura:



Que no nosso caso é representado pelos arquivos:



10.3.2 Sites: *CellularAutomata2D*

O primeiro passo foi a programação de um *site* que tivesse propriedades definidas para que se pudesse propagar a onda por malhas desses objetos. Foi criado então a primeira estrutura de dados que constitui dois arquivos:

1. *CellularAutomata2D.h*;
2. *CellularAutomata2D.cpp*;

A classe *CellularAutomata2D* implementa as ferramentas necessárias para manipular os *sites* e guardar todas as informações úteis sobre eles.

Ela é constituída de:

- membros privados de ponto flutuante que guardam informações de cada site em um determinado estágio:
  - *double F1;*
  - *double F2;*
  - *double F3;*
  - *double F4;*
  - *double F0;*
- membros privados de determinação dos valores *F1, F2, ...* somente para uso interno da classe:
  - *void setF0(double Value);*
  - *void setF1(double Value);*
  - *void setF2(double Value);*
  - *void setF3(double Value);*
  - *void setF4(double Value);*
- um membro privado *IsReflective* que indica se esse local é reflexivo ou não:
  - *bool IsReflective;*
- um membro público *F* que guarda um valor para ser desenhado na saída gráfica:
  - *double F;*

- métodos públicos para somar valores em *F1, F2, ...*, usado no momento da simulação:
  - *void plusF0(double Value);*
  - *void plusF1(double Value);*
  - *void plusF2(double Value);*
  - *void plusF3(double Value);*
  - *void plusF4(double Value);*
- métodos públicos para retornar os valores guardados em *F1, F2, ...*:
  - *double returnF0();*
  - *double returnF1();*
  - *double returnF2();*
  - *double returnF3();*
  - *double returnF4();*
- um método público para imprimir na tela os valores importantes:
  - *void print();*
- um método público que inicialize os dados *F1, F2, ...* com valores desejados:
  - *void init(double f0, double f1, double f2, double f3, double f4);*
- um método público que coloca zero nos valores de *F1, F2, ...*:
  - *void initZero();*
- um método público que calcula o valor de *F* para ser plotado:
  - *void calcF();*
- um método público que seta *IsReflective* como verdadeiro ou falso:
  - *void setIsReflective(bool Reflective);*
- um método público que retorna se *IsReflective* é verdadeiro ou falso:
  - *void returnIsReflective();*

- um construtor e um destrutor que alocam e liberam memória para o local:
  - *CellularAutomata2D();*
  - *~CellularAutomata2D();*

Considerações

A idéia de programar cada *site* de uma maneira independente vem da própria teoria do autômato celular. Dessa maneira temos um grupo dessas *partículas* que vão interagir uma com as outras. Além disso, dessa maneira modularizamos a programação de maneira que facilitamos os testes específicos para os *sites*.

10.3.3 Lattice: *Lattice2D*

A segunda coisa a fazer foi a programação de uma malha de *CellularAutomata2D*s e isso foi feito em dois arquivos:

1. *Lattice2D.h*;
2. *Lattice2D.cpp*;

A classe *Lattice2D* representa uma malha retangular de tamanho variável de *CellularAutomata2D*s. Ela é constituída de:

- membros privados que guardam o tamanho da malha:
  - *int NumberOfLines;*
  - *int NumberOfColumns;*
- um vetor de vetores de *CellularAutomata2D* privado que armazena a malha:
  - *CellularAutomata2D ***CellBox;*
- um método privado para inicialização da malha usado no construtor:
  - *void createCellBox();*
- um método privado para a destruição da malha usada no destrutor:
  - *void releaseCellBox();*
- um método público que zera todos os *CellularAutomata2D*s

- *void inputZero();*

- um método público que calcula o valor de F de toda a malha:
  - *void calcF();*

- um método público que adiciona valores a um determinado *CellularAutomata2D*:
  - *void plusCell(double f0, double f1, double f2, double f3, double f4, int Line, int Column);*

- métodos públicos para retornar os valores de *F1, F2, ...* de um determinado *CellularAutomata2D*
  - *double returnCellF0(int Line, int Column);*
  - *double returnCellF1(int Line, int Column);*
  - *double returnCellF2(int Line, int Column);*
  - *double returnCellF3(int Line, int Column);*
  - *double returnCellF4(int Line, int Column);*

- um método público para inicializar um *CellularAutomata2D*
  - *void initCell(double f0, double f1, double f2, double f3, double f4, int Line, int Column);*

- um método público para fazer um determinado local ser reflexivo *setIsReflective*:
  - *void setIsReflective(int Line, int Column, bool setIsReflective);*

- um método público que retorna se um determinado local é reflexivo *returnIsReflective*:
  - *bool returnIsReflective(int Line, int Column);*

- um método público prepara os dados para plotar a animação com OpenGL *printRealDataToFile*:
  - *void printRealDataToFile(char *FileName);*

- um método público que imprime a imagem em um arquivo fornecido *printToFile*:

- *bool printToFile(char *FileName);*

- um método que imprime o arquivo gráfico de saída *figura.ppm*. Esse método recebe uma precisão para qual varia as 10 primeiras cores:

  - *void printToFile(double Accurate);*

- um construtor padrão que imprime a mensagem de erro de poucos parâmetros:

  - *Lattice2D();*

- um construtor e um destrutor para inicializar e destruir a malha com os devidos atributos:

  - *Lattice2D(int NumLin, int NumCol);*
  - *~Lattice2D();*

Considerações

Até aqui foi programado somente o meio no qual iremos ter iterações. Dessa maneira isolamos um problema do outro, podendo então nos certificar do funcionamento desta parte do programa.

Aqui está formado também o primeiro módulo do programa, uma vez que outro programador, com pouco estudo, conseguiria manipular esse meio pra outro fim.

10.3.4 Propagador: *WaveSim2D*

O propagador é uma classe que manipula a grade de maneira que possamos ter a propagação de onda. A partir dessa classe, várias outras foram feitas mudando simplesmente algumas características. Por essa razão, essa classe não tem nenhum membro privado, mas sim protegido (*protected*). O propagador, *WaveSim2D*, é composto de dois arquivos:

1. *WaveSim2D.h*
2. *WaveSim2D.cpp*

A classe *WaveSim2D* é constituída de:

- membros protegidos que guardam o tamanho da malha a ser simulada:

  - *int NumberOfLines;*

- *int NumberOfColumns;*

- membros protegidos que guardam coordenadas de excitação:
  - *int InputLine;*
  - *int InputColumn;*

- um membro protegido que decide se o impulso de excitação deve ser repetido todo passo (*true*) ou não (*false*):
  - *Frequency;*

- um membro protegido do tipo *Lattice2D* que guarda a malha;
  - *Lattice2D *Lattice;*

- membros protegidos que implementam a matriz de transição, reflexiva para fronteiras e livre para pontos que não são de fronteira:
  - *virtual void leftBorder(Lattice2D *Result);*
  - *virtual void rightBorder(Lattice2D *Result);*
  - *virtual void topBorder(Lattice2D *Result);*
  - *virtual void bottomBorder(Lattice2D *Result);*
  - *virtual void middle(Lattice2D *Result);*
  - *virtual void corners(Lattice2D *Result);*

- um método protegido para inicializar a excitação que pode ser chamado uma ou várias vezes, dependendo se a entrada é ou não em freqüência:
  - *virtual void setInput();*

- um método protegido que devolve os valores de *F1, F2, ...*:
  - *virtual void getValues(double &f0, double &f1, double &f2, double &f3, double &f4, int Line, int Column);*

- um método público que simula o sistema por um dado número de passos:
  - *virtual void generate(int NumberOfSteps);*

- um método público que simula o sistema por um dado número de passos e coloca os arquivos para gerar animações em OpenGL, *void generate(int NumberOfSteps, char *DataFileBaseName)*:
  - *virtual void generate(int NumberOfSteps, char *DataFileBaseName);*
- um método público que simula o sistema e cria em uma animação, *void generate(int NumberOfSteps, double Accurate, char *AnimationName)*:
  - *virtual void generate(int NumberOfSteps, double Accurate ,char *AnimationName);*
- um método público
- um método público para imprimir valores numéricos na tela, útil para correção e testes:
  - *virtual void print();*
- um método público que imprime no arquivo *figura.ppm* a saída gráfica da simulação:
  - *virtual void printToFile(double Accurate);*
- um método público que imprime para um arquivo *printToFile(double Accurate, char *FileName)*:
  - *virtual void printToFile(double Accurate, char *FileName);*
- um método público que imprime para um arquivo para gerar animações para OpenGL *printRealDataToFile(char *FileName)*:
  - *virtual void printRealDataToFile(char *FileName);*
- um método público retorna uma linha de amostra *returnSampleLine(int SampleFromLine)*:
  - *virtual double* returnSampleLine(int SampleFromLine);*
- construtores e destrutores que inicializam e destroem o propagador:
  - *WaveSim2D() erro: não determinou o tamanho da malha;*

- *WaveSim2D(int LatticeLines, int LatticeColumns, int InputLin, int InputCol, bool type);*
- *~WaveSim2D();*

Considerações

Esta classe é a base de tudo, aqui podemos dizer que a programação da idéia do autômato celular acabou. Alguns métodos são colocados na forma *virtual* do *C++* para facilitar ainda mais um outro programador que deseje escrever uma classe derivada desta para atingir seus próprios fins. O próprio autor usou dessa técnica para poder analisar diversos resultados diferentes.

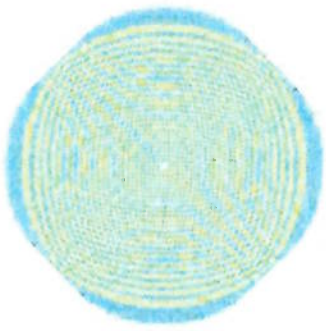
A excitação no caso deste programa é sempre feita em 4 pontos em torno do ponto dado de maneira a formar o efeito da pedra que entra numa poça d'água.

Outra coisa que vale a pena salientar é a completa ausência de membros privados. A dupla, membros protegidos e funções virtuais, fazem com que no caso de uma herança pública o reaproveitamento de código já testado seja muito maior.

Alguns resultados

Com este programa pronto pude fazer os primeiros teste e observar as primeiras propagações de onda com algum significado. A figura 10.1 apresenta uma malha com duzentas linhas por duzentas colunas, uma excitação em forma de Delta de Dirac na linha 100 coluna 100 simulado por 100 passos.

Figura 10.1: Onda bidimensional



No caso aqui já podemos observar as frentes de onda formadas. As cores, padrão em todo documento, dizem que quanto mais azul mais positiva é o valor da frente, e

quanto mais amarelo mais negativo é o vetor da frente da onda. Quando o valor vale zero é a parte branca da figura (10.2).

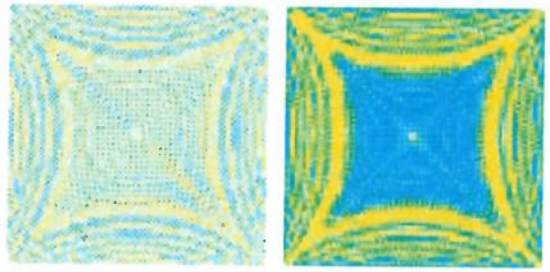
Figura 10.2: Códigos de cores das frentes de onda

Podemos com esse modelo verificar também a ocorrência de reflexão, aqui no caso com inversão de fase (figura 10.3).

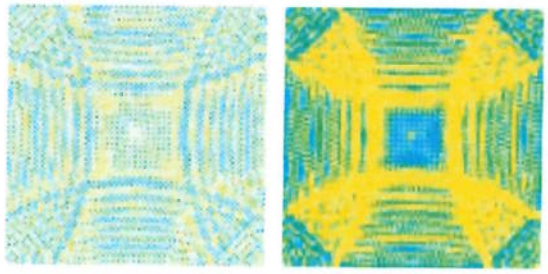
Figura 10.3: Onda refletida e com interferência da reflexão

Podemos então gerar uma série de situações diferentes para observar o fenômeno(todos com grades de 100 linhas por 100 colunas, sendo que as figuras *b, d* e *f* tem entradas em freqüência e todas tem excitação central)(figura 10.4):

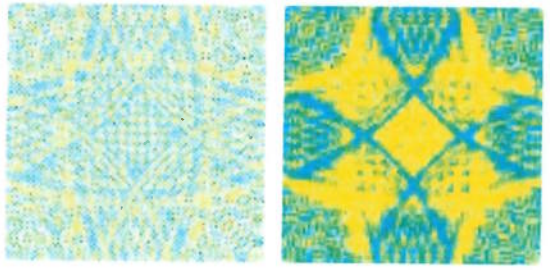
Figura 10.4: Casos de reflexão e interferência



(a) 100 passos (b) 100 passos



(c) 120 passos (d) 120 passos



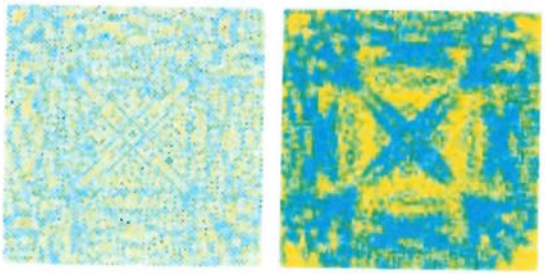
(e) 180 passos (f) 180 passos

Percebemos nas imagens da figura 10.4 que a propagação se encaminha para um estado de novo equilíbrio que podemos chamar de regime permanente, algo como o que já pode ser observado nas imagens da figura 10.5:

10.3.5 Índices de refração: *WaveSim2D2Enviroments*

Conforme mostrado anteriormente, pode-se definir para cada *CellularAutomata2D* um índice de refração diferente manipulando o quinto atributo, *F0*. Para tal foi escrito uma classe derivada de *WaveSim2D* chamada WaveSim2D2Enviroments. Ela consiste

Figura 10.5: Casos de reflexão em regime quase permanente



(a) 400 passos (b) 400 passos

de dois arquivos:

1. WaveSim2D2Enviroments.h
2. WaveSim2D2Enviroments.cpp

Esta classe consiste na mesma que WaveSim2D, com as seguintes diferenças:

- dois membros protegidos que guardam dois índices de refração diferentes e maiores ou iguais a 1:
  - *double RefractionIndex1;*
  - *double RefractionIndex2;*
- um membro protegido que guarda a linha de fronteira entre os dois meios:
  - *int Frontier;*
- um método protegido novo e uma sobrecarga do método *void middle* para fazer o novo cálculo:
  - *void middle(Lattice2D *Result);*
  - *void refractionCalc(double &F0, double &F1, double &F2, double &F3, double &F4, double RefractionIndex, int PositionX, int PositionY);*

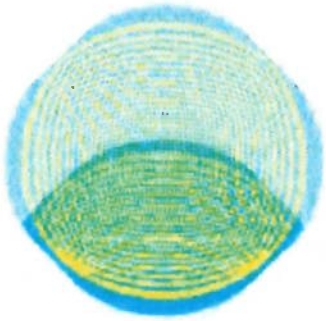
Considerações

Essa classe por ser derivada, ou seja, foi usada a técnica da herança, conserva todas as propriedades da classe mãe, no caso aqui *WaveSim2D*. Basta selecionar uma linha para mudança de meio, definir os dois índices de refração e simular o sistema como já feito anteriormente. O índice de refração pode servir futuramente como *Jacobiano* para transformação de superfícies irregulares que se queira simular para um meio plano como o programado.

Alguns resultados

Uma idéia do efeito de refração pode ser visto na figura 10.6. Neste caso temos uma malha de 200 linhas por 200 colunas, simulados 100 passos com transição de meio na linha 130. O índice de refração do meio superior é 1 e o do meio inferior é 1.5 com relação a velocidade de propagação livre da malha proposta.

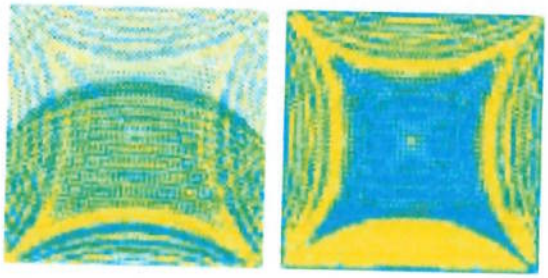
Figura 10.6: Onda bidimensional-refração



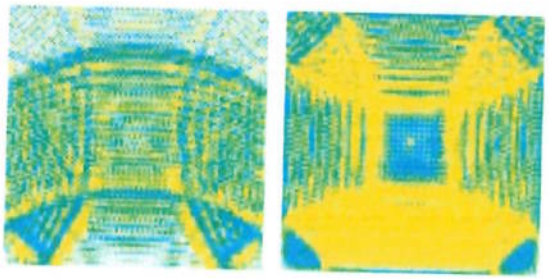
Pode-se perceber na linha de transição do meio a mudança de velocidade da onda e sua parte refletida. Na figura 10.7 podemos perceber a evolução de uma malha de 100 linhas por 100 colunas com índice de refração 1 no meio superior e 1.5 no meio inferior. A linha de transição foi determinada na linha 75. Mais uma vez, a coluna da direita são de imagens com entrada em freqüência.

Podemos observar nesta imagem que mesmo com dois meios o sistema consegue continuar simulando interferência e reflexão, e também a passagem contrária da reflexão entre meios. Como curiosidade a figura 10.8 traz o mesmo meio com 400 passos.

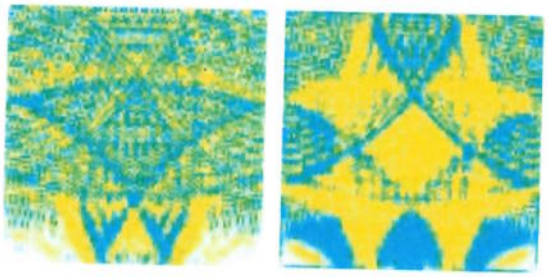
Figura 10.7: Casos de refração com reflexão e interferência



(a) 100 passos (b) 100 passos



(c) 120 passos (d) 120 passos



(e) 180 passos (f) 180 passos

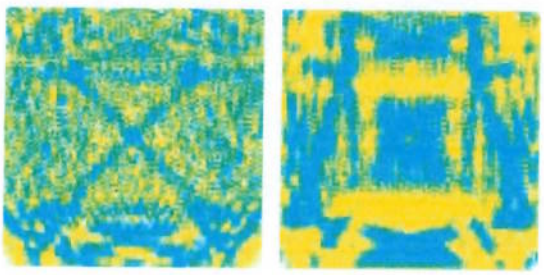
10.3.6 Amostras: *WaveSim2DSample*

Esta classe é derivada de *WaveSim2D2Enviroments*, com a diferença que ela tem a capacidade de plotar em um único arquivo uma amostra do meio simulado. Ela é constituída de dois arquivos:

1. *WaveSim2DSample.h*
2. *WaveSim2DSample.cpp*

De novo realmente esta classe tem um método público a mais que seleciona uma

Figura 10.8: Casos de refração com 400 passos



(a) 400 passos (b) 400 passos

linha e retorna um vetor com os valores da reta, que podem ser plotados mais tarde no *gnuplot*.

- double* returnSampleLine(int SampleFromLine);

Considerações

Amostrar um meio é uma atividade importante uma vez que isso simularia a presença de um sensor. O sensor nunca consegue pegar valores em todo meio. Por exemplo, pode-se dizer que estamos captando o valor lido por microfones em uma determinada posição. A importância disse vai ser salientada mais a frente.

A capacidade de se conseguir retirar amostras do modelo computacional é importante, pois são esses valores que irão ser comparados com valores reais. Conforme veremos mais adiante, essa capacidade vai ser bem útil na modelagem do detector de barreiras.

Alguns resultados

Considerando então que podemos fazer uma leitura de linhas podemos observar de maneira unidimensional resultados de nossa simulação. A figura 10.9 expõe uma amostra retirada da linha 30 de um meio de 100 linhas por 100 colunas com um único índice de refração, excitação central em forma de impulso, após 50 passos. A figura 10.10 mostra a mesma situação mas com entrada em freqüência, na linha 30 após 180 passos.

Figura 10.9: Amostra em impulso

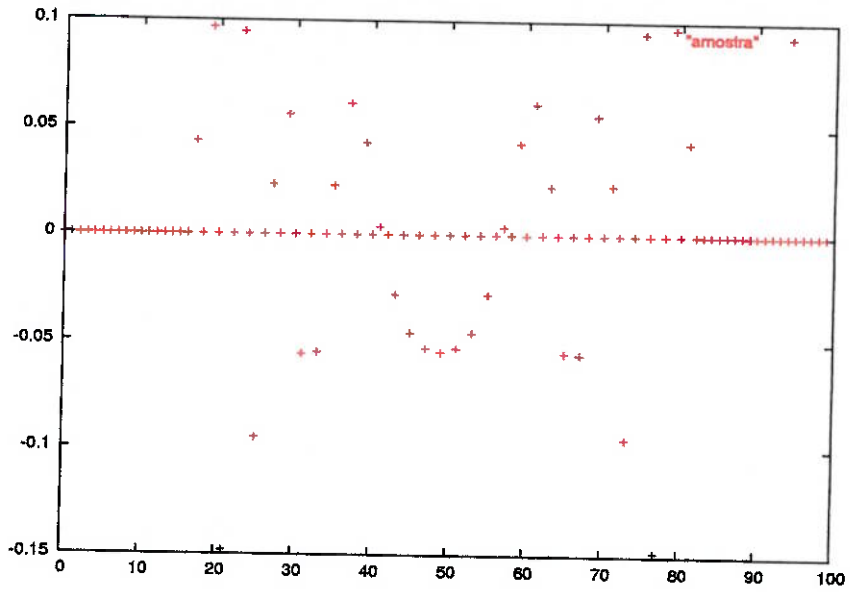
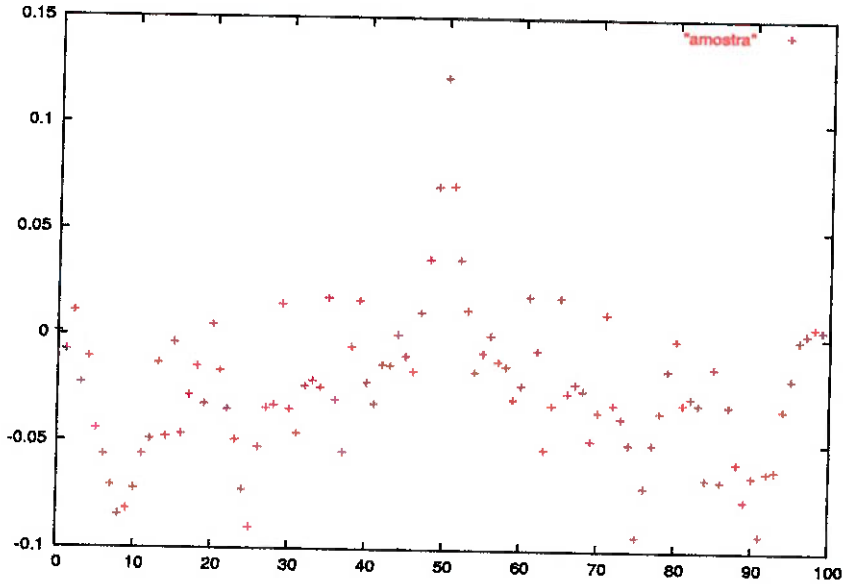


Figura 10.10: Amostra em freqüência



10.3.7 Criador de barreiras: *WaveSim2DReflect*

Esta classe foi criada para que se pudesse introduzir corpos dentro do meio. No caso, colocamos uma linha horizontal de um lado a outro no meio. Essa classe é derivada da

classe *WaveSim2DSample* e é constituída de dois arquivos:

1. *WaveSim2DReflect.h;*
2. *WaveSim2DReflect.cpp.*

Como novidade, temos um inteiro que guarda a linha a ser tornada reflexiva e uma sobrecarga do método *void middle(Lattice2D *Result)* para que seja implementada a linha.

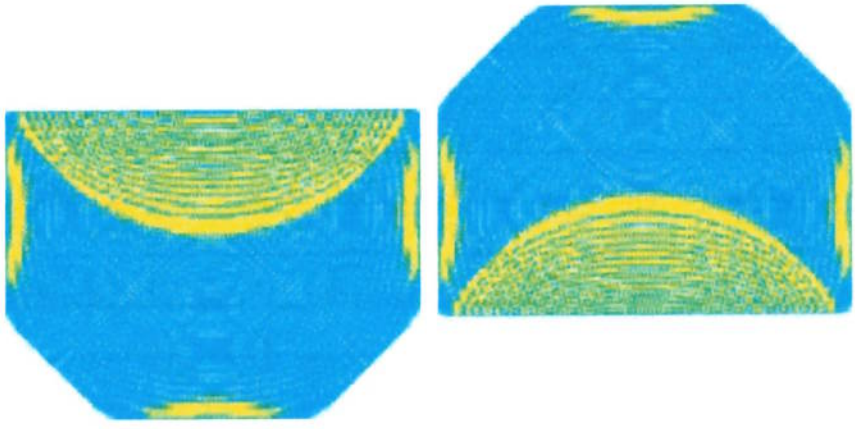
Considerações

Esta classe atesta o funcionamento da matriz de reflexão proposta, uma vez que pode ser aplicado a pontos no meio da grade com sucesso. Feita a simulação percebe-se que não há *vazamentos* no sistema.

Alguns Resultados

A figura 10.11 mostra o resultado de uma simulação com 200 linhas por 200 colunas, um único índice de refração igual a 1, com excitação central em freqüência e linhas reflexivas nas linhas 50 e 150 respectivamente por 150 passos.

Figura 10.11: Linha reflexiva



(a) linha 50 (b) linha 150

10.3.8 Refletor aleatório: *WaveSim2DRandomReflect*

O refletor de barreiras aleatório é uma classe derivada de *WaveSim2DReflect*. Ela é constituída de dois arquivos:

1. *WaveSim2DRandomReflect.h*
2. *Wavesim2DRandomReflect.cpp*

A diferença é a aplicação de uma sobrecarga do método generate e do método middle para que os mesmos pudessem receber um vetor de 0s e 1s, onde 1 significaria o ponto de uma determinada linha é reflexivo. Ele é um gerador de barreira conforme o usuário desejaria. É só mandar o vetor correto.

10.3.9 Refletor circular: *WaveSim2DCustomBarrier*

Essa classe é derivada diretamente de *WaveSim2D* e é destinada para programação de barreiras mais específicas. Até agora foi programada nela a barreira circular reflexiva aplicando o algoritmo de Bresham [17].

Essa classe é constituída de dois arquivos:

1. *WaveSim2DCustomBarrier.h*
2. *Wavesim2DCustomBarrier.cpp*

Esta classe consiste na mesma que WaveSim2D, com as seguintes diferenças:

- três membros protegidos que guardam as propriedades da barreira circular:
  - *int Radius;*
  - *int CenterLine;*
  - *int CenterColumn*
- um membro protegido que indica se existe um barreira circular (*CircularBarrierTrue*):
  - *bool CircularBarrierTrue;*
- métodos que implementam o algoritmo de Bresham para círculos:
  - *void setCircBarrier()*

- *void circlePoints(int Line, int Column);*
- *void setCircularBarrier(int CircularRadius, int LineCenter, int ColumnCenter);*

- um método novo de entrada de excitação, com excitação em linha na borda inferior:
  - *void setInput();*

Algumas considerações

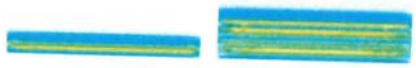
Para a barreira circular algumas mudanças foram feitas no sistema. A entrada, não mais pontual e sim uma linha no que chamamos de borda inferior e se propaga para cima.

Quando uma barreira circular é colocada, ela muda as condições internas de alguns pontos com relação a reflexividade e termina por final a gerar o que queremos.

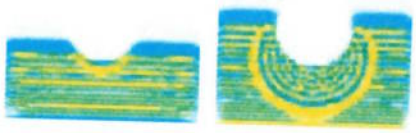
Alguns Resultados

Podemos observar nas imagens a seguir a propagação de uma onda nas condições descritas nas sessões acima em torno de uma barreira circular reflexiva (figura 10.12):

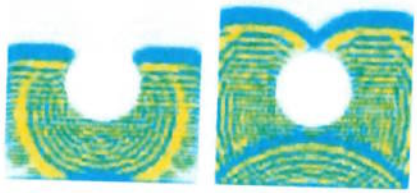
Figura 10.12: Barreira circular



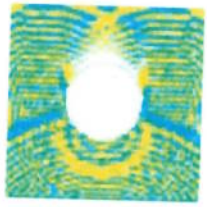
(a) 10 passos (b) 30 passos



(c) 50 passos (d) 75 passos



(e) 90 passos (f) 120 passos



(g) 150 passos

10.4 Difusão bidimensional

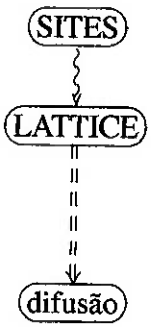
Implementamos também um programa que simula a difusão de partículas em um meio bidimensional retangular de bordas reflexivas. Como requisito mais uma vez, tínhamos

a portabilidade do código e a modularização do programa, gerando uma classe em *C++* altamente reutilizável.

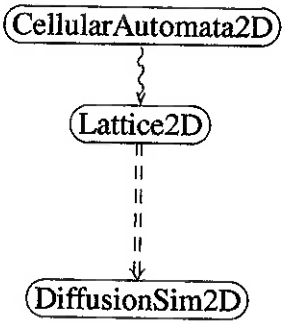
O programa utilizou a mesma estrutura de dados para a malha da propagação de ondas, mudando somente a parte mecânica conforme abordado anteriormente. O programa consiste então de três estruturas:

1. *sites ou locais*: cada ponto da malha a ser modelada;
2. *lattice ou malha*: o conjunto de pontos a ser modelado;
3. *difusão*: implementa as regras locais de iteração.

Temos então a seguinte estrutura:



Que no nosso caso é representado por:



10.4.1 Difusão: *DiffusionSim2D*

A difusão é uma classe que manipula a grade de maneira que possamos ter o efeito da difusão sobre essa grade. Ela foi feita de maneira genérica para que possamos usar no futuro a capacidade de herança do *C++* e termos outros efeitos. Por essa razão, essa classe não tem nenhum membro privado, mas sim protegido (*protected*). O difusor, como iremos chamar essa classe agora é constituído de dois arquivos

1. *DiffusionSim2D.h*

2. *DiffusionSim2D.cpp*

A classe *diffusionSim2D* é constituída de:

- membros protegidos que guardam o tamanho da malha a ser simulada:
  - *int NumberOfLines;*
  - *int NumberOfColumns;*
- membros protegidos que guardam coordenadas de excitação:
  - *int InputLine;*
  - *int InputColumn;*
- um membro protegido que decide se o impulso de excitação deve ser repetido todo passo (*true*) ou não (*false*):
  - *Frequency;*
- membros protegidos que guardam os valores das probabilidades p_l, já discutidas anteriormente:
  - *double P0;*
  - *double P1;*
  - *double P2;*
  - *double P3;*
- um membro protegido do tipo *Lattice2D* que guarda a malha;
  - *Lattice2D *Lattice;*
- membros protegidos que implementam a matriz de transição, reflexiva para fronteiras e livre para pontos que não são de fronteira:
  - *virtual void leftBorder(Lattice2D *Result);*
  - *virtual void rightBorder(Lattice2D *Result);*
  - *virtual void topBorder(Lattice2D *Result);*
  - *virtual void bottomBorder(Lattice2D *Result);*

- *virtual void middle(Lattice2D *Result);*
- *virtual void corners(Lattice2D *Result);*

- um método protegido para inicializar a excitação que pode ser chamado uma ou várias vezes, dependendo se a entrada é ou não em freqüência:
  - *virtual void setInput();*

- um método protegido que devolve os valores de *F1, F2, ...*:
  - *virtual void getValues(double &f0, double &f1, double &f2, double &f3, double &f4, int Line, int Column);*

- um método público que simula o sistema por um dado número de passos:
  - *virtual void generate(int NumberOfSteps);*

- um método público que simula o sistema por um dado número de passos e coloca os arquivos para gerar animações em OpenGL, *void generate(int NumberOfSteps, char *DataFileBaseName)*:
  - *virtual void generate(int NumberOfSteps, char *DataFileBaseName);*

- um método público que simula o sistema e cria em uma animação, *void generate(int NumberOfSteps, double Accurate, char *AnimationName)*:
  - *virtual void generate(int NumberOfSteps, double Accurate ,char *AnimationName);*

- um método público

- um método público para imprimir valores numéricos na tela, útil para correção e testes:
  - *virtual void print();*

- um método público que imprime no arquivo *figura.ppm* a saída gráfica da simulação:
  - *virtual void printToFile(double Accurate);*

- um método público que imprime para um arquivo *printToFile(double Accurate, char *FileName)*:

- *virtual void printToFile(double Accurate, char *FileName);*

- um método público que imprime para um arquivo para gerar animações para OpenGL *printRealDataToFile(char *FileName)*:

  - *virtual void printRealDataToFile(char *FileName);*

- construtores e destrutores que inicializam e destroem o propagador:

  - *DiffusionSim2D() erro: não determinou o tamanho da malha;*
  - *DiffusionSim2D(int LatticeLines, int LatticeColumns, int InputLin, int InputCol, bool type, double PD0, double PD1, double PD2, double PD3);*
  - *~DiffusionSim2D();*

Considerações

Como pode ser visto na implementação, fizemos um sistema mais genérico do que um sistema de difusão comum. Além de termos aqui a conservação da massa nas duas direções, temos também a conservação da massa e velocidade em cada direção independente. É como se tivéssemos dois processos de difusão ocorrendo simultaneamente. A excitação nesse caso é em um único ponto com concentração um 1 e segue a matriz da equação 8.5.

Com essa classe em mãos, o autor ou qualquer outro pode no futuro propor problemas mais complexos de difusão ou até mesmo utilizá-la como uma caixa preta dentro de outro programa.

Alguns resultados

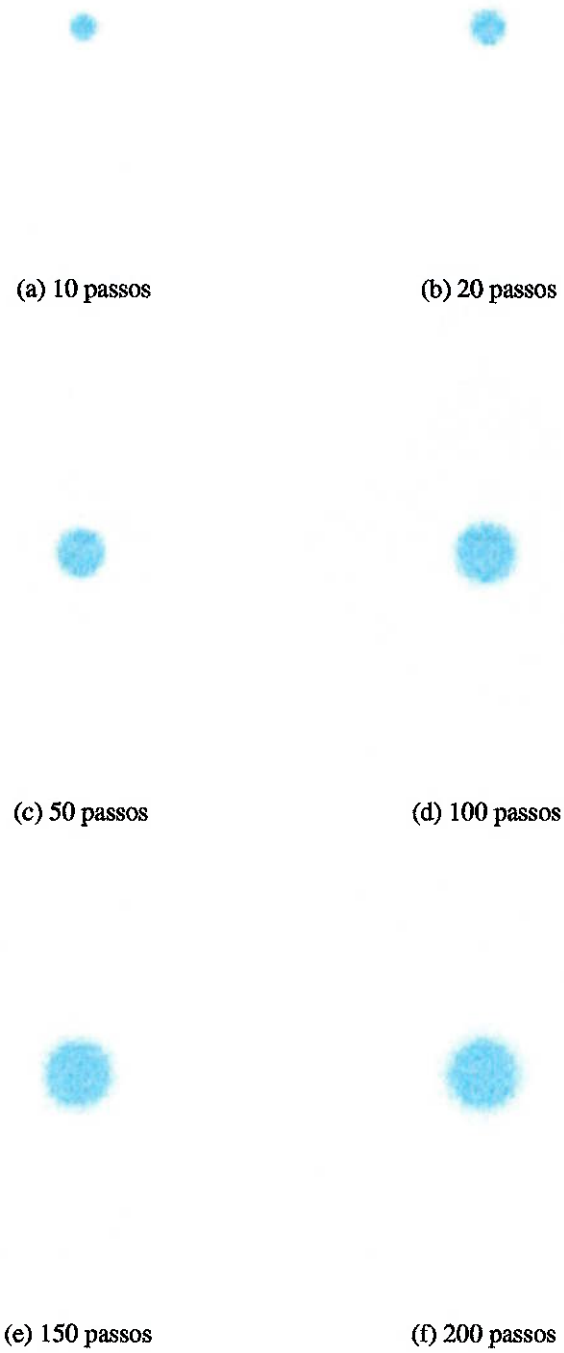
Com essa classe pronta podemos observar alguns de nossos resultados. O processo de difusão consiste em um processo de espalhamento de partículas em torno do ponto inicial.

Figura 10.13: Difusão

Na figura 10.13 podemos observar um processo de difusão, começando com uma excitação central em uma malha de 200×200 locais após 200 *passos*. Podemos observar uma situação de praticamente estabilidade.

Na seqüencia de imagens a seguir, figura 10.14, podemos observar a difusão em vários estágios até atingir o estágio da figura 10.13

Figura 10.14: Difusão no tempo



(a) 10 passos

(b) 20 passos

(c) 50 passos

(d) 100 passos

(e) 150 passos

(f) 200 passos

Capítulo 11

Resultados: animações em OpenGL

11.1 Descrição

Um dos objetivos do trabalho é o desenvolvimento de saídas gráficas tridimensionais indicando a amplitude. Não cabe aqui a descrição detalhada do programa, apresentando somente a idéia de um "plotador" de superfícies. Segue nas sessões seguintes amostras dos resultados conseguidos com a ajuda desse programa.

11.2 Propagação de ondas

11.2.1 Propagação de ondas sem barreira

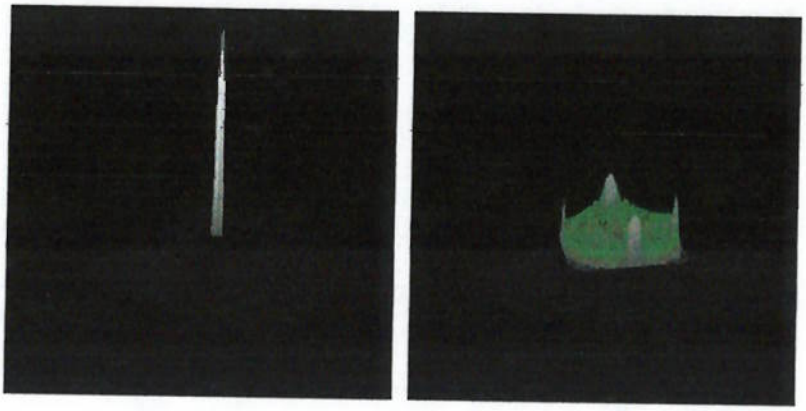
Nesta parte do trabalho apresentamos (figura 11.1) a propagação de uma onda, de fonte pontual conforme descrito anteriormente. Detalhe para a transição de meios com índices de refração 1.0 e 1.7.

11.2.2 Propagação de ondas com barreira circular e bordas reflexivas

Nesta parte do trabalho apresentamos (figura 11.2 a propagação de uma onda, de fonte em forma de linha conforme descrito anteriormente. Aqui temos em meio a propagação uma barreira circular reflexiva.

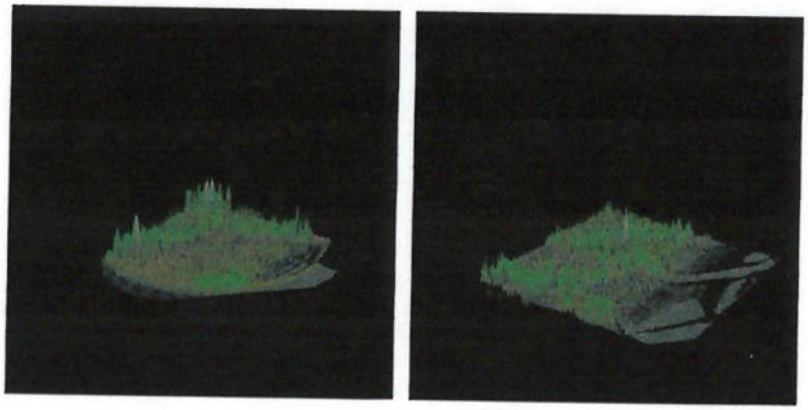
Na figura 11.3 temos a mesma situação da figura 11.2, mas agora as bordas direita esquerda e topo da propagação são absorvedoras.

Figura 11.1: Propagação de ondas em dois meios



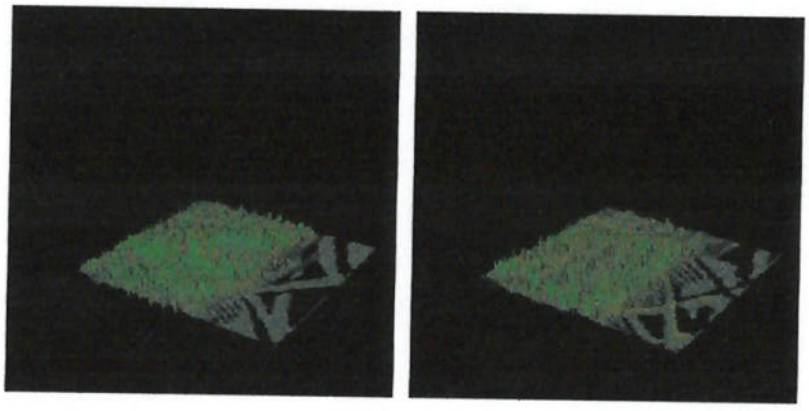
(a) 0 passos

(b) 36 passos



(c) 77 passos

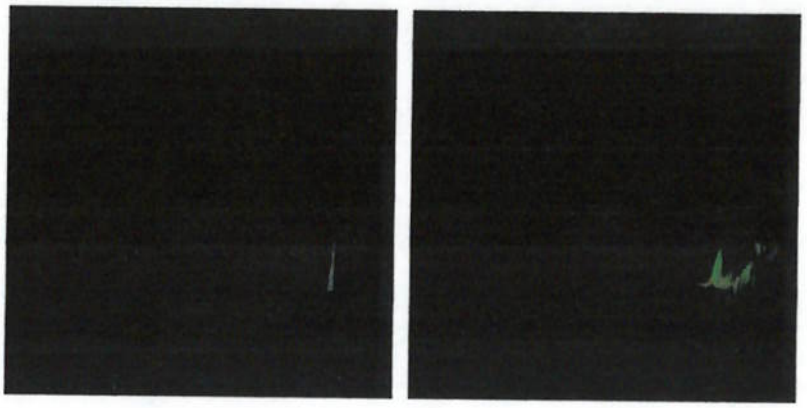
(d) 123 passos



(e) 150 passos

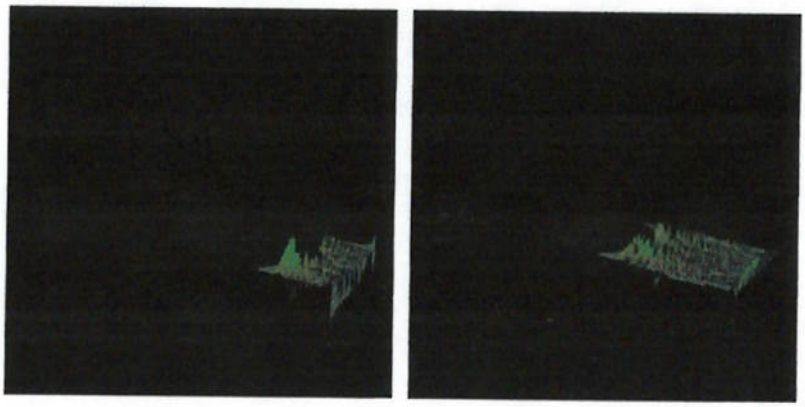
(f) 212 passos

Figura 11.2: Propagação de ondas com barreira circular



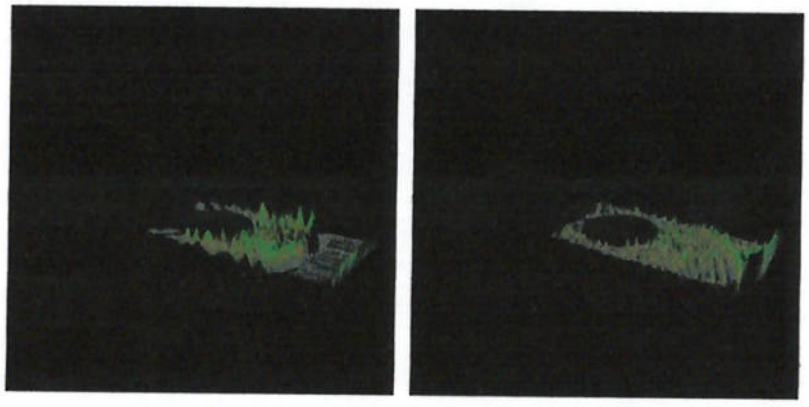
(a) 0 passos

(b) 17 passos



(c) 36 passos

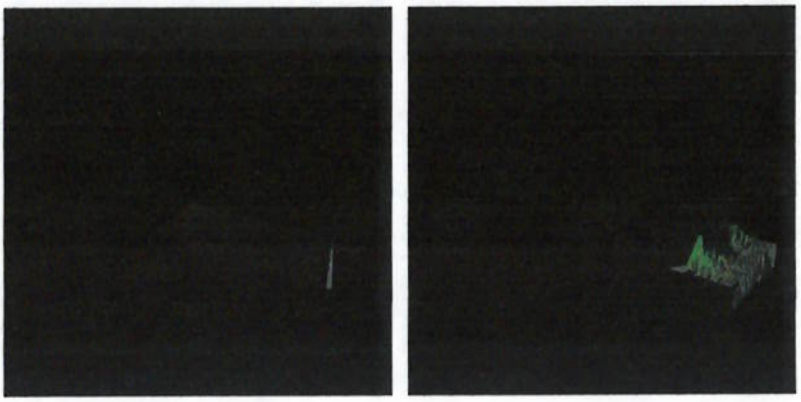
(d) 81 passos



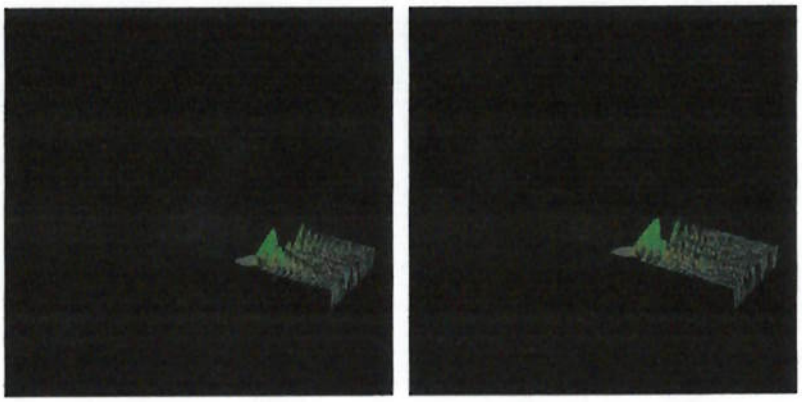
(e) 123 passos

(f) 165 passos

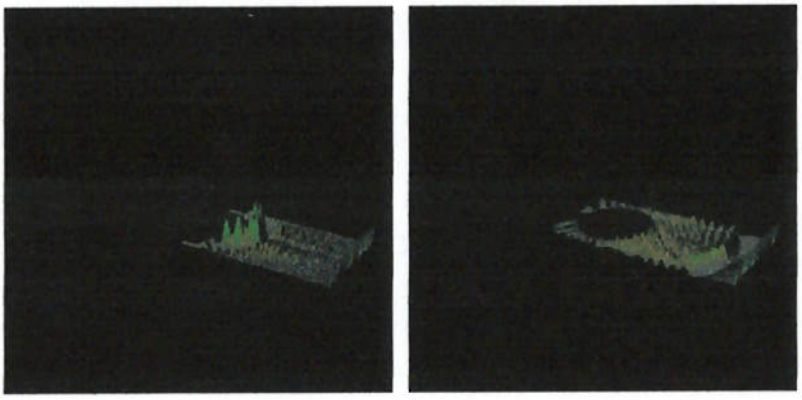
Figura 11.3: Propagação de ondas com barreira circular e bordas absorvedoras



(a) 0 passos (b) 30 passos



(c) 52 passos (d) 72 passos

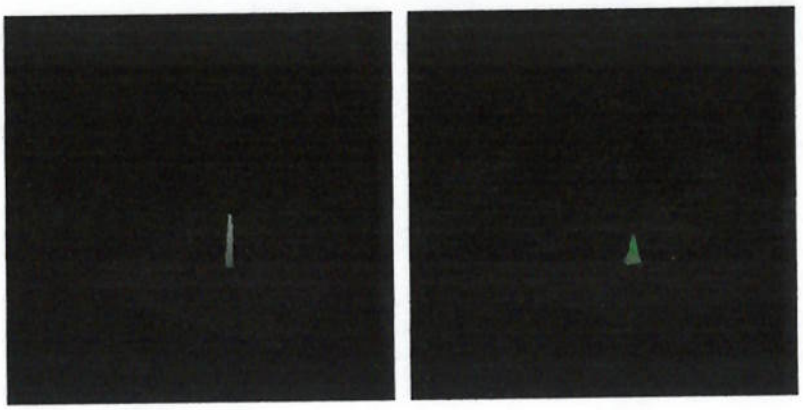


(e) 91 passos (f) 137 passos

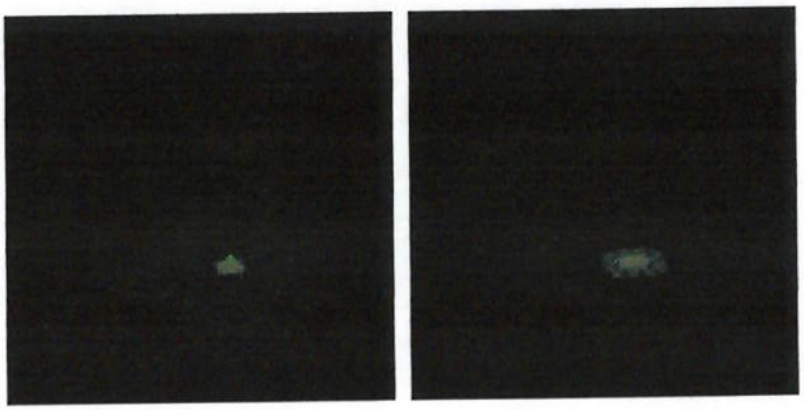
11.2.3 Difusão

O programa em OpenGL desenvolvido "plota" superfícies, portanto podemos verificar animações de processos de difusão. A figura 11.4 ilustra uma situação dessa.

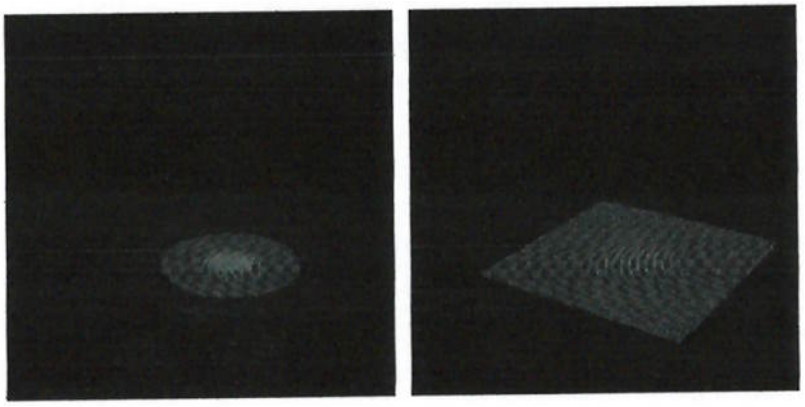
Figura 11.4: Difusão



(a) 0 passos (b) 3 passos



(c) 7 passos (d) 17 passos



(e) 73 passos (f) 380 passos

Capítulo 12

Resultados: Problemas inversos

12.1 Introdução

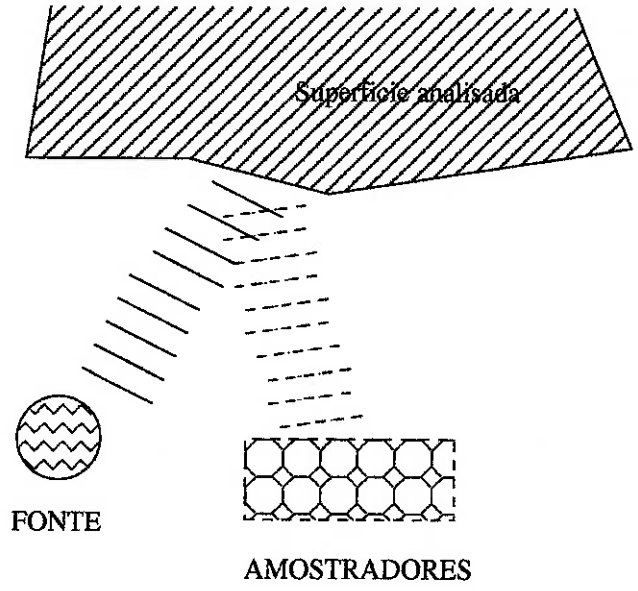
Neste capítulo faremos uma descrição do que foi executado com relação a problemas inversos, apresentando tanto programas quanto resultados gerados pelos mesmos. Abordamos aqui tanto o tratamento matemático específico de cada problema como também a apresentação de simulações reais envolvendo esses modelos.

Escolhemos para este trabalho focalizar os esforços de resolução de problemas inversos na propagação de ondas devido a sua fácil reversibilidade. Dado uma certa condição conseguimos para este caso saber qual foi a condição inicial que gerou a situação atual. Como no caso da difusão não existe essa garantia, ou seja, dado uma situação atual existem n situações anteriores corretas, é necessário que nos abstenhamos a problemas mais simples. Por esse motivo escolhemos então nos aprofundarmos mais no estudo de propagação de ondas.

12.2 Descrição do problema

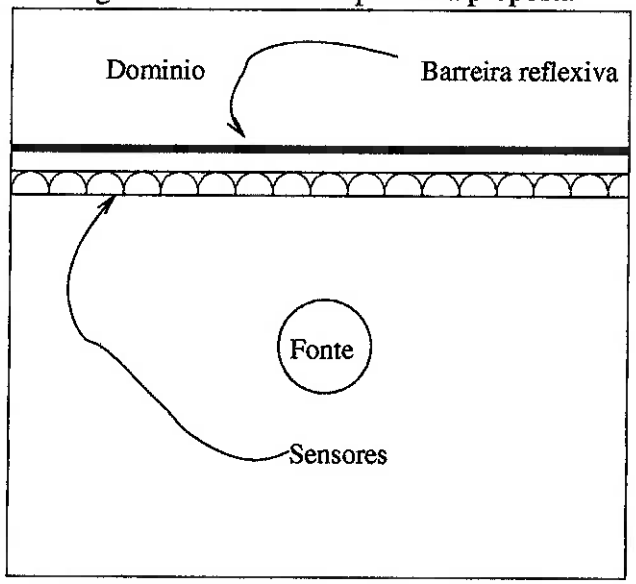
Sempre que tratamos um problema de propagação de ondas aqui consideramos um sistema fechado, onde temos algum tipo de excitação, e certas condições de contorno conhecidas e desconhecidas. Abordamos aqui um detector de barreiras, ou seja, dado um domínio, uma fonte de algum tipo de amostras queremos determinar qual foi o corpo que refletiu as frentes de onda (figura 12.1).

Figura 12.1: Detector de barreira: idéia



Dessa maneira já conseguimos enxergar aqui o espaço de modelos (quais pontos do domínio dão reflexivos) e o espaço de dados (valores dos sensores). Propomos inicialmente então uma configuração como o da figura 12.2. Temos um sistema real onde colocamos sensores, uma fonte de ondas e controlamos o ambiente de maneira a colher um conjunto de amostras, chamados neste trabalho de d_{obs}:

Figura 12.2: Detector: primeira proposta



Temos aqui então a idéia mais geral que norteia o trabalho, a procura de uma bar-

reira confinada e um espaço, onde sensores são valores colhidos em uma determinada linha e temos uma fonte de ondas, no caso pontual. A idéia é realizar experimentos com diversos parâmetros de modelo de maneira a tentar descobrir qual conjunto de parâmetros m_α se adapta a nossa amostra inicial (retirada do sistema real) com maior probabilidade.

12.3 Desenvolvimento matemático

Temos então que determinar agora, conforme discutido anteriormente qual a nossa função densidade de probabilidade a posteriori, para que maximizando esta função encontremos o resultado esperado. A função densidade de probabilidade a posteriori é dada conforme visto na equação 6.11:

$$f_{M|D}(m|d_{obs}) = \frac{f_{D|M}(d_{obs}|m)f_M(m)}{\int_M f_{D|M}(d_{obs}|m)f_M(m)dm}$$

No nosso caso, a função densidade de probabilidade a priori é simples, uma vez que consideramos que não temos informação nenhuma do sistema, ou seja, a função densidade de probabilidade a priori é o próprio estado de total ignorância. Isso resulta, de 6.11 com $f_M(m) = constante = \mu$:

$$f_{M|D}(m|d_{obs}) = \frac{f_{D|M}(d_{obs}|m)\mu}{\int_M f_{D|M}(d_{obs}|m)\mu dm} \tag{12.1}$$

Resta agora a função densidade de probabilidade $f_{D|M}(d_{obs}|m)$ que é um pouco mais complexa. Consideremos agora que cada ponto das linhas consideradas sensores é constituído de um sensor de intensidade de onda. Temos aqui erros de leitura, erros humanos e até mesmo erros de modelo, de maneira que o resultado fornecido pelos sensores não é um valor em delta de Dirac, mas sim uma função densidade de probabilidade que admitiremos, para este trabalho, normal (equação 6.3). Temos então uma função dessas para cada vez que variamos os parâmetros do modelo com o dado conseguido como média da distribuição e a variância dependendo das incertezas envolvidas. Se chamarmos os valores lidos a cada teste de d_{m_i}, teremos para cada ponto da linha a equação 12.2:

$$f(d^j_{obs}|m_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^j_{obs}-d^j_{m_i})^2}{\sigma^2}\right)} \tag{12.2}$$

Como necessitamos que todos os nossos sensores estejam corretos, temos aqui uma relação de e, o que faz com que da equação 12.2 tenhamos uma multiplicação como na

equação 12.3

$$f(d_{obs}|m_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^j_{obs}-d^j_{m_i})^2}{\sigma^2}\right)} \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^{j+1}_{obs}-d^{j+1}_{m_i})^2}{\sigma^2}\right)} \ldots \tag{12.3}$$

Voltando na equação 12.1, temos agora:

$$f_{M|D}(m|d_{obs}) = \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^j_{obs}-d^j_{m_i})^2}{\sigma^2}\right)} \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^{j+1}_{obs}-d^{j+1}_{m_i})^2}{\sigma^2}\right)} \ldots \mu}{\int_M \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^j_{obs}-d^j_{m_i})^2}{\sigma^2}\right)} \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{1}{2}\frac{(d^{j+1}_{obs}-d^{j+1}_{m_i})^2}{\sigma^2}\right)} \ldots \mu dm} \tag{12.4}$$

Já englobando todos os erros de leitura, modelos e processamento, sendo que este resultado final dá a probabilidade de o modelo estar correto. Usando um pouco de álgebra, podemos simplificar essa equação:

$$f_{M|D}(m|d_{obs}) = \frac{\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n e^{\sum_{j=0}^{n}\left(-\frac{1}{2}\frac{(d^j_{obs}-d^j_{m_i})^2}{\sigma^2}\right)} \mu}{normalizador} \tag{12.5}$$

Observando a equação 12.5, podemos perceber que para maximizar o valor da função densidade de probabilidade a posteriori temos que minimizar a relação quadrática dos resultados. Ao final temos que o resultado de maior probabilidade apresentará o menor valor da equação 12.6:

$$\sum_{j=0}^{n} [(d^j_{obs} - d^j_{m_i})^2] \tag{12.6}$$

12.4 Detector de barreiras em uma linha

Nesta parte do trabalho iremos descrever os resultados alcançados em um detector de pontos reflexivos dentro de uma linha. Para tal, usamos a biblioteca de otimização baseada em algoritmos genéticos *GAOptim* desenvolvida durante este trabalho. Para tanto, desenvolvemos dois sistemas comparadores que usam o resultado do desenvolvimento matemático anterior para maximizar o valor da função densidade de probabilidade a posteriori.

12.4.1 O sistema comparador simples

O sistema comparador simples, que procura uma linha totalmente reflexiva, foi criado através de uma classe *System* formada por dois arquivos:

1. *System.h*
2. *System.cpp*

O sistema é composto por:

- um membro privado para armazenar o número de passos a ser simulado:
  - *int Steps;*
- um membro privado para armazenar a linha de amostra:
  - *int SampleLine;*
- um vetor de double privado que guarda a amostra do caso *real*:
  - *double *Ideal;*
- membros privados para guardar propriedades do meio:
  - *int SysLatticeLines;*
  - *int SysLatticeColumns;*
  - *int SysInputLin;*
  - *int SysInputCol;*
  - *bool SysType;*
  - *int SysLineTransition;*
  - *double SysRefraction1;*
  - *double SysRefraction2;*
  - *int SysLineReflection;*
- construtores e destrutores que recebem as informações necessárias:
  - *System();*
  - *System(bool BeStill);*
  - *System(int LatticeLines, int LatticeColumns, int InputLin, int InputCol, bool type, int LineTransition, double Refraction1, double Refraction2, int LineReflection, int StepsToGo, int SampleLine);*
  - *~System();*
- um método público *eval* que compara o vetor *Ideal* com uma amostra de um sistema simulado.

Considerações

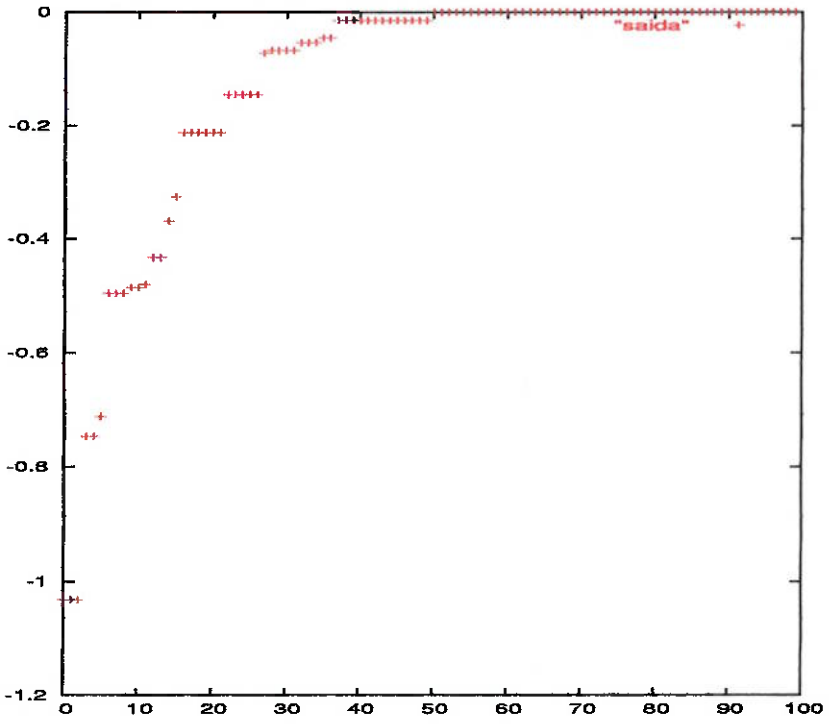
Esta classe é um sistema completo pronto para ser otimizado. Ele gera um corpo que é uma linha reflexiva e depois compara outros resultados a essa linha amostrando no mesmo ponto. No ponto ótimo, sua *eval* retornaria 0 (distribuição em delta de Dirac).

Resultados

O algoritmo foi testado em várias situações resultando em uma convergência rápida para o ponto ótimo. O problema deste algoritmo é que, para se garantir a chegada à situação ótima, deve-se garantir que todo ponto da linha reflexiva tenha influenciado a amostra. Isso pode se tornar um problema dado que a função *eval* exige simulações bidimensionais e isso é bastante custoso. Em um problema com 30 linhas por 30 colunas, excitação central em freqüência, foram calculadas por volta de 100 simulações por geração, cerca de 4800 das 268435456 (2^{28}) possibilidades foram analisadas, o que já representa um resultado bastante satisfatório.

A figura 12.3 dá uma idéia do resultado do programa.

Figura 12.3: Detector de barreira - convergência



12.4.2 O sistema comparador genérico

Após o sucesso do sistema *System*, o próximo passo lógico seria abordar corpos não inteiros dentro de uma linha como por exemplo um buraco, uma peneira ou um traço descontínuo nos extremos. Isso foi feito através da class *SystemLine*, que permitiu que vetores de entrada fossem fornecidos ao sistema. Essa classe é derivada da classe *System*.

O novo comparador é formado por dois arquivos:

1. *SystemLine.h*
2. *SystemLine.cpp*

Ela contém de diferente:

- um membro protegido para guardar uma semente de números aleatórios *Seed*:
  - *int Seed;*
- um membro protegido que guarda uma segunda linha como sensor *SampleFromLine*:
  - *int SampleFromLine;*
- um vetor protegido que guarda o valor do novo sensor Ideal2:
  - *double *Ideal2;*
- um membro que guarda a grandeza de erros a serem adicionados *ErrorSize*
  - *double ErrorSize;*
- um membro que guarda true se tem erros nas amostras *IdealGaussianErrors*
  - *bool IdealGaussianErrors;*
- um membro que guarda true se tem erros nos testes *SamplesGaussianErrors*
  - *bool SamplesGaussianErrros;*
- construtores e destrutores que recebem as informações necessárias:
  - *SystemLine();*

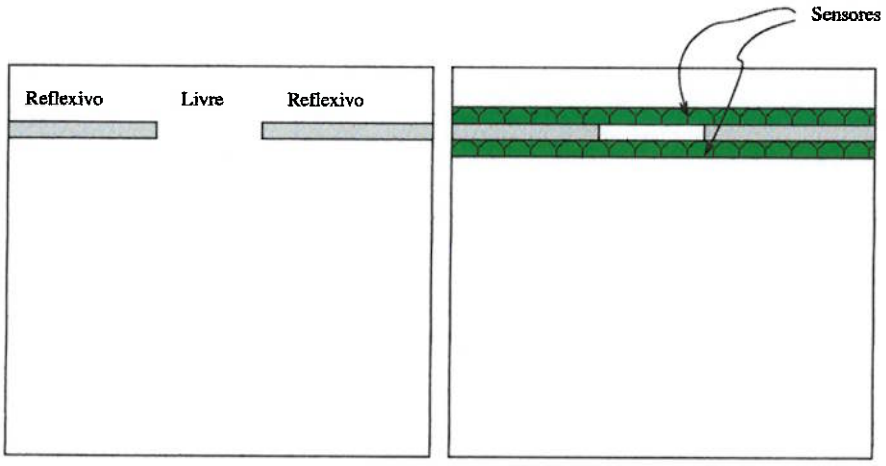
- *SystemLine(bool BeStill);*
- *SystemLine(int LatticeLines, int LatticeColumns, int InputLin, int InputCol, bool type, int LineTransition, double Refraction1, double Refraction2, int LineReflection, int StepsToGo, int SampleLine, int SampleLine2, int *Reflect, bool SamplesGaussian, bool IdealGaussian, double Error);*
- *~SystemLine();*

- métodos para ler uma semente em um arquivo e escrever uma nova semente:
  - *void getSeed();*
  - *void setSeed();*
- um método público *eval* que compara o vetor *Ideal* com uma amostra de um sistema simulado.

Considerações

Com esse sistema podemos impor na entrada uma característica desejada à barreira que queremos como real. Tendo isso em mente desenvolvemos três novos tipos de testes.

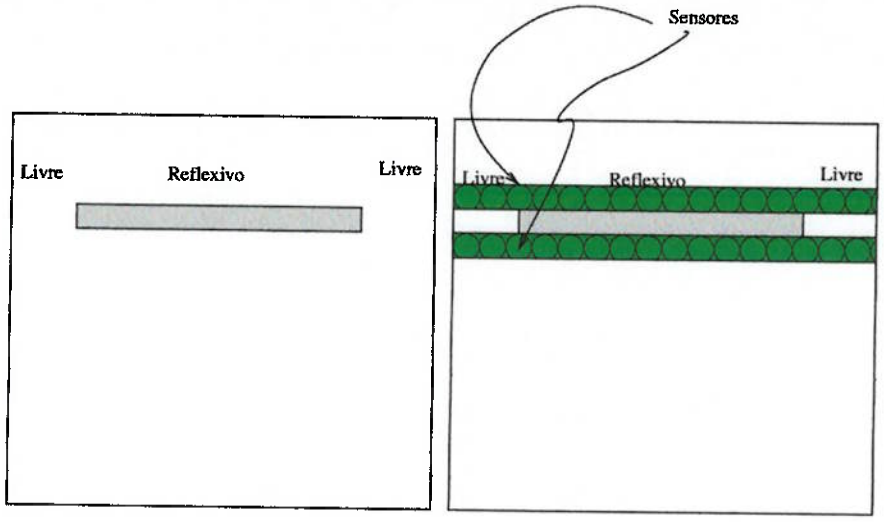
Figura 12.4: Teste do buraco



(a) idéia (b) montagem

A figura 12.4 ilustra o que chamamos de teste do buraco. Com a grade dividida em três, a parte do meio da linha é colocada como livre de maneira que temos um "buraco".

Figura 12.5: Teste do Corpo

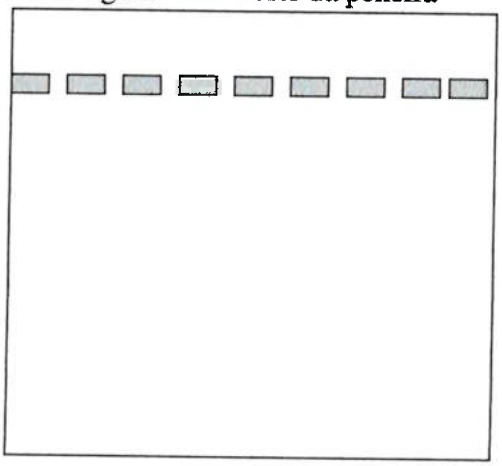


(a) idéia (b) montagem

A figura 12.5 apresenta a idéia contrária do buraco. Dessa vez, somente o terço do meio é considerado reflexivo de maneira que temos o efeito de um corpo linear que está em meio ao espaço de propagação.

Por último temos o teste da peneira, representada na figura 12.6.

Figura 12.6: Teste da peneira



Esse teste consiste de pontos reflexivos e livres alternados.

A otimização é feita através de uma biblioteca de otimização desenvolvida durante o projeto que utiliza Algoritmos Genéticos

Alguns Resultados

Foram realizados uma série de testes para que se pudesse determinar a eficiência de cada teste frente ao modelo desenvolvido para resolução de problemas inversos. O sistema que apresentou mais dificuldade de convergência foi o do corpo, que tendia a parar antes do devido e o mais simples foi o do buraco. O pequeno número de testes se dá devido ao grande custo computacional da simulação. Cada teste desse leva pelo menos 40 horas para ser completado. A manutenção da variedade genética também se faz importante aqui, não havendo convergência para populações do GA muito inferiores a 100.

Tabela 12.1: Tabela de resultados

Tipo	sucesso	tamanho	simulações
buraco	sim	30x30	12017
buraco	sim	100x100	50000 [1]
buraco	não	100x100	29396
buraco	sim	100x100	47287
buraco	sim	100x100	39439
corpo	sim	100x100	—
peneira	sim	100x100	75117

12.5 Procura de parâmetros de um circulo

O próximo passo foi a parametrização do problema. Determinamos que a barreira seria circular e reflexiva, totalmente contida dentro do sistema. Com isso, reduzimos as possibilidades de barreiras a variação de três variáveis:

1. posição do centro com relação a linha;
2. posição do centro com relação a coluna;
3. raio do círculo.

Isso fez com que escolhêssemos como método de procura da barreira a varredura exaustiva. Com isso tivemos sucesso, apesar de adicionarmos erros Gaussianos como anteriormente em todos os testes realizados.

Capítulo 13

Conclusão

13.1 Autômatos celulares

Conforme discutido durante este trabalho, uma onda, ou difusão pode ser simulada de maneira eficiente utilizando o método dos autômatos celulares, que resolveu com elegância e eficiência tanto o regime transiente como o regime permanente de propagação. O interessante deste modelo é a facilidade de implementação, muito maior do que por diferenças finitas por exemplo e a facilidade com que podem ser colocadas dentro do meio condições de contorno.

No entanto a conclusão mais importante não é sobre o modelo em si, mas sim sobre o que modelar usando autômatos celulares representa. Mais do que uma nova técnica, usamos aqui uma nova maneira de abordar o problema e de pensar sobre ele. Essa nova ciência [5] que só está sendo estudada agora com o barateamento do tempo computacional é extremamente interessante do ponto de vista da engenharia e justifica estudos mais profundos sobre o tema.

13.2 Sistemas de otimização

Durante este trabalho podemos perceber também a importância do estudo de sistemas de otimização de processos para ciências de engenharia. É só perceber que se um método de otimização não tivesse sido utilizado não existiria neste trabalho nenhum resultado para ser mostrado. O método utilizado aqui, algoritmos genéticos, mostrou-se eficiente na resolução de problemas envolvendo autômatos celulares reduzindo talvez de anos de processamento para dias os testes que executamos.

13.3 Problemas inversos

Quanto a problemas inversos podemos perceber por este trabalho algumas importantes conclusões:

1. alta instabilidade;

2. extremamente custoso.

A resolução de problemas inversos, no nosso caso o detector de barreiras se mostrou extremamente custoso no ponto de vista computacional. Mesmo com um computador relativamente poderoso, levamos alguns dias para terminar um teste simples de uma linha reflexiva $(50 - 6666660 horas)$ usando algoritmos genéticos. Caso contrário poderíamos levar anos analisando todas as probabilidades. Se pensarmos que levamos 50 horas para analisar 50000 possibilidades, usando uma regra de 3 simples podemos constatar que levaríamos 3169126500570573503741758O1.344 horas para analisar todas as possibilidades, ou 3617724315719832766828 4.908829224 anos, ou 36177243157198.327668285 bilhões de anos.

E pior de tudo é que podemos passar esse tempo todo efetuando cálculos e ainda por cima chegar em um resultado duvidoso, uma vez que constamos que o problema é altamente instável devido aos diversos erros e incertezas inerentes a eles.

13.4 Software livre

Durante a execução deste trabalho diversos programas foram utilizados para, desde escrever relatórios e documentação, até o desenvolvimento de programas para testes dos modelos desenvolvidos. Todos os programas utilizados neste trabalho foram programas livres, ou seja, que não é necessário pagar para obter uma licença de uso. Isso foi inclusive um dos objetivos do trabalho, provar que é possível realizar um trabalho de engenharia usando somente esse tipo de programa.

No entanto, ao final do trabalho constatei um outro fato que até certo ponto foi surpreendente: caso não existissem programas livres esse trabalho não teria sido possível de ser executado. Só o fato de não termos utilizado sistemas operacionais e programas de escritório proprietários nos gera uma economia de cerca de 5000 reais, quantia exorbitante visto a realidade de um mero estudante. Isso demonstra a importância do desenvolvimento de programas livres e nos levou a de um certo modo a tomar a decisão de que todos os programas desenvolvidos durante a execução deste trabalho que têm

alguma importância ou potencial de serem utilizados por outros usuários seriam, além de publicados na integra ao final deste trabalho, licenciados pela licença da *FSF - Free Software Foundation GPL.*

13.5 Objetivos alcançados

Os quatro objetivos principais foram alcançados. A propagação de ondas sonoras e a difusão foram modelada matematicamente, foi feito o estudo de validação de ambos os modelos e implementados computacionalmente. O detector de barreiras também foi implementado com sucesso assim como foi executado um estudo generalizado.

Quanto aos objetivos secundários, todos puderam ser atingidos também. A propagação de ondas sonoras foi modelada a partir não de um modelo específico, mas de um modelo que propaga qualquer tipo de onda, mecânica ou não. Isso aumentou a possibilidade de reutilização do código e o número de possíveis aplicações. Uma biblioteca que implementa algoritmos genéticos foi reescrito e aperfeiçoado, de maneira a possibilitar a implementação do detector de barreiras. Como saída gráfica utilizamos os arquivos do formato *.pgm* e *.ppm* como dito anteriormente. Quanto ao uso de programas livres, nenhum programa utilizado neste projeto foi programa proprietário, ou seja, este objetivo também foi atingido.

13.6 Idéias para possíveis trabalhos

O resultado deste trabalho é de natureza genérica de maneira que possibilita a reutilização do código gerado aqui para o progresso de trabalhos futuros. Existe ainda muito espaço para estudo de problemas inversos por exemplo. O Efeito da parametrização tem que ser melhor abordado e pode se mostrar um instrumento poderoso. A difisão de um problema genérico em uma série de problemas parametrizados pode ser a resposta para conseguir aumentar a generalidade do problema sem aumentar muito o custo computacional.

Autômatos celulares também apresentam muito campo para estudo e aplicações mais específicas. Em um trabalho futuro, poderíamos tentar aproximar o sistema teórico gerado aqui desenvolvendo comparações com dados experimentais de maneira a possibilitar aplicações de engenharia mais imediatas. Desse modo teríamos mais dados com relação ao comportamento de ondas e amostras o que nos permitiria fazer um levantamento mais preciso da função densidade de probabilidade a posteriori.

Apêndice A

Código fonte da biblioteca de algorítimos genéticos

A.1 Considerações

A GAOptim é uma biblioteca desenvolvida durante este trabalho em *C++* para servir como ferramenta de otimização através de algorítimos genéticos. Sua forma em template possibilita a otimização de funções especificadas ou de sistemas programados como classes que contenham algumas caractrísticas importantes como uma função *double eval(int *vet, int Size)*. Essa biblioteca foi licenciada através da licença *GPL*, só podendo ser utilizada futuramente de acordo com os termos da licença.

O sistema necessita de uma classe mesmo quando vai otimizar uma função. Como conforto uma classe *classe* é fornecida junto aos arquivos.

A.2 Chromossome.h

```
1
2 // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
3
4 // Copyright (C) 2003 Heitor Honda Federico.
5 //
6 //This file is part of the GAOptim library
7 //
8 // GAOptim is free
9 // software; you can redistribute it and/or modify it
```

```
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // GAOptim is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34
35 #ifndef CHROMOSSOME_H
36 #define CHROMOSSOME_H
37 #include<iostream>
38 #include<assert.h>
39 #include "RandomFree.h"
40 typedef double (*PtrToFunc)(int *Vet, int Size);
41     //funcao que pode ser passada para o programa
42     //otimizar.  Obrigatoriamente recebe um vetor
43     //e seu tamanho e retorna
44     //um valor de double que vai ser maximizado.
```

```
45
46 using namespace std;
47
48 template <class System>
49 class Chromossome{
50         public:
51                 //////////////
52                 //atributos//
53                 //////////////
54                 int Size;
55                 //guarda o tamanho do cromossomo
56                 double Fitness;
57                 //guarda o fenotipo do cromossomo
58                 int *GenBox;
59                 //vetor que carrega os gens
60                 int Base;
61                 //Guarda a base dos cromossomos
62                 int *Seed;
63                 //semente para numeros aleatorios
64                 bool ChromossomeType;
65                 //se false, cromossomo tem PtrToFunc
66                 //se true, cromossomo tem
67                     //um System Chromossome
68                 PtrToFunc FuncaoObjetivo;
69                 //funcao fornecida para o cromossomo
70                 System *SystemChromossome;
71                 Chromossome *Next;
72                 //ponteiro para o 6prximo nodo
73                 static int NumberOfEvals;
74                 //numero de evals calculada em
75                     //uma execucao do programa
76                 ////////////////
77                 //construtores//
78                 ////////////////
79                 Chromossome(); //default constructor
```

```
80                  Chromossome(int SizeChromossome,
81                      int *SeedChromossome,int BaseB,
82                      PtrToFunc FuncaoObj);
83                      //contrutor com funcao objetivo
84                  Chromossome(int SizeChromossome,
85                      int *SeedChromossome, int BaseB,
86                      System *SystemObject);
87                      //construtor com objeto
88                      //generico para ser otimizado
89                  //////////////
90                  //destrutores//
91                  //////////////
92                  ~Chromossome(); //destroi um cromossomo
93                  ////////////////////
94                  //funcoes e metodos//
95                  ////////////////////
96                  void generate();
97                  //atribui numeros  aleatorios
98                  //a um cromossomo em uma base
99                  void eval();
100                 //atribui o valor do fenotipo
101                 void printScreen();
102                 //imprime o fenotipo e o genotipo na tela
103                 void chromCopy(Chromossome *Chr);
104                 //copia o cromosomo atual
105                 //para o cromossomo apontado por Chr
106 };
107 ////////////////////////////////////////////////////////
108 //Inicializacao de membros áestticos//////////////////
109 ////////////////////////////////////////////////////////
110 template<class System>
111 int Chromossome<System>::NumberOfEvals = 0;
112     //inicializacao do membro estatico que
113     //conta o numero de evals
114
```

```
115 //////////////////////////////////////////////////
116 //Construtores: alocam a memoria necessaria e
117 //inicializa os atributos com valores dados
118 //////////////////////////////////////////////////
119 //#include "Chromossome.h"
120 template<class System>
121 Chromossome<System>::Chromossome(){
122         //a ser implementado
123         cout<<
124         "O Construtor default ainda
125 nao foi implementado !!!!!"<<endl;
126 };
127 template<class System>
128 Chromossome<System>::Chromossome(
129 int SizeChromossome, int *SeedChromossome,
130 int BaseB, PtrToFunc FuncaoObj){
131         Size = SizeChromossome;
132         Seed = SeedChromossome;
133         Fitness = 0.0;
134         Base=BaseB;
135         ChromossomeType = false;
136         //sabemos que o cromossomo tem um PtrToFunc
137         FuncaoObjetivo = FuncaoObj;
138         //funcao a ser maximizada
139         assert(Size>0);
140         //nao faz sentido um cromossomo
141         //de tamanho 0 ou negativo
142         GenBox = new int[Size];
143         assert (GenBox!=NULL);
144         //nao conseguiu alocar memoria
145         SeedChromossome = NULL;
146 };
147 template<class System> Chromossome<System>::
148 Chromossome(int SizeChromossome,
149 int *SeedChromossome, int BaseB,
```

```
150 System *SystemObject){
151         Size = SizeChromossome;
152         Seed = SeedChromossome;
153         Fitness = 0.0;
154         Base=BaseB;
155         ChromossomeType = true;
156         //sabemos que o cromossomo
157         //tem um objeto sistema
158         SystemChromossome = SystemObject;
159         //sistema a ser otimizado
160         assert(Size>0);
161         //nao faz sentido um cromossomo
162         //de tamanho 0 ou negativo
163         GenBox = new int[Size];
164         assert (GenBox!=NULL);
165         //nao conseguiu alocar memoria
166         SeedChromossome = NULL;
167         SystemObject = NULL;
168 };
169 //////////////////////////////////////////
170 //Destrutor:
171 //liberam a memoria reservada para o cromossomo
172 //////////////////////////////////////////
173 template<class System>
174 Chromossome<System>::~Chromossome(){
175         delete []GenBox;
176         Size = 0;
177         Seed = NULL;
178         Fitness = 0.0;
179         Base = 0;
180         ChromossomeType=false;
181         Next = NULL;
182         //evita destruir o proximo da lista
183         SystemChromossome=NULL;
184         //evita de destruir o objeto otimizado
```

```
185             GenBox=NULL;
186             //so pra garantir
187     };
188     ////////////////////////////////////////////////
189     //generate: gera numeros aleatorios
190     //para o genotipo em base Base
191     ////////////////////////////////////////////////
192     template<class System>
193     void Chromossome<System>::generate(){
194             //o metodo determina o numero
195             //k como sendo o inverso da base,
196             //criando assim uma proporcao de
197             //0-->Base e 0-->1
198             //em seguida, determina um numero
199             //randomico r entre 0 e 1.
200             //Feito isso, e determinado em que
201             //intervalo o numero r esta e que
202             //numero na base Base do cromossomo
203             //isso corresponde
204             double k = (double) 1/Base;
205             for(int i=0; i<Size; i++){
206                     double r = randomNumber(*Seed);
207                     int j;
208                     for(j = 1; r > (j*k); j++);
209                     GenBox[i]=j-1;
210             }
211     };
212     ////////////////////////////////////////////////
213     //printscreen: imprime na tela o
214     //fenotipo e genotipo do chromossomo
215     ////////////////////////////////////////////////
216     template<class System>
217     void Chromossome<System>::printScreen(){
218             cout<<"\n\n"; //pula duas linhas
219             cout<<
```

```
220             "O fenotipo do cromossomo eh: "<<Fitness<<endl;
221             cout<<
222             "O genotipo do cromossome eh: "<<endl;
223             for(int i = 0; i<Size; i++){
224                     cout<<GenBox[i]<<" ";
225             }
226             cout<<endl;
227     };
228     ////////////////////////////////////////////////////
229     //chromcopy: copia por valor um cromossomo para outro
230     ////////////////////////////////////////////////////
231     template<class System>
232     void Chromossome<System>::chromCopy(Chromossome *Chr){
233             assert (Chr->Size==Size);
234             //verifica se os cromossomos sao compativeis
235             assert (Chr->ChromossomeType==ChromossomeType);
236             //Chr->Size = Size;
237             //nao eh realmente necessario
238             Chr->Fitness = Fitness;
239             for(int i=0; i<Size; i++){
240                     Chr->GenBox[i]=GenBox[i];
241             }
242             Chr->Seed = Seed;
243             Chr->Base = Base;
244             Chr->ChromossomeType = ChromossomeType;
245             if(ChromossomeType)
246             Chr->SystemChromossome = SystemChromossome;
247             else Chr->FuncaoObjetivo = FuncaoObjetivo;
248             Chr=NULL;
249     };
250     ////////////////////////////////////////////////////
251     //eval: calcula o fenotipo definido para o cromossomo
252     ////////////////////////////////////////////////////
253     template<class System> void Chromossome<System>::eval(){
254             if(ChromossomeType) Fitness =
```

```
255                 SystemChromossome->eval(GenBox, Size);
256             else Fitness = FuncaoObjetivo(GenBox, Size);
257             ++Chromossome<System>::NumberOfEvals;
258 };
259
260 #endif
```

A.3 Chromlist.h

```
1  / GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33
34 #ifndef CHROMLIST_H
```

```
35 #define CHROMLIST_H
36 #include "Chromossome.h"
37
38 typedef double (*PtrToFunc)(int *vet, int size);
39
40 using namespace std;
41
42 template <class System>
43 class Chromlist{
44         public:
45         ////////////////
46         //Construtores//
47         ////////////////
48         Chromlist(PtrToFunc Funcao,
49         int BaseChromossomo);
50         //controi um lista de
51         //cromossomos com uma funcao de otimizacao
52         Chromlist(System *SystemObject,
53         int BaseChromossomo);
54         //constroi uma lista de cromossomos
55         //com um sistema para otimizar
56         Chromlist();
57         ////////////////
58         //Destrutor/////
59         ////////////////
60         ~Chromlist();
61         ////////////////////
62         //funcoes e metodos//
63         ////////////////////
64         void prepend(Chromossome<System> *Chr);
65         //adiona um cromossomo jah existente a lista
66         void prepend(int SizeChromossome,
67         int *SeedChromossome);
68         //cria e adiciona um cromossomo a lista
69         void del();
```

```
70          //deleta a cabeca da lista
71          void release();
72          //deleta todos os cromossomos
73          void sort();
74          //ordena os cromossomos de acordo com a Fitness
75          void printAll() const;
76          Chromossome<System>* head() const;
77          //////////////
78          //atributos//
79          //////////////
80          Chromossome<System> *H;
81          //aponta para a cabeca da lista
82          PtrToFunc FuncaoObjetivo;
83          //guarda uma funcao a otimizar
84          System *SystemChromlist;
85          //guarda um sistema a otimizat
86          bool ChromlistType;
87          //se verdadeiro guarda um sistema,
88          //se nao um funcao
89          int Base; //Base dos cromossomos da lista
90  };
91  ////////////////////////////////////////////////////
92  //Construtores, criam a lista vazia para
93  //adicionar cromossomos e determinam o tipo da lista
94  ////////////////////////////////////////////////////
95  template <class System> Chromlist<System>::
96  Chromlist(PtrToFunc Funcao, int BaseChromossomo){
97          H=NULL; //lista vazia
98          FuncaoObjetivo=Funcao;
99          ChromlistType=false;
100         Base=BaseChromossomo;
101 };
102 template <class System> Chromlist<System>::
103 Chromlist(System *SystemObject, int BaseChromossomo){
104         H=NULL; //lista vazia
```

```
105         SystemChromlist=SystemObject;
106         ChromlistType=true;
107         SystemObject=NULL;
108         Base=BaseChromossomo;
109 };
110 template <class System> Chromlist<System>::
111 Chromlist(){
112         cout<<"Construtor default
113 ________nao implementado"<<endl;
114 };
115 ////////////////////////////////////////////////
116 //Destrutores - destroem a lista inteira /////////
117 ////////////////////////////////////////////////
118 template <class System>
119 Chromlist<System>::~Chromlist(){
120         release();
121 };
122 ////////////////////////////////////////////////
123 //prepends-adicionam um cromossomo a lista /////////
124 ////////////////////////////////////////////////
125 template <class System> void Chromlist<System>::
126 prepend(Chromossome<System> *Chr){
127         Chr->Next=H;
128         H=Chr;
129         Chr=NULL;
130 };
131 template <class System> void Chromlist<System>::
132 prepend(int SizeChromossome, int *SeedChromossome){
133         if(ChromlistType){
134                 Chromossome<System> *Temp =
135                 new Chromossome<System>
136                     (SizeChromossome, SeedChromossome,
137                     Base, SystemChromlist);
138                 assert(Temp!=NULL);
139                 Temp->generate();
```

```
140                     Temp->eval();
141                     Temp->Next=H;
142                     H=Temp;
143                     Temp=NULL;
144             }
145             else{
146                     Chromossome<System> *Temp =
147                     new Chromossome<System>(SizeChromossome,
148                         SeedChromossome, Base, FuncaoObjetivo);
149                     assert(Temp!=NULL);
150                     Temp->generate();
151                     Temp->eval();
152                     Temp->Next=H;
153                     H=Temp;
154                     Temp=NULL;
155             }
156 };
157 ////////////////////////////////////////////////
158 //del(): deleta o elemento da cabeca da lista////
159 ////////////////////////////////////////////////
160 template <class System>
161 void Chromlist<System>::del(){
162             assert(H!=NULL); //lista esta vazia?
163             Chromossome<System> *Temp=H;
164             H=H->Next;
165             delete Temp;
166 };
167 ////////////////////////////////////////////////////
168 //release(): esvazia a lista//////////////////////
169 ////////////////////////////////////////////////////
170 template <class System>
171 void Chromlist<System>::release(){
172             while(H!=NULL){
173                     del();
174             }
```

```
175 };
176 //////////////////////////////////////////////
177 //printall(): imprime toda a lista ///////////////
178 //////////////////////////////////////////////
179 template <class System>
180 void Chromlist<System>::printAll() const{
181         Chromossome<System> *Temp=H;
182         while(Temp!=NULL){
183                 Temp->printScreen();
184                 Temp=Temp->Next;
185         }
186         cout<<"\n###"<<endl;
187         Temp=NULL;
188 };
189 //////////////////////////////////////////////
190 //sort(): ordena a lista
191 //ligada do menor ao maior H->menor
192 //////////////////////////////////////////////
193 template <class System>
194 void Chromlist<System>::sort(){
195         Chromossome<System> *H2;
196         Chromossome<System> *Ord;
197         Chromossome<System> *Now;
198         //
199         Ord=H;
200         H=H->Next;
201         Ord->Next=NULL;
202         //
203         Now=H;
204         H=H->Next;
205         Now->Next=NULL;
206         //
207         if(Ord->Fitness>=Now->Fitness){
208                 Now->Next=Ord;
209                 Ord=Now;
```

```
210         }
211         else{
212                 Ord->Next=Now;
213         }
214         //
215         while(H!=NULL){
216                 Now=H;
217                 H=H->Next;
218                 Now->Next=NULL;
219                 H2=Ord;
220                 if(Ord->Fitness>=Now->Fitness){
221                         Now->Next=Ord;
222                         Ord=Now;
223                 }
224                 else{
225                         while (1){
226                                 if(H2->Next==NULL){
227                                         H2->Next=Now;
228                                         break;
229                                 }
230                                 else{
231                 if(H2->Next->Fitness>Now->Fitness){
232                                 Now->Next=H2->Next;
233                                 H2->Next=Now;
234                                 break;
235                                 }
236                                 }
237                                 H2=H2->Next;
238                         }
239                 }
240
241         }
242         H=Ord;
243         Ord=NULL;
244         Now=NULL;
```

```
245             H2=NULL;
246     };
247     //////////////////////////////////////
248     //head(): Retorna a cabeca da lista /////
249     //////////////////////////////////////
250     template <class System> Chromossome<System>*
251     Chromlist<System>::head() const{
252             return(H);
253     };
254     #endif
```

A.4 Population.h

```
1  / GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33 #ifndef POPULATION_H
34 #define POPULATION_H
```

```
35 #include "Chromlist.h"
36
37 typedef double (*PtrToFunc)(int *vet, int size);
38
39 using namespace std;
40
41 template <class System>
42 class Population{
43         public:
44                 //////////////////
45                 //atributos ///////
46                 //////////////////
47                 int SizePopulation;
48                 //Tamanho da populacao
49                 int SizeChromossome;
50                 //Tamanho dos cromossomos da populacao
51                 int SizeOffSpring;
52                 //Numero de filhos da geracao
53                 int Seed;
54                 //Semente para numeros aleatorios
55                 int Base;
56                 //Base dos cromossomos
57                 double PCross;
58                 //Probabilidade de crossover
59                 double PMutat;
60                 //Probabilidade de mutacao
61                 double Prop;
62                 //Proporcao da melhor
63                 //populacao que sobrevive
64                 Chromossome<System>
65                 *BestFit; //melhor cromossomo
66                 Chromossome<System>
67                 *Head;
68                 //guarda a cabeca
69                 //do tamanho original
```

```
70      Chromlist<System> *ChromBox;
71      //guarda os cromossomos
72      PtrToFunc FuncaoObjetivo;
73      //funcao a ser otimizada
74      bool PopulationType;
75      //true se otimizar sistema,
76      //false se otimizar funcao
77      bool ChromossomeChange;
78      //Especial para CA_GA,
79      //verifica mudancas no cromossomo
80      //quando existe uma melhora
81      //da fitness
82      System *SystemPopulation;
83      ////////////////////
84      //Construtores////
85      ////////////////////
86      Population();
87      Population(PtrToFunc Funcao,
88      int SizePop, int SizeChrom, double Cross,
89      double Mutat, int BaseB, double PropSurv);
90      //Constroi uma populacao baseada em uma funcao
91      Population(System *SystemObject, int SizePop,
92      int SizeChrom, double Cross, double Mutat,
93      int BaseB, double PropSurv);
94      //Constroi uma populacao baseada em um sistema
95      ////////////////////
96      //Destrutores/////
97      ////////////////////
98      ~Population();
99      ////////////////////////
100     //Metodos e funcoes////
101     ////////////////////////
102     void crossover(); //processo de crossover
103     void mutate();
104     //processo de mutacao criando um novo
```

```
105                     //cromossomo a cada mutacao
106                     void mutateOnly();
107                     //processo de mutacao criando um novo
108                     //cromossomo com todas as mutacoes
109                     //daquele cromossomo
110                     void select(); //quem sobrevive?
111                     void printPopulation();
112                     //imprime toda a populacao na tela
113                     void generation(int HowMany);
114                     //roda as geracoes com mutate
115                     void generationOnly(int HowMany);
116                     //roda as çõgeraes com mutateOnly
117                     void getSeed();
118                     //le a semente de um arquivo seed
119                     void setSeed();
120                     //coloca uma nova semente no arquivo seed
121                     void crossChrom(Chromossome<System> *Chr1,
122                     Chromossome<System> *Chr2);
123                     //processo auxiliar do crossover
124                     void addChromossome(Chromossome<System>*Chr);
125                     //adiciona um cromossomo
126                     //especifico na populacao
127                     void addSolution(int *Solution);
128                     //adiciona uma solucao dada num genotipo
129 };
130 ////////////////////////////////////////////////
131 //Construtores : criam a populacao//////////////
132 ////////////////////////////////////////////////
133 template<class System> Population<System>::
134 Population(){
135         cout<<"Construtor␣defaut␣nao␣implementado";
136 };
137 template<class System> Population<System>::
138 Population(PtrToFunc Funcao, int SizePop,
139 int SizeChrom, double Cross, double Mutat,
```

```
140 int BaseB, double PropSurv){
141         SizePopulation=SizePop;
142         SizeChromossome=SizeChrom;
143         PCross=Cross;
144         PMutat=Mutat;
145         Base=BaseB;
146         Prop=PropSurv;
147         SizeOffSpring=0;
148         FuncaoObjetivo=Funcao;
149         getSeed(); //busca uma seed no arquivo
150         ChromBox = new Chromlist<System>(FuncaoObjetivo,
151         Base);
152         for(int i=0;i<SizePopulation;i++){
153                 ChromBox->prepend(SizeChromossome,
154                 &Seed);
155         }
156         ChromBox->sort();
157 //////////////////////primeiro melhor///////////////
158         BestFit = new Chromossome<System>
159         (SizeChromossome, &Seed, Base, FuncaoObjetivo);
160         Chromossome<System> *Temp;
161         Temp=ChromBox->H;
162         while(Temp->Next!=NULL) Temp=Temp->Next;
163         Temp->chromCopy(BestFit);
164         Temp=NULL;
165 ////////////////////////////////////////////////////
166         Head=ChromBox->H;
167         PopulationType=false;
168         ChromossomeChange=false;
169 };
170 template<class System> Population<System>::
171 Population(System *SystemObject, int SizePop,
172 int SizeChrom, double Cross, double Mutat, int BaseB,
173 double PropSurv){
174         SizePopulation=SizePop;
```

```
175         SizeChromossome=SizeChrom;
176         PCross=Cross;
177         PMutat=Mutat;
178         Base=BaseB;
179         Prop=PropSurv;
180         SizeOffSpring=0;
181         SystemPopulation = SystemObject;
182         getSeed(); //busca uma seed no arquivo
183         ChromBox = new Chromlist<System>
184         (SystemPopulation, Base);
185         for(int i=0;i<SizePopulation;i++){
186                 ChromBox->prepend(SizeChromossome, &Seed);
187         }
188         ChromBox->sort();
189 //////////////////////primeiro melhor/////////////////
190         BestFit = new Chromossome<System>
191         (SizeChromossome, &Seed, Base, SystemPopulation);
192         Chromossome<System> *Temp;
193         Temp=ChromBox->H;
194         while(Temp->Next!=NULL) Temp=Temp->Next;
195         Temp->chromCopy(BestFit);
196         Temp=NULL;
197 ////////////////////////////////////////////////////
198         Head=ChromBox->H;
199         PopulationType=true;
200         ChromossomeChange=false;
201 };
202 ////////////////////////////////////////////////////
203 //destrutor: libera a memoria ocupada pela populacao//
204 ////////////////////////////////////////////////////
205 template<class System> Population<System>::~Population(){
206         delete ChromBox;
207         delete BestFit;
208         setSeed();
209 };
```

```
210 //////////////////////////////////////////////////////
211 //getseed le a semente do arquivo seed e coloca outra
212 //setseed colota a semente no arquivo seed   (sao usados
213 //no constructor e destructor)
214 //////////////////////////////////////////////////////
215 template <class System> void Population<System>::getSeed(){
216         ifstream in("seed");
217         assert(in!=NULL);
218         in>>Seed;
219         in.close();
220         double H = randomNumber(Seed);
221         //çãinicializao da semente—ver man
222 };
223 template <class System> void Population<System>::setSeed(){
224         ofstream out("seed");
225         int k;
226         k=(int)(randomNumber(Seed)*10000)*(-1);
227         out<<k;
228         out.close();
229 };
230 //////////////////////////////////////////////////////
231 //printPopulation: imprime a populacao inteira//////
232 //////////////////////////////////////////////////////
233 template <class System> void Population<System>::
234 printPopulation(){
235         cout<<"\n\nA␣populacao␣eh:";
236         ChromBox->printAll();
237 };
238 //////////////////////////////////////////////////////
239 //generation:executa uma itercao completa de GA //
240 //////////////////////////////////////////////////////
241 template <class System> void Population<System>::
242 generation(int HowMany){
243         for(int j=0;j<HowMany;j++){
244                 ChromossomeChange=false;
```

```
245                     crossover();
246                     mutate();
247                     select();
248                     //////// Seleciona o melhor ////////
249                     Chromossome<System> *Temp;
250                     Temp=ChromBox->H;
251                     while(Temp->Next!=NULL)
252                     Temp=Temp->Next;
253                     for(int j=0; j<SizeChromossome; j++)
254                     {
255                         if(BestFit->GenBox[j]!=
256                         Temp->GenBox[j])
257                         {
258                             ChromossomeChange=true;
259                         }
260                     }
261                     Temp->chromCopy(BestFit);
262                     Temp=NULL;
263                     //system("clear");
264                     //cout<<"O melhor agora eh
265                     //(geracao="<<j<<"):"<<endl;
266                     //BestFit->printScreen();
267                     ////////////////////////////////
268             }
269 };
270 ////////////////////////////////////////////////////
271 //generationOnly: executa uma itercao completa de GA /
272 ////////////////////////////////////////////////////
273 template <class System> void Population<System>::
274 generationOnly(int HowMany){
275         for(int j=0;j<HowMany;j++){
276                     crossover();
277                     mutateOnly();
278                     select();
279                     //////// Seleciona o melhor ///////
```

```
280                     Chromossome<System > *Temp;
281                     Temp=ChromBox->H;
282                     while(Temp->Next!=NULL)
283                     Temp=Temp->Next;
284                     for(int j=0; j<SizeChromossome; j++)
285                     {
286                         if(BestFit->GenBox[j]!=
287                         Temp->GenBox[j])
288                         {
289                             ChromossomeChange=true;
290                         }
291                     }
292                     Temp->chromCopy(BestFit);
293                     Temp=NULL;
294                     //system("clear");
295                     //cout<<"O melhor agora eh
296                     //(geracao="<<j<<"):"<<endl;
297                     //BestFit->printScreen();
298                     //////////////////////////////////
299             }
300 };
301
302 ////////////////////////////////////////////////////
303 //crossover : parte do GAé, feito entre a populacao
304 //inicial ////////////////////////////////////////
305 ////////////////////////////////////////////////////
306 template <class System> void Population<System>::
307 crossover(){
308             int Par=0; //indica se o cromossomo
309             //selecionado no momentoé o primeiro
310             //<par==0> ou segundo <par==1>
311             Chromossome<System > *Temp;
312             Chromossome<System > *Chr;
313             Temp=Head;
314             while(Temp!=NULL){
```

```
315                     double Rk = (randomNumber(Seed));
316                     if(Rk<PCross){
317                             if(Par==0){
318                                     Chr=Temp;
319                                     Par=1;
320                             }
321                             else{
322                                     crossChrom(Chr, Temp);
323                                     Par=0;
324                                     SizeOffSpring++;
325                             }
326                     }
327                     Temp=Temp->Next;
328             }
329             Chr=NULL;
330             Temp=NULL;
331     };
332     template <class System> void Population<System>::
333     crossChrom(Chromossome<System> *Chr1, Chromossome<System>
334     *Chr2){
335             double Rk = randomNumber(Seed);
336             int Pos = (int) ((double)SizeChromossome*Rk);
337             //Aproximacao suficiente
338             //..... alternativa : perguntar ao professor .//
339             /* int Pos=0;
340             double K=1/SizeChromossome;
341             for(int i=0;i<SizeChromossome-1;i++){
342                     Rk=randomNumber(Seed);
343                     if(Rk<K){
344                         Pos=i;
345                         break;
346                     }
347             }*/
348             ////////////////////////////////////////////
349             if(PopulationType){
```

```
350                     Chromossome<System > *Temp =
351                     new Chromossome<System >(SizeChromossome,
352                     &Seed, Base, SystemPopulation);
353                     int i;
354                     for(i=0;i<Pos;i++){
355                             Temp->GenBox[i]=Chr1->GenBox[i];
356                     }
357                     for(;i<SizeChromossome;i++){
358                             Temp->GenBox[i]=Chr2->GenBox[i];
359                     }
360                     Temp->eval();
361                     ChromBox->prepend(Temp);
362                     Chr1=NULL;
363                     Chr2=NULL;
364                     Temp=NULL;
365             }
366             else{
367                     Chromossome<System > *Temp = new
368                     Chromossome<System >(SizeChromossome,
369                     &Seed, Base, FuncaoObjetivo);
370                     int i;
371                     for(i=0;i<Pos;i++){
372                             Temp->GenBox[i]=Chr1->GenBox[i];
373                     }
374                     for(;i<SizeChromossome;i++){
375                             Temp->GenBox[i]=Chr2->GenBox[i];
376                     }
377                     Temp->eval();
378                     ChromBox->prepend(Temp);
379                     Chr1=NULL;
380                     Chr2=NULL;
381                     Temp=NULL;
382             }
383 };
384 ////////////////////////////////////////////////////
```

```
385 //Mutate -->soh ocorre entre a populacao inicial//
386 ////////////////////////////////////////////////
387 template <class System> void Population<System>::
388 mutate(){
389     Chromossome<System> *Temp;
390     Temp=Head;
391     while(Temp!=0){
392             for(int i=0; i<SizeChromossome;i++){
393                     double k = randomNumber(Seed);
394                     if(k<PMutat){
395                     if(PopulationType){
396                     Chromossome<System>
397                     *aux = new
398                     Chromossome<System>
399                     (SizeChromossome,
400                     &Seed, Base,
401                     SystemPopulation);
402                     for(int j=0;
403                     j<SizeChromossome;j++){
404                     aux->GenBox[j]=
405                     Temp->GenBox[j];
406                     }
407                     /*******************/
408
409                     double rand = (double)1/Base;
410                     double rand2 = randomNumber(Seed);
411                     int m;
412                     for(m=1; rand2>m*rand;m++);
413                     aux->GenBox[i]=m-1;
414                     /*******************/
415                     SizeOffSpring++;
416                     aux->eval();
417                     ChromBox->prepend(aux);
418                     aux=NULL;
419                 }
```

```
420                     else{
421                         Chromossome<System> *aux = new
422                         Chromossome<System>(SizeChromossome,
423                         &Seed, Base, FuncaoObjetivo);
424                         for(int j=0;j<SizeChromossome;j++){
425                         aux->GenBox[j]=Temp->GenBox[j];
426                         }
427                         /**********************/
428                         double rand = (double)1/Base;
429                         double rand2 = randomNumber(Seed);
430                         int m;
431                         for(m=1; rand2>m*rand;m++);
432                         aux->GenBox[i]=m-1;
433                         /**********************/
434                         SizeOffSpring++;
435                         aux->eval();
436                         ChromBox->prepend(aux);
437                         aux=NULL;
438                     }
439
440                 }
441             }
442             Temp=Temp->Next;
443         }
444         Temp=NULL;
445 };
446 ////////////////////////////////////////////////
447 //MutateOnly --> soh
448 //ocorre entre a populacao inicial//
449 ////////////////////////////////////////////////
450 template <class System>
451 void Population<System>::mutateOnly(){
452             Chromossome<System> *Temp;
453             Temp=Head;
454             while(Temp!=0){
```

```
455             if(PopulationType)
456             {
457 //copia o cromossomo para um cromossomo auxiliar
458                 bool Mutation=false;
459                 Chromossome<System> *aux = new
460                 Chromossome<System>
461                 (SizeChromossome, &Seed, Base,
462                 SystemPopulation);
463                 for(int j=0;j<SizeChromossome;j++)
464                 {
465                     aux->GenBox[j]=Temp->GenBox[j];
466                 }
467                 for(int i=0; i<SizeChromossome;i++)
468                 {
469                     double k = randomNumber(Seed);
470                     if(k<PMutat)
471                     {
472                     //executa a mutacao
473                     //e avisa que executou
474                     double rand = (double)1/Base;
475                     double rand2 = randomNumber(Seed);
476                     int m;
477                     for(m=1; rand2>m*rand;m++);
478                     aux->GenBox[i]=m-1;
479                     Mutation=true;
480                     //aconteceu a mutacao
481                     }
482                 }
483                 if(Mutation)
484                 {
485                     SizeOffSpring++;
486                     aux->eval();
487                     ChromBox->prepend(aux);
488                     aux=NULL;
489                 }
```

```
490                 else
491                 {
492                     delete aux;
493                 }
494
495             }
496             else
497             {   bool Mutation=false;
498                 Chromossome<System> *aux =
499                 new Chromossome<System>
500                 (SizeChromossome, &Seed,
501                 Base, FuncaoObjetivo);
502                 for(int j=0;j<SizeChromossome;j++)
503                 {
504                     aux->GenBox[j]=Temp->GenBox[j];
505                 }
506                 for(int i=0; i<SizeChromossome;i++)
507                 {
508                     double k = randomNumber(Seed);
509                     if(k<PMutat)
510                     {
511                         //executa a
512                         //mutacao e avisa que executou
513                         double rand = (double)1/Base;
514                         double rand2 = randomNumber(Seed);
515                         int m;
516                         for(m=1; rand2>m*rand;m++);
517                         aux->GenBox[i]=m-1;
518                         Mutation=true;
519                         //aconteceu a mutacao
520                     }
521                 }
522                 if(Mutation)
523                 {
524                     SizeOffSpring++;
```

```
525                         aux->eval();
526                         ChromBox->prepend(aux);
527                         aux=NULL;
528                     }
529                     else
530                     {
531                         delete aux;
532                     }
533                 }
534                 Temp=Temp->Next;
535             }
536             Temp=NULL;
537 };
538 ////////////////////////////////////////////////////////
539 //Select: seleciona quem passa para a proxima geracao/////
540 ////////////////////////////////////////////////////////
541 template <class System> void Population<System>::select(){
542             ChromBox->sort();
543             int i=SizeOffSpring;
544             while (i!=0){
545                     ChromBox->del();
546                     i--;
547             }
548             double surv=1.0-Prop;
549             int survivors = (int) surv*SizePopulation;
550             for(i=0;i<survivors;i++) ChromBox->del();
551             for(i=0;i<survivors;i++) ChromBox->prepend
552             (SizeChromossome, &Seed);
553             ChromBox->sort();
554             Head=ChromBox->H;
555             SizeOffSpring=0;
556 };
557 ////////////////////////////////////////////
558 //addChromossome:
559 //adiciona um cromossomo a uma populacao/////
```

```
560 //Esse novo membro
561 //entra como um cromossomo filho //////////////
562 //////////////////////////////////////////////////
563 template <class System> void Population<System> ::
564 addChromossome(Chromossome<System> *Chr)
565 {
566     assert(Chr->Size == SizeChromossome);
567     SizeOffSpring++;
568     Chr->eval();
569     ChromBox->prepend(Chr);
570     Chr=NULL;
571 };
572 //////////////////////////////////////////////////
573 //addSolution adiciona
574 //um cromossomo dado um certo genotipo
575 //////////////////////////////////////////////////
576 template <class System> void Population<System> ::
577 addSolution(int *Solution)
578 {
579     Chromossome<System> *Chr =
580     new Chromossome<System>(SizeChromossome,
581     &Seed, Base, SystemPopulation);
582     for(int i=0; i<SizeChromossome; i++)
583     {
584         Chr->GenBox[i] = Solution[i];
585     }
586     addChromossome(Chr);
587     Chr=NULL;
588 };
589 #endif
```

A.5 Optimizer.h

```
1  // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33 #ifndef OPTIMIZER_H
34 #define OPTIMIZER_H
```

```
35 #include <fstream>
36 #include <string>
37 #include "Population.h"
38
39
40 typedef double (*PtrToFunc)(int *vet, int size);
41
42 using namespace std;
43
44 template <class System>
45 class Optimizer
46 {
47     protected:
48         ////////////////
49         //atributos /////
50         ////////////////
51             Population<System> *PopulationToOptimize;
52             char *OutputFile;
53         ////////////////
54         //metodos ///////
55         ////////////////
56
57     public:
58
59         ////////////////
60         //atributos /////
61         ////////////////
62
63
64         ////////////////
65         //construtores //
66         ////////////////
67         Optimizer();
68         Optimizer(PtrToFunc Funcao, int SizePop,
69         int SizeChrom, double Cross, double Mutat,
```

```
70          int BaseB, double PropSurv,
71          char *OutputFileName);
72          //Constroi uma populacao baseada em
73          //uma funcao
74          Optimizer(System *SystemObject,
75          int SizePop, int SizeChrom, double Cross,
76          double Mutat, int BaseB, double PropSurv,
77          char *OutputFileName);
78          //Constroi uma populacao
79          //baseada em um sistema
80          ////////////////
81          //Destrutores/////
82          ////////////////
83          ~Optimizer();
84
85          ////////////////
86          //metodos///////
87          ////////////////
88          virtual void stepsOfOptimization
89          (int HowManySteps, int StopAtStep,
90          bool RecordFile, bool DontUseMutationOnly);
91          //Faz a otimizacao da populacao
92          //com os parametros dados
93          //HowManySteps numero maximo de iteracoes
94          //StopAtStep numero de iteracoes
95          //sem evolucao permitida
96          /RecordFile grava em um arquivo chamdo
97          //steps o resultado das iteracoes
98          Chromossome<System>* returnBestFit();
99          //retorna um ponteiro para
100         //o chromossomo bestFit
101         void addChromossome(Chromossome<System> *Chr);
102         //adiciona um cromossomo na populacao como filho
103         void addSolution(int *Solution);
104         //adiciona um cromossomo a partir
```

```
105         //de um genotipo como filho
106 };
107 ////////////////////////////////////////////////////////
108 //Construtores e destrutores, alocam
109 //a memoria necessaria para armazenar a populacao
110 ////////////////////////////////////////////////////////
111 template <class System> Optimizer<System> :: Optimizer()
112 {
113     cerr<<"Sao necessarios parametros"<<endl;
114     exit(1);
115 };
116 template <class System> Optimizer<System>::
117 Optimizer(PtrToFunc Funcao, int SizePop, int SizeChrom,
118 double Cross, double Mutat, int BaseB, double PropSurv,
119 char *OutputFileName)
120 {
121     PopulationToOptimize = new P
122     opulation<System>(Funcao, SizePop, SizeChrom,
123     Cross, Mutat, BaseB, PropSurv);
124     OutputFile=OutputFileName;
125 };
126 template <class System> Optimizer<System>::
127 Optimizer(System *SystemObject, int SizePop, int SizeChrom,
128 double Cross, double Mutat, int BaseB, double PropSurv,
129 char *OutputFileName)
130 {
131     PopulationToOptimize = new
132     Population<System>(SystemObject, SizePop,
133     SizeChrom, Cross, Mutat, BaseB, PropSurv);
134     OutputFile=OutputFileName;
135 };
136 template <class System> Optimizer<System>::~Optimizer()
137 {
138     delete PopulationToOptimize;
139 };
```

```
140 //////////////////////////////////////////////
141 //stepsOfOptimization : Faz a otimizacao
142 //  observando duas regras de parada
143 //numero maximo de steps HowManySteps;
144 //numero de Steps sem melhora StopAtStep
145 //grava em um arquivo
146 //////////////////////////////////////////////
147 template <class System> void Optimizer<System> ::
148 stepsOfOptimization(int HowManySteps, int StopAtStep,
149 bool RecordFile, bool DontUseMutationOnly)
150 {
151     double BestFit;
152     int WithoutImprovement=0;
153     //abertura do arquivo
154     char FileName[80]="";
155     char FileNameLatex[80]="";
156     strcat(FileName, OutputFile);
157     strcat(FileNameLatex, OutputFile);
158     strcat(FileNameLatex, "latex");
159     ofstream FileOut(FileName);
160     //saida no formato gnuplot
161     ofstream FileOutLatex(FileNameLatex);
162     //saida no formato de tabela para relatorio
163     assert(FileOut!=NULL);
164     //verifica a abertura do arquivo
165     assert(FileOutLatex!=NULL);
166
167     BestFit=PopulationToOptimize->BestFit->Fitness;
168
169     if(RecordFile)
170     {
171         FileOut<<"0\t"<<BestFit<<endl;
172         FileOutLatex<<"0 & "<<BestFit<<" &";
173         for(int i=0;
174         i<PopulationToOptimize->BestFit->Size; i++)
```

```
175         {
176             FileOutLatex <<"␣"<<
177             PopulationToOptimize->BestFit->GenBox[i];
178         }
179         FileOutLatex <<endl;
180     }
181
182     for(int i=0; i<HowManySteps;i++)
183     {
184         if(WithoutImprovement<
185         (StopAtStep-1)/5 &&
186         DontUseMutationOnly==false)
187         {
188             PopulationToOptimize->
189             generationOnly(1);
190         }
191         else
192         {
193             DontUseMutationOnly=true;
194             //nao volta mais pro mutate only
195     //      PopulationToOptimize->PMutat=0.0005;
196             //valor padrao adotado
197             PopulationToOptimize->generation(1);
198         }
199         if(RecordFile)
200         {
201             FileOut<<(i+1)<<"\t"<<BestFit<<endl;
202             FileOutLatex <<(i+1)<<"␣&␣"<<BestFit<<"␣&";
203             for(int j=0;
204             j<PopulationToOptimize->BestFit->Size; j++)
205             {
206                 FileOutLatex <<"␣"<<
207                 PopulationToOptimize->BestFit->GenBox[j];
208             }
209             FileOutLatex <<endl;
```

```
210         }
211         if(PopulationToOptimize->BestFit->Fitness>BestFit)
212         {
213             BestFit=PopulationToOptimize->BestFit->Fitness;
214             if(PopulationToOptimize->ChromossomeChange)
215             {
216                 WithoutImprovement=0;
217             }
218         }
219         else
220         {
221             if(WithoutImprovement<StopAtStep-1)
222             {
223                 WithoutImprovement++;
224             }
225             else
226             {
227                 system("clear");
228                 cout<<"Processo␣Parado␣na␣"<<i<<
229                 "␣iteracao␣pois␣passou␣de␣"<<StopAtStep<<
230                 "␣iteracoes␣sem␣melhora!"<<endl;
231                 cout<<"Numero␣de␣avaliacoes␣calculados:␣"<<
232                 Chromossome<System>::NumberOfEvals<<endl;
233                 cout<<"O␣melhor␣(geracao="<<i<<"):"<<endl;
234                 PopulationToOptimize->BestFit->printScreen();
235                 FileOutLatex.close();
236                 FileOut.close();
237                 return;
238             }
239         }
240         system("clear");
241         cout<<"O␣melhor␣agora␣eh(geracao="<<i<<"):"<<
242             endl;
243         cout<<"Passou-se␣"<<WithoutImprovement<<
244         "␣geracoes␣sem␣melhora."<<endl;
```

```
245             cout<<"Numero de avaliacoes calculados: "<<
246             Chromossome<System>::NumberOfEvals<<endl;
247             PopulationToOptimize->BestFit->printScreen();
248         }
249         system("clear");
250         cout<<"Processo Parado depois de "<<
251         HowManySteps<<" iteracoes!"<<endl;
252         cout<<"O melhor (geracao=" <<
253         HowManySteps<<"):"<<endl;
254         cout<<"Numero de avaliacoes calculados: "<<
255         Chromossome<System>::NumberOfEvals<<endl;
256         PopulationToOptimize->BestFit->printScreen();
257         FileOutLatex.close();
258         FileOut.close();
259         return;
260 };
261 ////////////////////////////////////////////
262 //returnBestFit
263 //retorna um ponteiro pro chromossomo bestfit
264 ////////////////////////////////////////////
265 template <class System> Chromossome<System>*
266 Optimizer<System> :: returnBestFit()
267 {
268     return(PopulationToOptimize->BestFit);
269 };
270 ////////////////////////////////////////////
271 //addChromossome adiciona um
272 //cromossomo na populacao como filho
273 ////////////////////////////////////////////
274 template <class System> void Optimizer<System> ::
275 addChromossome(Chromossome<System> *Chr)
276 {
277     PopulationToOptimize->addChromossome(Chr);
278     Chr=NULL;
279 };
```

```
280 //////////////////////////////////////////////////
281 //addSolution adiciona um filho dado um genotipo
282 //necessario conferir se o filhoé valido///////
283 //////////////////////////////////////////////////
284 template <class System> void Optimizer<System> ::
285 addSolution(int *Solution)
286 {
287     PopulationToOptimize->addSolution(Solution);
288 };
289
290 #endif
```

A.6 Gerador de números aleatórios

A.6.1 RandomFree.h

```
1  // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
```

```
33 //////////////////////////////////////////////////
34 //Geracao de numeros aleatorios usando a bibliote///
35 //ca livre do C/////////////////////////////////////
36 //////////////////////////////////////////////////
37 #ifndef RANDOMFREE_H
38 #define RANDOMFREE_H
39     #include<iostream>
40     #include<cstdlib>
41     #include<cmath>
42     using namespace std;
43
44     double randomNumber(int &Seed);
45         //numero aleatorio
46         //real entre
47         //0 e 1.
48     double randomGaussian(int &Seed);
49         //numero aleatorio real
50         //seguinte uma distribuicao
51         //gaussiana (normal)
52         //de media 0 e
53         //desvio padrao 1
54
55 #endif
```

A.6.2 RandomFree.cpp

```
1 // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3 // Copyright (C) 2003 Heitor Honda Federico.
4 //
5 //This file is part of the GAOptim library
6 //
7 // GAOptim is free
8 // software; you can redistribute it and/or modify it
9 //under the  terms of the GNU General Public License
```

```
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33
34 #include "RandomFree.h"
35 double randomNumber(int &Seed)
36 {
37     if(Seed<=0)
38     //çãinicializao quando a semente
39     //menor do que zero
40     {
41         if(Seed==0)
42         {
43             Seed=-1;
44             //se a semente fornecida for 0,
```

```
45                  //usa-se a semente ãpadro -1
46              }
47              Seed=-Seed;
48              srandom(Seed);
49          }
50          return(((double)(random()))/(RAND_MAX+1.0));
51  };
52  double randomGaussian(int &Seed)
53  {
54          double aux ;
55          static double NormalDeviateNext;
56          //guarda um numero aleatorio na distribuicao normal
57          double Rand1;
58          //guarda um numero aleatorio entre -1 e 1
59          double Rand2;
60          //guarda um numero aleatorio entre -1 e 1
61          double Sphere;
62          //guarda o calculo do quadrado do
63          //raio de Rand1 e Rand2
64          bool IsSphere = false;
65          //boleano,  eh esfera == true
66          static bool NormalDeviateReady;
67          //escolher dois numeros dentro
68          //de uma esfera áunitria:
69          if(Seed < 0)
70          {
71              NormalDeviateReady = false;
72              //inicializa
73          }
74          if(!NormalDeviateReady)
75          {
76              while (!IsSphere)
77              {
78                  Rand1 = 2.0*randomNumber(Seed) - 1.0;
79                  //estamos em um quadrado
```

```
80              Rand2 = 2.0*randomNumber(Seed) - 1.0;
81              //unitario
82              Sphere = pow(Rand1, 2) + pow(Rand2, 2);
83              //soma dos quadrados
84              if(Sphere >= 1 || Sphere == 0);
85              //tentar de novo
86              else
87              {
88                  IsSphere=true;
89                  //encontramos um ponto
90                  //na esfera áunitria
91              }
92          }
93          aux = sqrt(-2.0*log(Sphere)/Sphere);
94          NormalDeviateNext = Rand1*aux;
95          NormalDeviateReady = true;
96          return(Rand2*aux);
97      }
98      else
99      {
100         NormalDeviateReady = 0;
101         return (NormalDeviateNext);
102     }
103 };
```

A.7 Classe *classe*

A.7.1 classe.h

```
1  // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
```

```
33 #ifndef CLASSE_H
34 #define CLASSE_H
35 #include<iostream>
36
37 using namespace std;
38 class classe{
39         public:
40                 int a;
41                 double eval(int *vet, int size);
42                 classe(int aa);
43 };
44 #endif
45
46 \end{lstliting}
47 \subsection{classe.cpp}
48 \begin{lstlisting}[]{}
49 // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
50
51 // Copyright (C) 2003 Heitor Honda Federico.
52 //
53 //This file is part of the GAOptim library
54 //
55 // GAOptim is free
56 // software; you can redistribute it and/or modify it
57 //under the  terms of the GNU General Public License
58 //as published by the Free Software Foundation;
59 //either version 2, or (at your option) any later
60 //version.
61
62 // GAOptim is distributed in the hope that it will
63 //be useful, but WITHOUT ANY WARRANTY; without
64 //even the implied warranty of MERCHANTABILITY or
65 //FITNESS FOR A PARTICULAR PURPOSE.  See the
66 // GNU General Public License for more details.
67
```

```
68 // You should have received a copy of the GNU General
69 //Public License along with this library; see the
70 //file COPYING.  If not, write to the Free Software
71 //Foundation, 59 Temple Place - Suite 330,
72 //Boston, MA 02111-1307, USA.
73
74
75 //Heitor Honda Federico
76 //sardaukar@uol.com.br
77 //Rua Saint Tropez 565, Jardim Mediterraneo
78 //Cotia, Sao Paulo Brazil
79 //CEP:06708-700
80
81 #include "classe.h"
82 classe::classe(int aa){
83         a=aa;
84 //       cout<<"Criei"<<endl;
85 }
86 double classe::eval(int *vet, int size){
87         double value=0.0;
88         for(int i=0;i<size;i++)  value=value+vet[i];
89         vet=NULL;
90         value=value+a;
91         return (-value);
92 }
```

A.8 GAOptim

```
1  // GAOptim: Genetic Algorithim optimizator -*- C++ -*-
2
3  // Copyright (C) 2003 Heitor Honda Federico.
4  //
5  //This file is part of the GAOptim library
6  //
7  // GAOptim is free
8  // software; you can redistribute it and/or modify it
9  //under the  terms of the GNU General Public License
10 //as published by the Free Software Foundation;
11 //either version 2, or (at your option) any later
12 //version.
13
14 // GAOptim is distributed in the hope that it will
15 //be useful, but WITHOUT ANY WARRANTY; without
16 //even the implied warranty of MERCHANTABILITY or
17 //FITNESS FOR A PARTICULAR PURPOSE.  See the
18 // GNU General Public License for more details.
19
20 // You should have received a copy of the GNU General
21 //Public License along with this library; see the
22 //file COPYING.  If not, write to the Free Software
23 //Foundation, 59 Temple Place - Suite 330,
24 //Boston, MA 02111-1307, USA.
25
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33 #ifndef GAOPTIM_H
34 #define GAOPTIM_H
```

```
35 #include "Optimizer.h"
36 #include "classe.h"
37 #endif
```

Apêndice B

Código fonte do programa para gerar Cellular Automata 1D

B.1 Considerações

Este programa não tem intenção de reutilização de código, sendo escrito somente para a geração de exemplos de Cellular Automata 1D. Consiste de um único arquivo.

B.2 RuleSelect.cpp

```
1  #include<iostream>
2  #include<fstream>
3
4  using namespace std;
5
6  int** allocLattice(int Line, int Column);
7  void freeLattice(int Line, int **Lattice);
8  void read(int &a,int &b,int &c,int &d,int &e,int &f,int &g,
9      int &h,int &Line, int &Column);
10 void nextStep(int a,int b,int c,int d,int e,int f,int g,int h,
11     int Line, int Column, ofstream &File, int **Lattice);
12 void simulate();
13
14
15 int main()
```

```
16 {
17     simulate();
18     return 0;
19 };
20
21 void read(int &a,int &b,int &c,int &d,int &e,int &f,int &g,
22     int &h,int &Line, int &Column)
23 {
24     cout<<"Entre com o numero de passos: ";
25     cin>>Line;
26     cout<<"Entre com o numero de casas: ";
27     cin>>Column;
28     cout<<"Entre com a regra:"<<endl;
29     cout<<"1 1 1 = ";
30     cin>>a;
31     cout<<"1 1 0 = ";
32     cin>>b;
33     cout<<"1 0 1 = ";
34     cin>>c;
35     cout<<"1 0 0 = ";
36     cin>>d;
37     cout<<"0 1 1 = ";
38     cin>>e;
39     cout<<"0 1 0 = ";
40     cin>>f;
41     cout<<"0 0 1 = ";
42     cin>>g;
43     cout<<"0 0 0 = ";
44     cin>>h;
45     return;
46 }
47
48 int** allocLattice(int Line, int Column)
49 {
50     int **Lattice;
```

```
51      assert(Line>0&&Column>0);
52      assert(Column-1>Line);
53      Lattice = new int*[Line];
54      assert(Lattice!=NULL);
55      for(int i=0; i<Line;i++)
56      {
57          Lattice[i] = new int[Column];
58          assert(Lattice[i]!=NULL);
59      }
60      return (Lattice);
61  }
62  void freeLattice(int Line, int **Lattice)
63  {
64      for(int i=0; i<Line;i++)
65      {
66          delete []Lattice[i];
67      }
68      delete []Lattice;
69  }
70  void nextStep(int a,int b,int c,int d,int e,int f,int g,
71      int h,int Line, int Column, ofstream &File, int **Lattice)
72  {
73      for(int i=1; i<Line;i++)
74      {
75          for(int j=0;j<Column;j++)
76          {
77              if(j!=0 && j!=Column-1){
78                  if(Lattice[i-1][j-1]==1&&
79                  Lattice[i-1][j]==1&&Lattice[i-1][j+1]==1)
80                  {
81                      Lattice[i][j]=a;
82                  }
83                  if(Lattice[i-1][j-1]==1&&
84                  Lattice[i-1][j]==1&&Lattice[i-1][j+1]==0)
85                  {
```

```
86                      Lattice[i][j]=b;
87                  }
88                  if(Lattice[i-1][j-1]==1&&
89                  Lattice[i-1][j]==0&&Lattice[i-1][j+1]==1)
90                  {
91                      Lattice[i][j]=c;
92                  }
93                  if(Lattice[i-1][j-1]==1&&
94                  Lattice[i-1][j]==0&&Lattice[i-1][j+1]==0)
95                  {
96                      Lattice[i][j]=d;
97                  }
98                  if(Lattice[i-1][j-1]==0&&
99                  Lattice[i-1][j]==1&&Lattice[i-1][j+1]==1)
100                 {
101                     Lattice[i][j]=e;
102                 }
103                 if(Lattice[i-1][j-1]==0&&
104                 Lattice[i-1][j]==1&&Lattice[i-1][j+1]==0)
105                 {
106                     Lattice[i][j]=f;
107                 }
108                 if(Lattice[i-1][j-1]==0&&
109                 Lattice[i-1][j]==0&&Lattice[i-1][j+1]==1)
110                 {
111                     Lattice[i][j]=g;
112                 }
113                 if(Lattice[i-1][j-1]==0&&
114                 Lattice[i-1][j]==0&&Lattice[i-1][j+1]==0)
115                 {
116                     Lattice[i][j]=h;
117                 }
118             }
119             File<<Lattice[i][j]<<endl;
120             cout<<Lattice[i][j];
```

```
121             }
122             cout<<endl;
123         }
124
125 }
126 void simulate(){
127         int a, b, c, d, e, f, g, h;
128             //letras para armazenar regras
129         int Line;
130             //numero de passos == numero de linhas da matriz
131         int Column;
132             //numero de automatas
133         int **Lattice;
134             //armazena o resultado no tempo
135         ofstream File("saida.pgm");
136             //arquivo com a figura
137         read(a, b, c, d, e, f, g, h, Line, Column);
138             //leitura dos dados
139         Lattice = allocLattice(Line, Column);
140             //aloca memoria para a grade
141         File<<"P2"<<endl;
142         File<<Column<<"␣"<<Line<<endl;
143         File<<"1"<<endl;
144         for(int i=0; i<Line; i++)
145         {
146             for(int j=0; j<Column;j++)
147             {
148                 Lattice[i][j]=0;
149             }
150         }
151         Lattice[0][(Column/2)-1]=1;//estado inicial
152         for(int i=0;i<Column;i++)
153         {
154             File<<Lattice[0][i]<<endl;
155         }
```

```
156     nextStep(a, b, c, d, e, f, g, h,
157         Line, Column, File, Lattice);
158     freeLattice(Line, Lattice);
159 };
```

Apêndice C

Código fonte dos programas de Cellular Automata 2D e do detector de barreiras

C.1 Considerações

Os arquivos *CellularAutomata2D*, *Lattice2D*, *WaveSim2D* e *DiffusionSim2D* constituem o que chamamos de módulo de propagação de ondas e difusão bidimensionais em meio plano e retangular. Através das heranças da classe *WaveSim2D* podemos atingir diversos outros objetivos como reflexão de linha e amostragem. A classe *System* foi criada para um fim específico de simulação de detecção de barreiras e utiliza classes derivadas do módulo.

O arquivo da função principal *int main()* também se encontra aqui e contém o código utilizado para testes e geração das imagens colocadas neste documento.

C.2 Classe *CellularAutomata2D*

C.2.1 CellularAutomata2D.h

```
1 // CAPropag2D: 2D Wave and Diffusion simulator
2  //using Cellular Automata -*- C++ -*-
3
4 // Copyright (C) 2003 Heitor Honda Federico.
5 //
```

```
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27 //Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
33 #define CELLULARAUTOMATA2D_H
34
35 #include<iostream>
36 #include<cmath>
37
38 using namespace std;
39 class CellularAutomata2D
40 {    ////////////////////////////////
```

```
41      //membros e metodos privados//
42      ////////////////////////////////
43      private:
44          ////////////
45          //atributos//
46          ////////////
47          double F1;  //      f2
48          double F2;  //       |
49          double F3;  //f3<---f0--->f1
50          double F4;  //       |
51          double F0;  //      f4
52
53          bool IsReflective; //Se o pontoé  de Reflexao
54
55          /////////
56          //metodos//
57          /////////
58          void setF0(double Value);
59          void setF1(double Value);
60          void setF2(double Value);
61          void setF3(double Value);
62          void setF4(double Value);
63
64      ////////////////////////////////
65      //membros e metodos publicos//
66      ////////////////////////////////
67      public:
68          ////////////
69          //atributos//
70          ////////////
71          double F; //soma vetorial de Fi i=[0,4]
72          ////////////////////////////////
73          //Construtores e destrutores//
74          ////////////////////////////////
75          CellularAutomata2D();
```

```
76          //aloca o automato : construtor default
77          ~CellularAutomata2D();
78          //destrutor : ainda nao implementado
79
80          //////////
81          //metodos//
82          //////////
83          void plusF0(double Value); // soma ao valor
84          void plusF1(double Value); // Fi o valor
85          void plusF2(double Value); // contido em Value
86          void plusF3(double Value); //
87          void plusF4(double Value); //
88
89          double returnF0(); //retorna o valor de Fi
90          double returnF1(); //
91          double returnF2(); //
92          double returnF3(); //
93          double returnF4(); //
94
95          void print();
96          //imprime o valor de Fi com i [0,4]
97          void init(double f0, double f1,
98          double f2, double f3, double f4);
99          //inicializa com um valor determinado;
100         void initZero();
101         //zera os valores de Fi com i [0,4]
102         void calcF();
103         void setIsReflective(bool Reflective);
104         //seta a variavel IsRelfective
105         bool returnIsReflective();
106 };
107 #endif
```

C.2.2 CellularAutomata2D.cpp

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27 #include "CellularAutomata2D.h"
28
29 //////////////////////////////////////////////
30 //construtor--coloca zero nos valores       //
31 //destrutor nao implementado                //
32 //////////////////////////////////////////////
33 CellularAutomata2D::CellularAutomata2D()
34 {
35     initZero();
```

```
36      F=0.0;
37  }
38  CellularAutomata2D::~CellularAutomata2D()
39  {
40      //implementar
41      //   cout<<"testando";
42  }
43  //////////////////////////////////////////
44  //private setFi(): coloca um
45  //valor determinado em Fi
46  //////////////////////////////////////////
47  void CellularAutomata2D::setF1(double Value)
48  {
49      F1=Value;
50  }
51  void CellularAutomata2D::setF2(double Value)
52  {
53      F2=Value;
54  }
55  void CellularAutomata2D::setF3(double Value)
56  {
57      F3=Value;
58  }
59  void CellularAutomata2D::setF4(double Value)
60  {
61      F4=Value;
62  }
63  void CellularAutomata2D::setF0(double Value)
64  {
65      F0=Value;
66  }
67  ////////////////////////////////////////////////
68  //public plus Fi : soma a Fi um valor
69  ////////////////////////////////////////////////
70  void CellularAutomata2D::plusF0(double value)
```

```
71  {
72      F0=F0+Value;
73  }
74  void CellularAutomata2D::plusF1(double Value)
75  {
76      F1=F1+Value;
77  }
78  void CellularAutomata2D::plusF2(double Value)
79  {
80      F2=F2+Value;
81  }
82  void CellularAutomata2D::plusF3(double Value)
83  {
84      F3=F3+Value;
85  }
86  void CellularAutomata2D::plusF4(double Value)
87  {
88      F4=F4+Value;
89  }
90  //////////////////////////////////////////////////
91  //plublic returnFi():retorna o valor de Fi
92  //////////////////////////////////////////////////
93  double CellularAutomata2D::returnF0()
94  {
95      return(F0);
96  }
97  double CellularAutomata2D::returnF1()
98  {
99      return(F1);
100 }
101 double CellularAutomata2D::returnF2()
102 {
103     return(F2);
104 }
105 double CellularAutomata2D::returnF3()
```

```
106 {
107     return(F3);
108 }
109 double CellularAutomata2D::returnF4()
110 {
111     return(F4);
112 }
113 //////////////////////////////////////////////////
114 //public print: imprimena tela o resultado
115 //////////////////////////////////////////////////
116 void CellularAutomata2D::print()
117 {
118     cout<<"\n\nOs valores sao:"<<endl;
119     cout<<"F0 = "<<F0<<endl;
120     cout<<"F1 = "<<F1<<endl;
121     cout<<"F2 = "<<F2<<endl;
122     cout<<"F3 = "<<F3<<endl;
123     cout<<"F4 = "<<F4<<endl;
124 }
125 //////////////////////////////////////////////////
126 //public init: inicializa com algum valor
127 //os Fi com i=[0,4]//
128 //////////////////////////////////////////////////
129 void CellularAutomata2D::init(double f0, double f1,
130 double f2, double f3, double f4)
131 {
132     setF0(f0);
133     setF1(f1);
134     setF2(f2);
135     setF3(f3);
136     setF4(f4);
137 }
138 //////////////////////////////////////////////////
139 //public initZero:
140 //inicializa os Fis com zero  com i=[0,4]///
```

```
141 //////////////////////////////////////////////////////
142 void CellularAutomata2D::initZero()
143 {
144     init(0.0, 0.0, 0.0, 0.0, 0.0);
145 }
146 //////////////////////////////////////////////////////
147 //public CalcF : faz a soma vetorial de fi //////////
148 //////////////////////////////////////////////////////
149 void CellularAutomata2D::calcF()
150 {/*
151     double A;
152     double B;
153     double C;
154
155     A=F1-F3;
156     B=F2-F4;
157
158     A=A*A;
159     B=B*B;
160
161     C=A+B;
162
163     F=sqrt(C)+F0;     */
164     //F=F1+F2+F3+F4+F0;
165     F=F1+F2+F3+F4+F0;
166 }
```

C.3 Classe *Lattice2D*

C.3.1 Lattice2D.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27 /Heitor Honda Federico
28 //sardaukar@uol.com.br
29 //Rua Saint Tropez 565, Jardim Mediterraneo
30 //Cotia, Sao Paulo Brazil
31 //CEP:06708-700
32
```

```
33 #ifndef LATTICE2D_H
34 #define LATTICE2D_H
35
36 #include<fstream>
37 //#include<string>
38 #include "CellularAutomata2D.h"
39
40 using namespace std;
41 class Lattice2D
42 {
43     ////////////////////////////////
44     //membros e metodos privados//
45     ////////////////////////////////
46     private:
47         ////////////
48         //atributos//
49         ////////////
50         int NumberOfLines;
51         //guarda o nume de linhas
52         int NumberOfColumns;
53         //guarda o numero de colunas
54         CellularAutomata2D ***CellBox;
55         //matriz de cellulares automatos
56         /////////
57         //metodos//
58         /////////
59         void createCellBox();
60         //inializa a matriz de cellulares
61         //automatos, usado no construtor
62         void releaseCellBox();
63         //deleta a matriz CellBox,
64         //usado no construtor
65     ////////////////////////////////
66     //membros e metodos publicos//
67     ////////////////////////////////
```

```
68      public:
69          //////////
70          //atributos//
71          //////////
72
73          ////////////////////////////
74          //Construtores e destrutores//
75          ////////////////////////////
76          Lattice2D(); //construtor default: erro
77          Lattice2D(int NumLin, int NumCol);
78              //aloca a matriz com o numero de
79              //linhas e colunas especificado e
80              //coloca zero nos sites
81          ~Lattice2D();
82          //destrutor : ainda nao implementado
83          /////////
84          //metodos//
85          /////////
86          void inputZero();
87          //coloca zero nos atributos dos
88          //automatos celulares de toda a grade
89          void calcF();
90          //calcula a soma vetorial
91          //de Fis para toda a malha i=[0, 4]
92          //e retorna o maior modulo para calculo
93          void plusCell(double f0, double f1,
94          double f2, double f3, double f4, int Line,
95          int Column); //adiciona ao automato cellular
96                       //um determinado valor em seus
97                       //atributos
98          double returnCellF0(int Line, int Column);
99          //retorna o valor de Fi
100         double returnCellF1(int Line, int Column);
101         //de uma determinado automato Line, Column
102         double returnCellF2(int Line, int Column);
```

```
103             //dentro da malha
104         double returnCellF3(int Line, int Column);
105         double returnCellF4(int Line, int Column);
106         void initCell(double f0, double f1, double f2,
107         double f3, double f4, int Line, int Column);
108             //Coloca em um automato cellular
109             //da linha Line e coluna Column
110             //os valores de f1, f2, f3, f4
111             //e f0
112         void setIsReflective(int Line, int Column,
113         bool IsReflective);
114         //seta se o siteé ou nao reflexivo
115         bool returnIsReflective(int Line, int Column);
116         //retorna se o siteé reflexivo
117         void printRealDataToFile(char *FileName);
118         void printToFile(double Accurate, char *FileName);
119         //escreve para um arquivo com o nome file
120         void printToFile(double Accurate);
121         //escreve para um arquivo padrao
122         //com a precisao especificada para
123         //valores entre 0 e Acurate*10
124         //Accurate [0 0.1]
125 };
126 #endif
```

C.3.2 Lattice2D.cpp

```
1 // CAPropag2D: 2D Wave and Diffusion simulator
2  //using Cellular Automata -*- C++ -*-
3
4 // Copyright (C) 2003 Heitor Honda Federico.
5 //
6 //This file is part of the CAPropag2D library
7 //
8 // CAPropag2D is free
```

```
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34 #include "Lattice2D.h"
35
36
37 ////////////////////////////////////////
38 //Construtor: constroi a
39 /grade inicial inicializando todos  //
40 //os sites com zero/
41 //Destrutor: nao esta implementado,
42 //deleta na verdade os     //
43 //sites
```

```
44 //////////////////////////////////////////
45 Lattice2D::Lattice2D()
46 {
47     cerr<<"Entre com dimensoes"<<endl;
48     exit(1);
49 };
50 Lattice2D::Lattice2D(int NumLin, int NumCol)
51 {
52     ////////////////////////////////////
53     //Recebe o tamanho e
54     //verifica a coerencia das dimensoes//
55     ////////////////////////////////////
56     NumberOfLines=NumLin;
57     NumberOfColumns=NumCol;
58     assert((NumberOfLines>0)&&(NumberOfColumns>0));
59     //verifica coerencia das dimensoes
60     ////////////////////////////////////////
61     //Inicializacao dos estados internos///////
62     ////////////////////////////////////////
63     createCellBox();
64 };
65
66 Lattice2D::~Lattice2D()
67 {
68     releaseCellBox();
69     //libera a memoria allocada
70     //para a matriz de cellulares automatos
71     CellBox=NULL;
72 };
73
74 //private CreateCellBox :
75 //monta a matriz de automatos cellulares//////
76 void Lattice2D::createCellBox()
77 {
78 //////////////////////////////////////
```

```
79      //Aloca os ponteiros que
80      //ãsero utilizados no processo///
81      ////////////////////////////////////
82      CellBox =
83      new CellularAutomata2D**[NumberOfLines];
84      assert(CellBox!=NULL);
85      //verifica sucesso da primeira alocacao
86      for(int i=0;i<NumberOfLines;i++)
87      {
88          CellBox[i] =
89          new CellularAutomata2D*[NumberOfColumns];
90          assert(CellBox[i]!=NULL);
91          //verifica o sucesso
92          //das alocacoes sucessivas
93      }
94      ////////////////////////////////////////////
95      //Inicializa os sites//////////////////////
96      ////////////////////////////////////////////
97      for(int i=0;i<NumberOfLines;i++)
98      {
99          for(int j=0;j<NumberOfColumns;j++)
100         {
101             CellBox[i][j] = new CellularAutomata2D();
102             assert(CellBox[i][j]!=NULL);
103             //verifica o sucesso da inicializacao
104         }
105     }
106 };
107
108 //private releaseCellBox desaloca
109 //a memoria usado pela matriz de cellulares automatos
110 void Lattice2D::releaseCellBox()
111 {
112     assert(CellBox!=NULL);
113     //verifica o inicio do delete se existe CellBox
```

```
114     for(int i=0;i<NumberOfLines;i++)
115     {
116         for(int j=0;j<NumberOfColumns;j++)
117         {
118             assert(CellBox[i][j]!=NULL);
119             //verifica se existe o site
120             delete CellBox[i][j];
121         }
122     }
123     for(int i=0;i<NumberOfLines;i++)
124     {
125         assert(CellBox[i]!=NULL);
126         delete []CellBox[i];
127     }
128     delete []CellBox;
129     CellBox = NULL;
130 };
131
132 //////////////////////////////////////////////////
133 //public inputZero:
134 //coloca zero em todos os atributos dos sites//////
135 //////////////////////////////////////////////////
136 void Lattice2D::inputZero()
137 {
138     for(int i=0; i<NumberOfLines; i++)
139     {
140         for(int j=0; j<NumberOfColumns; j++)
141         {
142             CellBox[i][j]->initZero();
143         }
144     }
145 };
146
147 //////////////////////////////////////////////
148 //public calcF :calcula a soma vetorial dos
```

```
149 //Fis para toda a malha i=[0, 4]////
150 ////////////////////////////////////////////
151 void Lattice2D::calcF()
152 {
153     for(int i=0; i<NumberOfLines; i++)
154     {
155         for(int j=0; j<NumberOfColumns; j++)
156         {
157             CellBox[i][j]->calcF();
158         }
159     }
160 };
161
162 ////////////////////////////////////////////
163 //public pluCell:adiciona
164 //valores aos valores
165 //jah existente de um automato ///
166 //cellular  em uma linha e
167 coluna determinada                          //
168 ////////////////////////////////////////////
169 void Lattice2D::
170 plusCell(double f0, double f1, double f2,
171 double f3, double f4, int Line, int Column)
172 {
173     assert(Line>=0 && Line<NumberOfLines &&
174     Column>=0 && Column<NumberOfColumns);
175     //existe uma linha Line e coluna Column
176     CellBox[Line][Column]->plusF0(f0);
177     CellBox[Line][Column]->plusF1(f1);
178     CellBox[Line][Column]->plusF2(f2);
179     CellBox[Line][Column]->plusF3(f3);
180     CellBox[Line][Column]->plusF4(f4);
181 };
182
183 ////////////////////////////////////////////
```

```
184 //plublic returnCellFi():
185 //retorna o valor de Fi  para um automato celular     //
186 //na linha Line e coluna Column
187 //////////////////////////////////////////////
188 double Lattice2D::returnCellF0(int Line, int Column)
189 {
190     assert(Line>=0 && Line<NumberOfLines &&
191     Column>=0 && Column<NumberOfColumns);
192     //existe uma linha Line e coluna Column
193     return(CellBox[Line][Column]->returnF0());
194 };
195 double Lattice2D::returnCellF1(int Line, int Column)
196 {
197     assert(Line>=0 && Line<NumberOfLines &&
198     Column>=0 && Column<NumberOfColumns);
199     //existe uma linha Line e coluna Column
200     return(CellBox[Line][Column]->returnF1());
201 };
202 double Lattice2D::returnCellF2(int Line, int Column)
203 {
204     assert(Line>=0 && Line<NumberOfLines
205     && Column>=0 && Column<NumberOfColumns);
206     //existe uma linha Line e coluna Column
207     return(CellBox[Line][Column]->returnF2());
208 };
209 double Lattice2D::returnCellF3(int Line, int Column)
210 {
211     assert(Line>=0 && Line<NumberOfLines &&
212     Column>=0 && Column<NumberOfColumns);
213     //existe uma linha Line e coluna Column
214     return(CellBox[Line][Column]->returnF3());
215 };
216 double Lattice2D::returnCellF4(int Line, int Column)
217 {
218     assert(Line>=0 && Line<NumberOfLines
```

```
219     && Column>=0 && Column<NumberOfColumns);
220     //existe uma linha Line e coluna Column
221     return(CellBox[Line][Column]->returnF4());
222 };
223
224 ////////////////////////////////////////////
225 //public:initCell :
226 //coloca no automato de linha Line e coluna column o
227 //valor dados pelos atributos fi. Usado para
228 //entradas continuas em frequencia///
229 ////////////////////////////////////////////////
230 void Lattice2D::
231 initCell(double f0, double f1, double f2,
232 double f3, double f4, int Line, int Column)
233 {
234     assert(Line>=0 && Line<NumberOfLines &&
235     Column>=0 && Column<NumberOfColumns);
236     //existe uma linha Line e coluna Column
237     CellBox[Line][Column]->init(f0, f1, f2, f3, f4);
238 };
239 ////////////////////////////////////////////////
240 //returnIsReflective  e setIsReflective
241 //setam e verificam se o site eh ////////
242 //reflexivo ////////////////////////////////////
243 ////////////////////////////////////////////////
244 void Lattice2D ::
245 setIsReflective(int Line, int Column, bool IsReflective)
246 {
247     assert(Line>=0);
248     assert(Line<NumberOfLines);        //teste de
249     assert(Column>=0);
250     assert(Column<NumberOfColumns); //coerencia
251     CellBox[Line][Column]->setIsReflective(IsReflective);
252 };
253 bool Lattice2D :: returnIsReflective(int Line, int Column)
```

```
254 {
255     assert(Line>=0 && Line<NumberOfLines);          //teste de
256     assert(Column>=0 && Column<NumberOfColumns); //coerencia
257     return(CellBox[Line][Column]->returnIsReflective());
258 };
259
260 ////////////////////////////////////////////////
261 //public printRealDataToFile
262 //imprime os dados reais obtidos no arquivo de nome
263 //especificado
264 ////////////////////////////////////////////////
265 void Lattice2D::printRealDataToFile(char *FileName)
266 {
267     ofstream FileOut(FileName);
268     assert(FileOut!=NULL);
269     //verifica a abertura do arquivo
270     calcF();
271     FileOut<<NumberOfLines<<"␣"<<
272     NumberOfColumns<<endl;
273     for(int i=0; i<NumberOfLines; i++)
274     {
275         for(int j=0; j<NumberOfColumns; j++)
276         {
277             FileOut<<"␣"<<CellBox[i][j]->F;
278             //FileOut<<i<<"\t"<<j<<"\t"<<
279             CellBox[i][j]->F<<endl;
280         }
281         FileOut<<endl;
282     }
283     FileOut.close();
284 };
285
286 ////////////////////////////////////////////
287 //public: printToFile imprime a
288 //soma vetorial de Fis no arquivo com o nome dado//
```

```
289 //////////////////////////////////
290 void Lattice2D::printToFile(double Accurate)
291 {
292     assert(Accurate>0&&Accurate<=0.1);
293     //abertura do arquivo
294     ofstream FileOut("figura.ppm");
295     assert(FileOut!=NULL);
296     //verifica a abertura do arquivo
297     //calculo dos valores a serem "plotados"
298     calcF();
299     //determinacao da escala
300     FileOut<<"P3"<<endl;
301     //tipo do arquivo
302     FileOut<<NumberOfColumns<<"␣"<<NumberOfLines<<endl;
303     //tamanho do arquivo
304     FileOut<<"10"<<endl;
305     //variedade de cores
306     for(int i=0; i<NumberOfLines; i++)//coloca os pontos
307     {
308         for(int j=0; j<NumberOfColumns; j++)
309         {
310             if(CellBox[i][j]->F==0)
311             {
312                 FileOut<<"10␣10␣10"<<endl;
313             }
314             else
315             {
316                 if(CellBox[i][j]->F>0)
317                 {
318                     if(CellBox[i][j]->F<Accurate*1)
319                     {
320                         FileOut<<"9␣10␣10"<<endl;
321                     }
322                     else
323                     {
```

```
324                 if(CellBox[i][j]->F<Accurate*2)
325                 {
326                     FileOut<<"8␣10␣10"<<endl;
327                 }
328                 else
329                 {
330                 if(CellBox[i][j]->F<Accurate*3)
331                 {
332                 FileOut<<"7␣10␣10"<<endl;
333                 }
334                 else
335                 {
336                 if(CellBox[i][j]->F<Accurate*4)
337                 {
338                 FileOut<<"6␣10␣10"<<endl;
339                 }
340                 else
341                 {
342                 if(CellBox[i][j]->F<Accurate*5)
343                 {
344                 FileOut<<"5␣10␣10"<<endl;
345                 }
346                 else
347                 {
348                 if(CellBox[i][j]->F<Accurate*6)
349                 {
350                 FileOut<<"4␣10␣10"<<endl;
351                 }
352                 else
353                 {
354                 if(CellBox[i][j]->F<Accurate*7)
355                 {
356                 FileOut<<"3␣10␣10"<<endl;
357                 }
358                 else
```

```
359                     {
360                     if(CellBox[i][j]->F<Accurate*8)
361                     {
362                     FileOut<<"2 10 10"<<endl;
363                     }
364                     else
365                     {
366                     if(CellBox[i][j]->F<Accurate*9)
367                     {
368                     FileOut<<"1 10 10"<<endl;
369                     }
370                     else
371                     {
372                     FileOut<<"0 10 10"<<endl;
373                     }
374                     }
375                     }
376                     }
377                     }
378                     }
379                     }
380                     }
381                 }
382             }
383             else
384             {
385                 if(CellBox[i][j]->F>=Accurate*1)
386                 {
387                     FileOut<<"10 10 9"<<endl;
388                 }
389                 else
390                 {
391                     if(CellBox[i][j]->F>=Accurate*2)
392                     {
393                         FileOut<<"10 10 8"<<endl;
```

```
394                 }
395                 else
396                 {
397                 if(CellBox[i][j]->F>=Accurate*3)
398                 {
399                 FileOut<<"10 10 7"<<endl;
400                 }
401                 else
402                 {
403                 if(CellBox[i][j]->F>=Accurate*4)
404                 {
405                 FileOut<<"10 10 6"<<endl;
406                 }
407                 else
408                 {
409                 if(CellBox[i][j]->F>=Accurate*5)
410                 {
411                 FileOut<<"10 10 5"<<endl;
412                 }
413                 else
414                 {
415                 if(CellBox[i][j]->F>=Accurate*6)
416                 {
417                 FileOut<<"10 10 4"<<endl;
418                 }
419                 else
420                 {
421                 if(CellBox[i][j]->F>=Accurate*7)
422                 {
423                 FileOut<<"10 10 3"<<endl;
424                 }
425                 else
426                 {
427                 if(CellBox[i][j]->F>=Accurate*8)
428                 {
```

```
429                                 FileOut<<"10 10 2"<<endl;
430                                 }
431                                 else
432                                 {
433                                 if(CellBox[i][j]->F>=Accurate*9)
434                                 {
435                                 FileOut<<"10 10 1"<<endl;
436                                 }
437                                 else
438                                 {
439                                 FileOut<<"10 10 0"<<endl;
440                                 }
441                                 }
442                                 }
443                                 }
444                                 }
445                                 }
446                                 }
447                                 }
448                             }
449                         }
450                     }
451                 }
452             }
453             FileOut.close();
454 };
455 ////////////////////////////////////////////////
456 ////////////////////////////////////////////////
457 ////////////////////////////////////////////////
458
459 void Lattice2D::
460 printToFile(double Accurate, char *FileName)
461 {
462     assert(Accurate>0&&Accurate<=0.1);
463     //abertura do arquivo
```

```
464     ofstream FileOut(FileName);
465     assert(FileOut!=NULL);
466     //verifica a abertura do arquivo
467     //calculo dos valores a serem "plotados"
468     calcF();
469     //determinacao da escala
470     FileOut<<"P3"<<endl;
471     //tipo do arquivo
472     FileOut<<NumberOfColumns<<"␣"<<NumberOfLines<<endl;
473     //tamanho do arquivo
474     FileOut<<"10"<<endl;
475     //variedade de cores
476     for(int i=0; i<NumberOfLines; i++)
477     //coloca os pontos
478     {
479         for(int j=0; j<NumberOfColumns; j++)
480         {
481             if(CellBox[i][j]->F==0)
482             {
483                 FileOut<<"10␣10␣10"<<endl;
484             }
485             else
486             {
487                 if(CellBox[i][j]->F>0)
488                 {
489                     if(CellBox[i][j]->F<Accurate*1)
490                     {
491                         FileOut<<"9␣10␣10"<<endl;
492                     }
493                     else
494                     {
495                     if(CellBox[i][j]->F<Accurate*2)
496                     {
497                     FileOut<<"8␣10␣10"<<endl;
498                     }
```

```
499                 else
500                 {
501                 if(CellBox[i][j]->F<Accurate*3)
502                 {
503                 FileOut<<"7␣10␣10"<<endl;
504                 }
505                 else
506                 {
507                 if(CellBox[i][j]->F<Accurate*4)
508                 {
509                 FileOut<<"6␣10␣10"<<endl;
510                 }
511                 else
512                 {
513                 if(CellBox[i][j]->F<Accurate*5)
514                 {
515                 FileOut<<"5␣10␣10"<<endl;
516                 }
517                 else
518                 {
519                 if(CellBox[i][j]->F<Accurate*6)
520                 {
521                 FileOut<<"4␣10␣10"<<endl;
522                 }
523                 else
524                 {
525                 if(CellBox[i][j]->F<Accurate*7)
526                 {
527                 FileOut<<"3␣10␣10"<<endl;
528                 }
529                 else
530                 {
531                 if(CellBox[i][j]->F<Accurate*8)
532                 {
533                 FileOut<<"2␣10␣10"<<endl;
```

```
534                     }
535                     else
536                     {
537                     if(CellBox[i][j]->F<Accurate*9)
538                     {
539                     FileOut<<"1 10 10"<<endl;
540                     }
541                     else
542                     {
543                     FileOut<<"0 10 10"<<endl;
544                     }
545                     }
546                     }
547                     }
548                     }
549                     }
550                     }
551                     }
552                     }
553                 }
554                 else
555                 {
556                     if(CellBox[i][j]->F>=Accurate*1)
557                     {
558                     FileOut<<"10 10 9"<<endl;
559                     }
560                     else
561                     {
562                     if(CellBox[i][j]->F>=Accurate*2)
563                     {
564                     FileOut<<"10 10 8"<<endl;
565                     }
566                     else
567                     {
568                     if(CellBox[i][j]->F>=Accurate*3)
```

```
569                 {
570                 FileOut<<"10␣10␣7"<<endl;
571                 }
572                 else
573                 {
574                 if(CellBox[i][j]->F>=Accurate*4)
575                 {
576                 FileOut<<"10␣10␣6"<<endl;
577                 }
578                 else
579                 {
580                 if(CellBox[i][j]->F>=Accurate*5)
581                 {
582                 FileOut<<"10␣10␣5"<<endl;
583                 }
584                 else
585                 {
586                 if(CellBox[i][j]->F>=Accurate*6)
587                 {
588                 FileOut<<"10␣10␣4"<<endl;
589                 }
590                 else
591                 {
592                 if(CellBox[i][j]->F>=Accurate*7)
593                 {
594                 FileOut<<"10␣10␣3"<<endl;
595                 }
596                 else
597                 {
598                 if(CellBox[i][j]->F>=Accurate*8)
599                 {
600                 FileOut<<"10␣10␣2"<<endl;
601                 }
602                 else
603                 {
```

```
604                         if(CellBox[i][j]->F>=Accurate*9)
605                         {
606                         FileOut<<"10 10 1"<<endl;
607                         }
608                         else
609                         {
610                         FileOut<<"10 10 0"<<endl;
611                         }
612                         }
613                         }
614                         }
615                         }
616                         }
617                         }
618                         }
619                         }
620                     }
621                 }
622             }
623         }
624         FileOut.close();
625 };
```

C.4 Classe *WaveSim2D*

C.4.1 WaveSim2D.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34 #ifndef WAVESIM2D_H
35 #define WAVESIM2D_H
36
37 #include<iostream>
38 #include "CellularAutomata2D.h"
39 #include "Lattice2D.h"
40
41
42 using namespace std;
43
44
45 class WaveSim2D
46 {
47     protected:
48     //////////////////
49     //atributos ////////
50     //////////////////
51
52     int NumberOfLines;
53         //numero de linhas da malha
54     int NumberOfColumns;
55         //numero de colunas da malha
56     int InputLine;
57         //linha da coordenada da excitacao
58     int InputColumn;
59         //coluna da coordenada da exitacao
60     bool Frequency;
61         //true caso a excitacao ocorra todo passo
62     Lattice2D *Lattice;
63         //malha onde ocorre a propagacao
64     //////////////////
65     //metodos //////////
66     //////////////////
67
```

```
68      virtual void leftBorder(Lattice2D *Result);
69      virtual void rightBorder(Lattice2D *Result);
70      virtual void topBorder(Lattice2D *Result);
71      virtual void bottomBorder(Lattice2D *Result);
72      virtual void middle(Lattice2D *Result);
73      virtual void corners(Lattice2D *Result);
74          //Funcoes para percorrer a malha aplicando as
75          //matrizes coerentes a cada situacao:
76          //livre com bordas reflexivas
77
78      virtual void setInput();
79          //coloca a excitacao
80      virtual void getValues(double &f0,
81          double &f1, double &f2,
82          double &f3, double &f4, int Line, int Column);
83          //coloca os valores
84          //de fis nos enderecos forncecidos
85      public:
86      ////////////////////
87      //Construtores /////
88      ////////////////////
89      WaveSim2D();
90          //erro: nao determinou o tamanho da malha
91      WaveSim2D(int LatticeLines,
92          int LatticeColumns, int InputLin,
93          int InputCol, bool type);
94          //Cria a situacao para simular com o
95          //numero de linhas NumLin e o numero
96          //de colunas NumCol
97
98      ~WaveSim2D(); //libera a memoria alocada
99      ////////////////////
100     //metodos///////////
101     ////////////////////
102         virtual void generate(int NumberOfSteps);
```

```
103         //Avanca a malha NumberOfSteps passos
104         virtual void generate(int NumberOfSteps,
105         char *DataFileBaseName); //coloca nos arquivos
106                                  //os dados da simulacao
107         virtual void generate(int NumberOfSteps,
108         double Accurate, char *AnimationName);
109         //Avanca a malha enquanto
110         //escreve imagems de cada passo
111         virtual void print();
112         //Imprime o valor da Lattice em um determinado site
113          //usado para debug
114         virtual void printToFile(double Accurate);
115         //imprime em um arquivo padrao
116         virtual void printToFile(double Accurate,
117         char *FileName);
118         //imprime para  para uma
119         arquivo  de nome FileName
120         virtual void printRealDataToFile(char *FileName);
121         //imprime os valores de F no arquivo
122         virtual double* returnSampleLine(int SampleFromLine);
123         //retorna um vetor de amostra da Linha indicada
124
125 };
126 #endif
```

C.4.2 WaveSim2D.cpp

```
1
2 // CAPropag2D: 2D Wave and Diffusion simulator
3  //using Cellular Automata -*- C++ -*-
4
5 // Copyright (C) 2003 Heitor Honda Federico.
6 //
7 //This file is part of the CAPropag2D library
8 //
```

```
9  // CAPropag2D is free
10 // software; you can redistribute it and/or modify it
11 //under the  terms of the GNU General Public License
12 //as published by the Free Software Foundation;
13 //either version 2, or (at your option) any later
14 //version.
15
16 // CAPropag2D  is distributed in the hope that it will
17 //be useful, but WITHOUT ANY WARRANTY; without
18 //even the implied warranty of MERCHANTABILITY or
19 //FITNESS FOR A PARTICULAR PURPOSE.  See the
20 // GNU General Public License for more details.
21
22 // You should have received a copy of the GNU General
23 //Public License along with this library; see the
24 //file COPYING.  If not, write to the Free Software
25 //Foundation, 59 Temple Place - Suite 330,
26 //Boston, MA 02111-1307, USA.
27
28
29 //Heitor Honda Federico
30 //sardaukar@uol.com.br
31 //Rua Saint Tropez 565, Jardim Mediterraneo
32 //Cotia, Sao Paulo Brazil
33 //CEP:06708-700
34
35
36 #include "WaveSim2D.h"
37
38 //////////////////////////////////////////////////
39 //Construtor inicializa a grade e recebe as coordenadas
40 //Destrutor desaloca a grade e zera atributos relevantes
41 //////////////////////////////////////////////////
42 WaveSim2D::WaveSim2D()
43 {
```

```
44        cerr<<"Entre com dimensoes"<<endl;
45        exit(1);
46    };
47
48    WaveSim2D::WaveSim2D(int LatticeLines, int LatticeColumns,
49    int InputLin, int InputCol, bool type)
50    {
51        NumberOfLines = LatticeLines;
52        NumberOfColumns = LatticeColumns;
53        assert(NumberOfLines>2 && NumberOfColumns>2);
54            //verifica se existe pelo menos uma casa
55            //com menos do que 3, nao tem sentido a
56            //propagacao
57        InputLine=InputLin;
58        InputColumn=InputCol;
59        assert(InputLine>=0 && InputLine<NumberOfLines &&
60        InputColumn>0 && InputColumn<NumberOfColumns);
61            //coerencia do input dentro da grade
62        Lattice = new Lattice2D(NumberOfLines, NumberOfColumns);
63            //inicializa a grade com zeros nos atributos
64            //dos sites
65        Frequency=type;
66    };
67    WaveSim2D::~WaveSim2D()
68    {
69        assert(Lattice!=NULL);
70            //verifica se existe algo a ser apagado
71        delete Lattice;
72        Frequency=false;
73        NumberOfLines = 0;
74        NumberOfColumns = 0;
75        InputLine=0;
76        InputColumn=0;
77    };
78
```

```
79 //////////////////////////////////////////////////////
80 //Avanca o numero de passos requeridos pelo áusurio//
81 //////////////////////////////////////////////////////
82 void WaveSim2D::generate(int NumberOfSteps,
83 char *DataFileBaseName)
84 {
85     Lattice2D *aux;
86     //ponteiro auxiliar para inverter grids
87     Lattice2D *Result = new
88     Lattice2D(NumberOfLines, NumberOfColumns);
89     //grade auxiliar para armazenar resultados parciais
90     bool Freq = true;
91     //faz o controle de excitacoes
92     caso excitado em frequencia
93
94     for(int i=0;i<NumberOfSteps;i++)
95     {   char FileName[80];
96         char Number[80];
97         sprintf(Number, "%d", i);
98         strcpy(FileName, DataFileBaseName);
99         strcat(FileName, ".");
100        strcat(FileName,Number);
101        //////////////generate//////////////////
102        if(Freq)
103        {
104            setInput();
105            if(!Frequency)
106            {
107                Freq=false;
108            }
109        }
110        Result->inputZero();
111        //zera a grade Result
112        leftBorder(Result);
113        //Funcoes para percorrer
```

```
114             //a malha aplicando as
115             rightBorder(Result);
116             //matrizes coerentes a cada situacao:
117             topBorder(Result);
118             //livre com bordas reflexivas
119             bottomBorder(Result);    //
120             middle(Result);          //
121             //corners(Result);
122             //Nao afetam  o problema proposto
123             aux=Lattice;
124             Lattice=Result;
125             Result=aux;
126             //   cout<<"ok: "<<i<<endl;
127             //////////////////////////////////
128             printRealDataToFile(FileName);
129         }
130         delete Result; //destroi a matriz auxiliat
131         aux=NULL;
132         //seta o ponteiro auxiliar para NULL
133
134 };
135 //gera uma animacao
136 void WaveSim2D::
137 generate(int NumberOfSteps,
138 double Accurate, char *AnimationName)
139 {
140         Lattice2D *aux;
141         //ponteiro auxiliar para inverter grids
142         Lattice2D *Result = new
143         Lattice2D(NumberOfLines, NumberOfColumns);
144         //grade auxiliar para
145         //armazenar resultados parciais
146         bool Freq = true;
147         //faz o controle de excitacoes
148         //caso excitado em frequencia
```

```
149
150     for(int i=0;i<NumberOfSteps;i++)
151     {   char AniName[80];
152         char Number[80];
153         sprintf(Number, "%d", i);
154         strcpy(AniName, AnimationName);
155         strcat(AniName, ".");
156         strcat(AniName,Number);
157         ////// generate ////////////////////
158         if(Freq)
159         {
160             setInput();
161             if(!Frequency)
162             {
163                 Freq=false;
164             }
165         }
166         Result->inputZero();
167         //zera a grade Result
168         leftBorder(Result);
169         //Funcoes para percorrer
170         //a malha aplicando as
171         rightBorder(Result);
172         //matrizes coerentes a cada situacao:
173         topBorder(Result);
174         //livre com bordas reflexivas
175         bottomBorder(Result);    //
176         middle(Result);          //
177         //corners(Result);
178         //Nao afetam  o problema proposto
179         aux=Lattice;
180         Lattice=Result;
181         Result=aux;
182         //  cout<<"ok: "<<i<<endl;
183         ////////////////////////////////////
```

```
184             printToFile(Accurate , AniName);
185         }
186         delete Result; //destroi a matriz auxiliat
187         aux=NULL;    //seta o ponteiro auxiliar para NULL
188
189 };
190
191 void WaveSim2D::generate(int NumberOfSteps)
192 {
193         Lattice2D *aux;
194         //ponteiro auxiliar para inverter grids
195         Lattice2D *Result = new Lattice2D(NumberOfLines,
196             NumberOfColumns);
197             //grade auxiliar para
198             //armazenar resultados parciais
199         bool Freq = true;
200             //faz o controle de excitacoes
201             //caso excitado em frequencia
202             //setInput();
203         for(int i=0; i<NumberOfSteps; i++)
204         {
205             if(Freq)
206             {
207                 setInput();
208                 if(!Frequency)
209                 {
210                     Freq=false;
211                 }
212             }
213             Result->inputZero();//zera a grade Result
214             leftBorder(Result); /Funcoes para percorrer
215             rightBorder(Result);/a malha aplicando as matrizes
216             topBorder(Result);  //coerentes a cada situacao
217             bottomBorder(Result);//livre com bordas reflexivas
218             middle(Result);      //
```

```
219             //corners(Result);  //Nao afetam  o problema proposto
220             aux=Lattice;
221             Lattice=Result;
222             Result=aux;
223         //  cout<<"ok: "<<i<<endl;
224         }
225         delete Result;
226             //destroi a matriz auxiliat
227         aux=NULL;
228             //seta o ponteiro auxiliar para NULL
229 };
230 //private setInput: coloca a excitacao inicial
231 void WaveSim2D::setInput()
232 {
233 //Lattice->initCell(0, 1, 0, 0 ,0, InputLine, InputColumn);
234 Lattice->plusCell(0, 1, 0, 0 ,0, InputLine, InputColumn+1);
235 Lattice->plusCell(0, 0, 0, 1 ,0, InputLine, InputColumn-1);
236 Lattice->plusCell(0, 0, 1, 0 ,0, InputLine-1, InputColumn);
237 Lattice->plusCell(0, 0, 0, 0 ,1, InputLine+1, InputColumn);
238
239
240 };
241 //private: leftborder :executa a
242 //transicao na borda esquerda
243 void WaveSim2D::leftBorder(Lattice2D *Result)
244 {
245     double f0, f1, f2, f3, f4;
246             //variaveis temporarias usadas para calculo
247     for(int i=1;i<NumberOfLines-1; i++)
248             //percorre a borda esquerda
249     {
250             getValues(f0, f1, f2, f3, f4, i, 0);
251             Result->plusCell(0, -f3, 0, 0, 0, i, 1);
252     }
253
```

```
254 };
255 //private: rightborder :
256 //executa a transicao na borda direita
257 void WaveSim2D::rightBorder(Lattice2D *Result)
258 {
259     double f0, f1, f2, f3, f4;
260         //variaveis temporarias usadas para calculo
261     for(int i=1;i<NumberOfLines-1; i++)
262         //percorre a borda esquerda
263     {
264         getValues(f0, f1, f2, f3, f4,
265         i, NumberOfColumns-1);
266         Result->plusCell(0, 0, 0,
267         -f1, 0, i, (NumberOfColumns-2));
268     }
269 };
270 //private: topBorder :executa a
271 //transicao na borda superior
272 void WaveSim2D::topBorder(Lattice2D *Result)
273 {
274     double f0, f1, f2, f3, f4;
275         //variaveis temporarias usadas para calculo
276     for(int j=1;j<NumberOfColumns-1; j++)
277         //percorre a borda esquerda
278     {
279         getValues(f0, f1, f2, f3, f4, 0, j);
280         Result->plusCell(0, 0, 0, 0, -f2, 1, j);
281     }
282 };
283 //private: bottomborder :executa a
284 //transicao na borda inferior
285 void WaveSim2D::bottomBorder(Lattice2D *Result)
286 {
287     double f0, f1, f2, f3, f4;
288         //variaveis temporarias usadas para calculo
```

```
289     for(int j=1;j<NumberOfColumns-1; j++)
290         //percorre a borda esquerda
291     {
292         getValues(f0, f1, f2, f3, f4,
293         NumberOfLines-1, j);
294         Result->plusCell(0, 0, -f4,
295         0, 0, NumberOfLines-2, j);
296     }
297
298 };
299 //private: middle :executa a transicao na parte central
300 void WaveSim2D::middle(Lattice2D *Result)
301 {
302     double f0, f1, f2, f3, f4;
303         //variaveis temporarias usadas para calculo
304     double F1, F2, F3, F4;
305     for(int i=1; i<NumberOfLines-1; i++)
306     {
307         for(int j=1;j<NumberOfColumns-1;j++)
308         {
309             getValues(f0, f1, f2, f3, f4, i, j);
310             F1=0.5*(f1+f2-f3+f4); //
311             F2=0.5*(f1+f2+f3-f4); //Multiplicacao de
312             F3=0.5*(-f1+f2+f3+f4); //Matrizes na raca
313             F4=0.5*(f1-f2+f3+f4); //
314
315         Result->plusCell(0, F1, 0, 0, 0, i, (j+1));
316         Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
317         Result->plusCell(0, 0, 0, F3, 0, i, j-1);
318         Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
319
320         }
321     }
322
323 };
```

```
324 //private: corners :executa a transicao nos cantos
325 void WaveSim2D::corners(Lattice2D *Result)
326 {
327     double f0, f1, f2, f3, f4;
328         //variaveis temporarias usadas para calculo
329     getValues(f0, f1, f2, f3, f4, 0, 0);
330     Result->plusCell(0, -f3, 0, 0, 0, 0, 1);
331     Result->plusCell(0, 0, 0, 0, -f2, 1, 0);
332
333     getValues(f0, f1, f2, f3, f4, (NumberOfLines-1), 0);
334     Result->plusCell(0, -f3, 0,
335     0, 0, (NumberOfLines-1), 1);
336     Result->plusCell(0, 0, -f4,
337     0, 0, (NumberOfLines-2), 0);
338
339     getValues(f0, f1, f2, f3, f4,
340     (NumberOfLines-1), (NumberOfColumns-1));
341     Result->plusCell(0, 0, 0, -f1, 0, (NumberOfLines-1),
342         (NumberOfColumns-2));
343     Result->plusCell(0, -f4, 0, 0, 0, (NumberOfLines-2),
344         (NumberOfColumns-1));
345
346     getValues(f0, f1, f2, f3, f4, 0, (NumberOfColumns-1));
347     Result->plusCell(0, 0, 0, -f1, 0, 0,
348         (NumberOfColumns-2));
349     Result->plusCell(0, 0, 0, 0, -f2, 1,
350         (NumberOfColumns-1));
351 };
352 //private:getValuesFis:
353 //coloca os valores de fis nos enderecos forncecidos
354 void WaveSim2D::getValues(double &f0, double &f1, double &f2,
355 double &f3, double &f4, int Line, int Column)
356 {
357     f0=Lattice->returnCellF0(Line, Column);
358     f1=Lattice->returnCellF1(Line, Column);
```

```
359         f2=Lattice->returnCellF2(Line, Column);
360         f3=Lattice->returnCellF3(Line, Column);
361         f4=Lattice->returnCellF4(Line, Column);
362     };
363
364     ////////////////////////////////////////////////////////////
365     //public: print() : imprime os valores fe Fi
366     //para todos os sites/
367     ////////////////////////////////////////////////////////////
368     void WaveSim2D::print()
369     {
370         double f0, f1, f2, f3, f4;
371             //variaveis temporarias usadas para calculo
372         cout<<"\n"<<"Lattice:"<<endl;
373         cout<<"i\tj\tf1\tf2\tf3\tf4"<<endl;
374         for(int i=0; i<NumberOfLines;i++)
375         {
376             for(int j=0; j<NumberOfColumns;j++)
377             {
378                 getValues(f0, f1, f2, f3, f4, i, j);
379                 cout<<i<<"\t"<<j<<"\t"<<f1<<"\t"
380                     <<f2<<"\t"<<f3<<"\t"<<f4<<endl;
381             }
382         }
383     };
384
385     ////////////////////////////////////////////////////////
386     //public:  printToFile imprime em um arquivo padrao
387     ////////////////////////////////////////////////////////
388     void WaveSim2D::printToFile(double Accurate)
389     {
390         assert(Accurate>0&&Accurate<=0.1);
391         Lattice->printToFile(Accurate);
392     }
393     void WaveSim2D::printToFile
```

```
394 (double Accurate, char *FileName)
395 {
396     assert(Accurate>0&&Accurate<=0.1);
397     Lattice->printToFile(Accurate, FileName);
398 }
399 void WaveSim2D::printRealDataToFile(char *FileName)
400 {
401     Lattice->printRealDataToFile(FileName);
402 }
403 ////////////////////////////////////////////////////
404 //public returnSampleLine,
405 //retorna um ponteiro para um vetor na linha//
406 //com os valores de F                          //
407 ////////////////////////////////////////////////
408 double* WaveSim2D::returnSampleLine(int SampleFromLine)
409 {
410     double *LocalSample;
411     //armazena a amostra dentro deste escopo
412     double f0, f1, f2, f3, f4;
413     //variaveis locais para auxilio de calculo
414     assert(SampleFromLine>0 && SampleFromLine<NumberOfLines);
415     //coerencia dos dados
416     LocalSample = new double[NumberOfColumns];
417     assert(LocalSample!=NULL); //conseguiu alocar o vetor;
418     for(int i=0;i<NumberOfColumns;i++)
419     {
420         getValues(f0, f1, f2, f3, f4, SampleFromLine, i);
421         LocalSample[i]=f0+f1+f2+f3+f4;
422         //equivalente a funcao calcF
423     }
424     return(LocalSample);
425 };
```

C.5 Classe *DiffusionSim2D*

C.5.1 DiffusionSim2D.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34 #ifndef DIFFUSIONSIM2D_H
35 #define DIFFUSIONSIM2D_H
36
37 #include "Lattice2D.h"
38
39
40 using namespace std;
41
42
43 class DiffusionSim2D
44 {
45     protected:
46         //////////////////
47         //atributos/////////
48         //////////////////
49         int NumberOfLines;
50         //numero de linhas da malha
51         int NumberOfColumns;
52         //numero de colunas da malha
53         int InputLine;
54         //linha da coordenada da excitacao
55         int InputColumn;
56         //coluna da coordenada da exitacao
57         double P0;
58         //Valores das probabilidades de difusao
59         double P1;  //P0 reflexao , P1 horario
60         double P2;  //P2 antihorario
61         double P3;  //P3 "passar direto"
62         bool Frequency;
63         //true caso a excitacao ocorra todo passo
64         Lattice2D *Lattice;
65         //malha onde ocorre a propagacao
66         //////////////////
67         //metodos//////////
```

```
68          ////////////////////
69
70          virtual void leftBorder(Lattice2D *Result);
71          //Funcoes para percorrer a malha aplicando as
72          virtual void rightBorder(Lattice2D *Result);
73          //matrizes coerentes a cada situacao:
74          virtual void topBorder(Lattice2D *Result);
75          //livre com bordas reflexivas
76          virtual void bottomBorder(Lattice2D *Result);
77          virtual void middle(Lattice2D *Result);
78          virtual void corners(Lattice2D *Result);
79
80          virtual void setInput();
81          //coloca a excitacao
82          virtual void getValues(double &f0, double &f1,
83          double &f2, double &f3, double &f4,
84          int Line, int Column);
85          //coloca os valores de fis nos
86          //enderecos forncecidos
87      public:
88          ////////////////////
89          //Construtores /////
90          ////////////////////
91          DiffusionSim2D();
92          //erro: nao determinou o tamanho da malha
93          DiffusionSim2D(int LatticeLines,
94          int LatticeColumns, int InputLin, int InputCol,
95          bool type, double PD0, double PD1, double PD2,
96          double PD3);/
97          //Cria a situacao para simular com o
98          //numero de linhas NumLin e o numero
99          //de colunas NumCol
100
101         ~DiffusionSim2D();
102         //libera a memoria alocada
```

```
103         //////////////////
104         //metodos//////////
105         //////////////////
106         virtual void generate(int NumberOfSteps);
107         //Avanca a malha NumberOfSteps passos
108         virtual void generate(int NumberOfSteps,
109         char *DataFileBaseName); //coloca nos arquivos
110                                  //os dados da simulacao
111         virtual void generate(int NumberOfSteps,
112         double Accurate, char *AnimationName);
113         //Avanca a malha enquanto escreve
114         //imagems de cada passo
115         virtual void print();
116         //Imprime o valor da Lattice
117         //em um determinado site
118         //usado para debug
119         virtual void printToFile(double Accurate);
120         //imprime em um arquivo padrao
121         virtual void printToFile
122             (double Accurate, char *FileName);
123         //imprime para  para uma
124         //arquivo  de nome FileName
125         virtual void printRealDataToFile(char *FileName);
126         //imprime os valores de F no arquivo
127 };
128 #endif
```

C.5.2 DiffusionSim2D.cpp

```
1 // CAPropag2D: 2D Wave and Diffusion simulator
2  //using Cellular Automata -*- C++ -*-
3
4 // Copyright (C) 2003 Heitor Honda Federico.
5 //
6 //This file is part of the CAPropag2D library
```

```
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33 #include "DiffusionSim2D.h"
34
35
36 //////////////////////////////////////////////////////////////
37 //Construtor inicializa a grade e recebe as coordenadas//////
38 //Destrutor desaloca a grade e zera atributos relevantes /////
39
40 DiffusionSim2D::DiffusionSim2D()
41 {
```

```
42      cerr <<"Entre com dimensoes"<<endl;
43      exit(1);
44  };
45
46  DiffusionSim2D::DiffusionSim2D(int LatticeLines,
47  int LatticeColumns, int InputLin, int InputCol,
48  bool type, double PD0, double PD1,
49  double PD2, double PD3)
50  {
51      NumberOfLines = LatticeLines;
52      NumberOfColumns = LatticeColumns;
53      assert(NumberOfLines>2 && NumberOfColumns>2);
54      //verifica se existe pelo menos uma casa
55      //com menos do que 3, nao tem sentido a
56      //propagacao
57      assert((PD0+PD1+PD2+PD3)==1);
58      //coerencia das probabilidades
59      P0=PD0;
60      P1=PD1;
61      P2=PD2;
62      P3=PD3;
63      InputLine=InputLin;
64      InputColumn=InputCol;
65      assert(InputLine>=0 && InputLine<NumberOfLines &&
66      InputColumn>0 && InputColumn<NumberOfColumns);
67      //coerencia do input dentro da grade
68      Lattice = new Lattice2D(NumberOfLines,
69      NumberOfColumns); //inicializa a grade com zeros
70                        //nos atributos
71                       //dos sites
72      Frequency=type;
73  };
74  DiffusionSim2D::~DiffusionSim2D()
75  {
76      assert(Lattice!=NULL);
```

```
77      //verifica se existe algo a ser apagado
78      delete Lattice;
79      Frequency=false;
80      NumberOfLines = 0;
81      NumberOfColumns = 0;
82      InputLine=0;
83      InputColumn=0;
84  };
85
86  ////////////////////////////////////////////////////
87  //Avanca o numero de passos requeridos pelo áusurio
88  ////////////////////////////////////////////////////
89  //coloca os dados em arquivos
90  //iniciados com o nome DataFileBaseName
91  void DiffusionSim2D::
92  generate(int NumberOfSteps, char *DataFileBaseName)
93  {
94      Lattice2D *aux;
95      //ponteiro auxiliar para inverter grids
96      Lattice2D *Result =
97      new Lattice2D(NumberOfLines, NumberOfColumns);
98      //grade auxiliar para armazenar resultados parciais
99      bool Freq = true;
100     //faz o controle de excitacoes caso
101     //excitado em frequencia
102
103     for(int i=0;i<NumberOfSteps;i++)
104     {   char FileName[80];
105         char Number[80];
106         sprintf(Number, "%d", i);
107         strcpy(FileName, DataFileBaseName);
108         strcat(FileName, ".");
109         strcat(FileName,Number);
110         //////////////generate//////////////////
111         if(Freq)
```

```
112         {
113             setInput();
114             if(!Frequency)
115             {
116                 Freq=false;
117             }
118         }
119         Result->inputZero();
120         //zera a grade Result
121         leftBorder(Result);
122         //Funcoes para percorrer a malha aplicando as
123         rightBorder(Result);
124         //matrizes coerentes a cada situacao:
125         topBorder(Result);
126         //livre com bordas reflexivas
127         bottomBorder(Result);    //
128         middle(Result);          //
129         //corners(Result);
130         //Nao afetam  o problema proposto
131         aux=Lattice;
132         Lattice=Result;
133         Result=aux;
134         //  cout<<"ok: "<<i<<endl;
135         //////////////////////////////////
136         printRealDataToFile(FileName);
137     }
138     delete Result; //destroi a matriz auxiliat
139     aux=NULL;//seta o ponteiro auxiliar para NULL
140
141 };
142 //gera uma animacao
143 void DiffusionSim2D::
144 generate(int NumberOfSteps, double Accurate,
145 char *AnimationName)
146 {
```

```
147     Lattice2D *aux;
148     //ponteiro auxiliar para inverter grids
149     Lattice2D *Result =
150     new Lattice2D(NumberOfLines, NumberOfColumns);
151     //grade auxiliar para armazenar
152     //resultados parciais
153     bool Freq = true;
154     //faz o controle de excitacoes
155     //caso excitado em frequencia
156
157     for(int i=0;i<NumberOfSteps;i++)
158     {    char AniName[80];
159          char Number[80];
160          sprintf(Number, "%d", i);
161          strcpy(AniName, AnimationName);
162          strcat(AniName, ".");
163          strcat(AniName,Number);
164          //////////generate//////////////////
165          if(Freq)
166          {
167               setInput();
168               if(!Frequency)
169               {
170                    Freq=false;
171               }
172          }
173          Result->inputZero();
174          //zera a grade Result
175          leftBorder(Result);
176          //Funcoes para percorrer
177          //a malha aplicando as
178          rightBorder(Result);
179          //matrizes coerentes a cada situacao:
180          topBorder(Result);
181          //livre com bordas reflexivas
```

```
182             bottomBorder(Result);     //
183             middle(Result);           //
184             //corners(Result);
185             //Nao afetam  o problema proposto
186             aux=Lattice;
187             Lattice=Result;
188             Result=aux;
189             //   cout<<"ok: "<<i<<endl;
190             ////////////////////////////////////
191             printToFile(Accurate, AniName);
192         }
193         delete Result; //destroi a matriz auxiliat
194         aux=NULL;    //seta o ponteiro auxiliar para NULL
195
196 };
197 void DiffusionSim2D::generate(int NumberOfSteps)
198 {
199     Lattice2D *aux;
200     //ponteiro auxiliar para inverter grids
201     Lattice2D *Result =
202     new Lattice2D(NumberOfLines, NumberOfColumns);
203     //grade auxiliar para armazenar
204     //resultados parciais
205     bool Freq = true;
206     //faz o controle de excitacoes
207     //caso excitado em frequencia
208     //setInput();
209     for(int i=0; i<NumberOfSteps; i++)
210     {
211         if(Freq)
212         {
213             setInput();
214             if(!Frequency)
215             {
216                 Freq=false;
```

```
217                 }
218             }
219             Result->inputZero();
220             //zera a grade Result
221             leftBorder(Result);
222             //Funcoes para percorrer
223             //a malha aplicando as
224             rightBorder(Result);
225             //matrizes coerentes a cada situacao:
226             topBorder(Result);
227             //livre com bordas reflexivas
228             bottomBorder(Result);     //
229             middle(Result);           //
230             //corners(Result);
231             //Nao afetam  o problema proposto
232             aux=Lattice;
233             Lattice=Result;
234             Result=aux;
235             //   cout<<"ok: "<<i<<endl;
236         }
237         delete Result; //destroi a matriz auxiliat
238         aux=NULL;    //seta o ponteiro auxiliar para NULL
239 };
240
241 //private setInput: coloca a excitacao inicial
242 void DiffusionSim2D::setInput()
243 {
244         //Lattice->initCell(0, 1, 0, 0 ,0,
245         InputLine, InputColumn);
246         Lattice->plusCell(0, 0.25, 0.25,
247         0.25 ,0.25, InputLine, InputColumn);
248         //entrada pontual igual a 1
249 };
250 //private: leftborder :
251 executa a transicao na borda esquerda
```

```
252 void DiffusionSim2D::
253 leftBorder(Lattice2D *Result)
254 {
255     double f0, f1, f2, f3, f4;
256     //variaveis temporarias usadas para calculo
257     for(int i=1;i<NumberOfLines-1; i++)
258     //percorre a borda esquerda
259     {
260         getValues(f0, f1, f2, f3, f4, i, 0);
261         Result->plusCell(0, f3, 0, 0, 0, i, 1);
262     }
263
264 };
265 //private: rightborder :
266 //executa a transicao na borda direita
267 void DiffusionSim2D::rightBorder(Lattice2D *Result)
268 {
269     double f0, f1, f2, f3, f4;
270     //variaveis temporarias usadas para calculo
271     for(int i=1;i<NumberOfLines-1; i++)
272     //percorre a borda esquerda
273     {
274         getValues(f0, f1, f2, f3,
275         f4, i, NumberOfColumns-1);
276         Result->plusCell(0, 0, 0,
277         f1, 0, i, (NumberOfColumns-2));
278     }
279 };
280 //private: topBorder :executa a
281 //transicao na borda superior
282 void DiffusionSim2D::topBorder(Lattice2D *Result)
283 {
284     double f0, f1, f2, f3, f4;
285     //variaveis temporarias usadas para calculo
286     for(int j=1;j<NumberOfColumns-1; j++)
```

```
287     //percorre a borda esquerda
288     {
289         getValues(f0, f1, f2, f3, f4, 0, j);
290         Result->plusCell(0, 0, 0, 0, f2, 1, j);
291     }
292 };
293 //private: bottomborder :executa a
294 //transicao na borda inferior
295 void DiffusionSim2D::bottomBorder(Lattice2D *Result)
296 {
297     double f0, f1, f2, f3, f4;
298     //variaveis temporarias usadas para calculo
299     for(int j=1;j<NumberOfColumns-1; j++)
300     //percorre a borda esquerda
301     {
302         getValues(f0, f1, f2, f3,
303         f4, NumberOfLines-1, j);
304         Result->plusCell(0, 0, f4, 0,
305         0, NumberOfLines-2, j);
306     }
307
308 };
309 //private: middle :executa a transicao na parte central
310 void DiffusionSim2D::middle(Lattice2D *Result)
311 {
312     double f0, f1, f2, f3, f4;
313     //variaveis temporarias usadas para calculo
314     double F1, F2, F3, F4;
315     for(int i=1; i<NumberOfLines-1; i++)
316     {
317         for(int j=1;j<NumberOfColumns-1;j++)
318         {
319             getValues(f0, f1, f2, f3, f4, i, j);
320             F1=P0*f1+P1*f2+P2*f3+P3*f4; //
321             F2=P3*f1+P0*f2+P1*f3+P2*f4; //
```

```
322                 //Multiplicacao de matrizes na raca
323                 F3=P2*f1+P3*f2+P0*f3+P1*f4; //
324                 F4=P1*f1+P2*f2+P3*f3+P0*f4; //
325
326
327                 Result->plusCell(0, F1, 0, 0, 0, i, (j+1));//
328                 Result->plusCell(0, 0, 0, 0, F4, (i+1), j);//
329                 Result->plusCell(0, 0, 0, F3, 0, i, (j-1));//
330                 Result->plusCell(0, 0, F2, 0, 0, (i-1), j);//
331
332             }
333         }
334
335 };
336 //private: corners :executa a transicao nos cantos
337 void DiffusionSim2D::corners(Lattice2D *Result)
338 {
339     double f0, f1, f2, f3, f4;
340     //variaveis temporarias usadas para calculo
341     getValues(f0, f1, f2, f3, f4, 0, 0);
342     Result->plusCell(0, f3, 0, 0, 0, 0, 1);
343     Result->plusCell(0, 0, 0, 0, f2, 1, 0);
344
345     getValues(f0, f1, f2, f3, f4,
346     (NumberOfLines-1), 0);
347     Result->plusCell(0, f3, 0, 0, 0,
348     (NumberOfLines-1), 1);
349     Result->plusCell(0, 0, f4, 0, 0,
350     (NumberOfLines-2), 0);
351
352     getValues(f0, f1, f2, f3, f4, (NumberOfLines-1),
353     (NumberOfColumns-1));
354     Result->plusCell(0, 0, 0, f1, 0, (NumberOfLines-1),
355     (NumberOfColumns-2));
356     Result->plusCell(0, f4, 0, 0, 0, (NumberOfLines-2),
```

```
357         (NumberOfColumns-1));
358
359         getValues(f0, f1, f2, f3, f4, 0,
360         (NumberOfColumns-1));
361         Result->plusCell(0, 0, 0, f1, 0, 0,
362         (NumberOfColumns-2));
363         Result->plusCell(0, 0, 0, 0, f2, 1,
364         (NumberOfColumns-1));
365 };
366 //private:getValuesFis: coloca os valores de
367 //fis nos enderecos forncecidos
368 void DiffusionSim2D::getValues(double &f0,
369 double &f1, double &f2, double &f3, double &f4,
370 int Line, int Column)
371 {
372         f0=Lattice->returnCellF0(Line, Column);
373         f1=Lattice->returnCellF1(Line, Column);
374         f2=Lattice->returnCellF2(Line, Column);
375         f3=Lattice->returnCellF3(Line, Column);
376         f4=Lattice->returnCellF4(Line, Column);
377 };
378
379 //////////////////////////////////////////////////
380 //public: print() : imprime os
381 //valores fe Fi para todos os sites/////////
382 //////////////////////////////////////////////////
383 void DiffusionSim2D::print()
384 {
385         double f0, f1, f2, f3, f4;
386         //variaveis temporarias usadas para calculo
387         cout<<"\n"<<"Lattice:"<<endl;
388         cout<<"i\tj\tf1\tf2\tf3\tf4"<<endl;
389         for(int i=0; i<NumberOfLines;i++)
390         {
391             for(int j=0; j<NumberOfColumns;j++)
```

```
392         {
393             getValues(f0, f1, f2, f3, f4, i, j);
394             cout<<i<<"\t"<<j<<"\t"<<f1<<
395             "\t"<<f2<<"\t"<<f3<<"\t"<<f4<<endl;
396         }
397     }
398 };
399
400 //////////////////////////////////////////////////////
401 //public:  printToFile imprime em um arquivo padrao
402 //////////////////////////////////////////////////////
403 void DiffusionSim2D::printToFile(double Accurate)
404 {
405     assert(Accurate>0&&Accurate<=0.1);
406     Lattice->printToFile(Accurate);
407 }
408 void DiffusionSim2D::
409 printToFile(double Accurate, char *FileName)
410 {
411     assert(Accurate>0&&Accurate<=0.1);
412     Lattice->printToFile(Accurate, FileName);
413 }
414
415 void DiffusionSim2D::printRealDataToFile(char *FileName)
416 {
417     Lattice->printRealDataToFile(FileName);
418 }
```

C.6 Classe *WaveSim2DCustomBarrier*

C.6.1 WaveSim2DCustomBarrier.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34 #ifndef WAVESIM2DCUSTOMBARRIER_H
35 #define WAVESIM2DCUSTOMBARRIER_H
36
37 #include "WaveSim2D.h"
38
39 using namespace std;
40
41 class WaveSim2DCustomBarrier : public WaveSim2D
42 {
43     protected:
44         void setInput(); //excitacao em linha
45         void middle(Lattice2D *Result);
46     ////////////////////////////////////////////////
47         void setCircBarrier();
48         void circlePoints(int Line, int Column);
49         bool CircularBarrierTrue;
50         int Radius;
51         int CenterLine;
52         int CenterColumn;
53     ////////////////////////////////////////////////
54         void chooseLine();
55         void setLineUp(int Lin0,
56         int Col0, int Lin1, int Col1);
57         void setLineDown(int Lin0,
58         int Col0, int Lin1, int Col1);
59         bool LineBarrierTrue;
60         int Line0,Column0;
61         int Line1,Column1;
62     ////////////////////////////////////////////////
63     public:
64         ///////////////
65         //consturtores//
66         ///////////////
67         WaveSim2DCustomBarrier();
```

```
68          WaveSim2DCustomBarrier(int LatticeLines,
69          int LatticeColumns,int InputLin,
70          int InputCol, bool type);
71          ~WaveSim2DCustomBarrier();
72
73          ////////////////
74          //metodos///////
75          ////////////////
76          void setCircularBarrier(int CircularRadius,
77          int LineCenter, int ColumnCenter);
78          void setLineBarrier(int Lin0, int Col0,
79          int Lin1, int Col1);
80  };
81
82  #endif
```

C.6.2 WaveSim2DCustomBarrier.cpp

```
1   // CAPropag2D: 2D Wave and Diffusion simulator
2    //using Cellular Automata -*- C++ -*-
3
4   // Copyright (C) 2003 Heitor Honda Federico.
5   //
6   //This file is part of the CAPropag2D library
7   //
8   // CAPropag2D is free
9   // software; you can redistribute it and/or modify it
10  //under the  terms of the GNU General Public License
11  //as published by the Free Software Foundation;
12  //either version 2, or (at your option) any later
13  //version.
14
15  // CAPropag2D  is distributed in the hope that it will
16  //be useful, but WITHOUT ANY WARRANTY; without
17  //even the implied warranty of MERCHANTABILITY or
```

```
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34 #include "WaveSim2DCustomBarrier.h"
35 //////////////////////////////////////////////////
36 //Cosntrutores:fazem o mesmo que a classe pai///
37 //////////////////////////////////////////////////
38
39 WaveSim2DCustomBarrier ::
40 WaveSim2DCustomBarrier() : WaveSim2D()
41 {
42 };
43
44 WaveSim2DCustomBarrier::
45 WaveSim2DCustomBarrier(int LatticeLines,
46 int LatticeColumns, int InputLin, int InputCol,
47 bool type) : WaveSim2D(LatticeLines, LatticeColumns,
48 InputLin, InputCol, type)
49 {
50     CircularBarrierTrue=false;
51     LineBarrierTrue = false;
52 };
```

```
53 WaveSim2DCustomBarrier :: ~WaveSim2DCustomBarrier()
54 {
55 };
56 //////////////////////////////////////////////////
57 //middle confere se o site eh ou nao reflexivo /////
58 //////////////////////////////////////////////////
59 void WaveSim2DCustomBarrier::middle(Lattice2D *Result)
60 {
61     double f0, f1, f2, f3, f4;
62     //variaveis temporarias usadas para calculo
63     double F1, F2, F3, F4;
64     if(CircularBarrierTrue)
65     {
66         setCircBarrier();
67     }
68     if(LineBarrierTrue)
69     {
70         chooseLine();
71     }
72 /*
73     for(int i=1; i<NumberOfLines-1; i++)
74     {
75         for(int j=1;j<NumberOfColumns-1;j++)
76         {
77             if(Lattice->returnIsReflective(i, j))
78             {
79                 cout<<" 1";
80             }
81             else
82             {
83                 cout<<" 0";
84             }
85         }
86         cout<<endl;
87     }
```

```
88      int pause;
89      cin>>pause;
90  */
91
92      for(int i=1; i<NumberOfLines-1; i++)
93      {
94          for(int j=1;j<NumberOfColumns-1;j++)
95          {
96              getValues(f0, f1, f2, f3, f4, i, j);
97
98              if(Lattice->returnIsReflective(i, j))
99              {
100     Result->plusCell(0, -f3, 0, 0, 0, i, (j+1));
101     Result->plusCell(0, 0, 0, 0, -f2, (i+1), j);
102     Result->plusCell(0, 0, 0, -f1, 0, i, j-1);
103     Result->plusCell(0, 0, -f4, 0, 0, (i-1), j);
104             }
105             else //nao eh reflexivo
106             {
107                 F1=0.5*(f1+f2-f3+f4);    //
108                 F2=0.5*(f1+f2+f3-f4);    //
109                 Multiplicacao de matrizes na raca
110                 F3=0.5*(-f1+f2+f3+f4);   //
111                 F4=0.5*(f1-f2+f3+f4);    //
112
113         Result->plusCell(0, F1, 0, 0, 0, i, (j+1));
114         Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
115         Result->plusCell(0, 0, 0, F3, 0, i, j-1);
116         Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
117             }
118         }
119     }
120 };
121
122
```

```
123
124
125 //////////////////////////////////////////////
126 //setInput  neste caso coloca uma excitacao
127 em linha em//////
128 //Linha = NumberofLines//////////////////////
129 //////////////////////////////////////////////
130 void WaveSim2DCustomBarrier :: setInput()
131 {
132     for(int j=1;j<NumberOfColumns-1; j++)
133     {
134         Lattice->plusCell(0, 0, 0, 0,
135         -0.25, NumberOfLines-1, j);
136     }
137     /*
138     Lattice->plusCell(0, 1, 0,
139     0 ,0, InputLine , InputColumn+1);
140     Lattice->plusCell(0, 0,
141     0, 1 ,0, InputLine , InputColumn-1);
142     Lattice->plusCell(0, 0,
143     1, 0 ,0, InputLine-1, InputColumn);
144     Lattice->plusCell(0, 0,
145     0, 0 ,1, InputLine+1, InputColumn);
146     */
147
148 };
149 //////////////////////////////////////////
150 //setCircularBarrier//////////////////////
151 //////////////////////////////////////////
152 void WaveSim2DCustomBarrier ::
153 setCircularBarrier(int CircularRadius,
154 int LineCenter, int ColumnCenter)
155 {
156     CircularBarrierTrue=true;
157     Radius = CircularRadius;
```

```
158     CenterLine = LineCenter;
159     CenterColumn = ColumnCenter;
160     setCircBarrier();
161 };
162 void WaveSim2DCustomBarrier ::
163 setCircBarrier()
164 {
165     assert(CenterLine>Radius) ;
166     //garante que no maximo encosta
167     //na na borda linha 0
168     assert(CenterLine<NumberOfLines-Radius);
169     //garante que no maximo encosta na borda
170     assert(CenterColumn>Radius);
171     //o mesmo para colunas
172     assert(CenterColumn<NumberOfColumns-Radius);
173     //o mesmo para colunas
174 //////////////////////////////////
175     int x = 0;        //coluna
176     int y = Radius; //linha
177     double d = 5.0/4.0 - Radius;
178     circlePoints(y, x);
179
180     while(y > x)
181     {
182         if(d < 0)
183         {
184             d += 2.0*x+3.0;
185         }
186         else
187         {
188             d += 2.0*(x-y)+5.0;
189             y--;
190         }
191         x++;
192         circlePoints(y, x);
```

```
193         }
194 //////////////////////////////////
195         return;
196 };
197 //circlePoints determina os simetricos de um dado ponto
198 //em uma circunferencia na origem
199 void WaveSim2DCustomBarrier ::
200 circlePoints(int Line, int Column)
201 {
202         Lattice->setIsReflective
203         (CenterLine+Line, CenterColumn+Column, true);
204
205         Lattice->setIsReflective
206         (CenterLine+Column, CenterColumn+Line, true);
207
208         Lattice->setIsReflective
209         (CenterLine+Column, CenterColumn-Line, true);
210
211         Lattice->setIsReflective
212         (CenterLine+Line, CenterColumn-Column, true);
213
214         Lattice->setIsReflective
215         (CenterLine-Line, CenterColumn-Column, true);
216
217         Lattice->setIsReflective
218         (CenterLine-Column, CenterColumn-Line, true);
219
220         Lattice->setIsReflective
221         (CenterLine-Column, CenterColumn+Line, true);
222
223         Lattice->setIsReflective
224         (CenterLine-Line, CenterColumn+Column, true);
225
226 };
227 //////////////////////////////////////////////////
```

```
228 ///setLine Barrier: poe uma barreira em linha/
229 ////////////////////////////////////////////
230
231 void WaveSim2DCustomBarrier ::
232 setLineBarrier(int Lin0, int Col0, int Lin1, int Col1)
233 {
234     Line0=Lin0;
235     Column0=Col0;
236     Line1=Lin1;
237     Column1=Col1;
238     LineBarrierTrue=true;
239 };
240 void WaveSim2DCustomBarrier :: chooseLine()
241 {   int aux;
242     if(Column0<Column1)
243     {//garante ponto 0 a esquerda
244         aux=Column0;
245         Column0=Column1;
246         Column1=aux;
247         aux=Line0;
248         Line0=Line1;
249         Line1=Line0;
250     }
251     if(Column0<=Column1)
252     {//garante reta para cima
253         if(Column1-Column0>=Line1-Line0)
254         {//menos de 45 graus
255             setLineUp(Line0, Column0, Line1, Column1);
256         }
257         else
258         {//mais de 45 graus
259             setLineUp(Column0, Line0, Column1, Line1);
260         }
261     }
262     else
```

```
263     {
264         if(Column0-Column1>=-(Line1-Line0))
265         {//menos de 45 graus
266         setLineDown(Line0, Column0, Line1, Column1);
267         }
268         else
269         {//mais de 45 graus
270         setLineDown(Column0, Line0, Column1, Line1);
271         }
272     }
273 };
274
275
276 void WaveSim2DCustomBarrier::
277 setLineUp(int Lin0, int Col0, int Lin1, int Col1)
278 {
279     int x0 = Col0;
280     int x1 = Col1;
281     int y0 = Lin0;
282     int y1 = Lin1;
283     int dx = x1 - x0;
284     int dy = y1 - y0;
285     int d = 2*dy-dx;
286     int EncrE = 2*dy;
287     int EncrNE = 2*(dy-dx);
288     int x = x0;
289     int y = y0;
290
291     Lattice->setIsReflective(y, x, true);
292
293     while (x<x1)
294     {
295         if(d<=0)
296         {
297             d += EncrE;
```

```
298                 x++;
299             }
300             else
301             {
302                 d +=EncrNE;
303                 x++;
304                 y++;
305             }
306             Lattice->setIsReflective(y, x, true);
307         }
308 };
309
310
311 void WaveSim2DCustomBarrier ::
312 setLineDown(int Lin0, int Col0, int Lin1, int Col1)
313 {
314     int x0 = Col0;
315     int x1 = Col1;
316     int y0 = Lin0;
317     int y1 = Lin1;
318     int dx = x1 - x0;
319     int dy = y1 - y0;
320     int d = 2*dy-dx;
321     int EncrE = 2*dy;
322     int EncrNE = 2*(dy-dx);
323     int x = x0;
324     int y = y0;
325
326     Lattice->setIsReflective(y, x, true);
327
328     while (x<x1)
329     {
330         if(d<=0)
331         {
332             d += EncrE;
```

```
333                 x++;
334             }
335             else
336             {
337                 d +=EncrNE;
338                 x++;
339                 y--;
340             }
341             Lattice->setIsReflective(y, x, true);
342         }
343 };
```

C.7 Classe WaveSim2D2Enviroments

C.7.1 Wavesim2D2Enviroments.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34 #ifndef WAVESIM2D2ENVIROMENTS_H
35 #define WAVESIM2D2ENVIROMENTS_H
36
37 #include<iostream>
38 #include "CellularAutomata2D.h"
39 #include "Lattice2D.h"
40 #include "WaveSim2D.h"
41
42 using namespace std;
43
44 class WaveSim2D2Enviroments : public WaveSim2D
45 {
46     protected:
47         ////////////////
48         //atributos/////
49         ////////////////
50         double RefractionIndex1;
51             //indice de refracao do meio de cima
52         double RefractionIndex2;
53             //indice de refracao do meio de baixo
54         int Frontier;
55             //linha de fronteira entre os dois meiosi
56
57         ////////////////
58         //metodos///////
59         ////////////////
60         void middle(Lattice2D *Result);
61             //tem parcelas estaticas agora
62         void refractionCalc(double &F0, double &F1, double &F2,
63             double &F3, double &F4, double RefractionIndex,
64             int PositionX, int PositionY);
65             //faz o novo calculo no meio da grade
66     public:
67         ////////////////
```

```
68          //construtores//
69          ////////////////
70          WaveSim2D2Enviroments();
71              //erro, nao foi fornecido dados
72              //suficientes para a construcao
73          WaveSim2D2Enviroments(int LatticeLines,
74              int LatticeColumns,
75              int InputLin, int InputCol, bool type,
76              int LineTransition,
77              double Refraction1, double Refraction2);
78              //Cria um objeto do novo tipo
79              //mas este com a transicao de
80              //ambientes
81  };
82  #endif
```

C.7.2 WaveSim2D2Enviroments.cpp

```
1   // CAPropag2D: 2D Wave and Diffusion simulator
2    //using Cellular Automata -*- C++ -*-
3
4   // Copyright (C) 2003 Heitor Honda Federico.
5   //
6   //This file is part of the CAPropag2D library
7   //
8   // CAPropag2D is free
9   // software; you can redistribute it and/or modify it
10  //under the  terms of the GNU General Public License
11  //as published by the Free Software Foundation;
12  //either version 2, or (at your option) any later
13  //version.
14
15  // CAPropag2D  is distributed in the hope that it will
16  //be useful, but WITHOUT ANY WARRANTY; without
17  //even the implied warranty of MERCHANTABILITY or
```

```
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34
35 #include "WaveSim2D2Enviroments.h"
36
37 //////////////////////////////////////////////////
38 //Construtor: faz o mesmo que em
39 //WaveSim2D, com a diferenca /////
40 //que existe uma linha de transicao ///////////////
41 //////////////////////////////////////////////////
42
43 WaveSim2D2Enviroments::
44 WaveSim2D2Enviroments() : WaveSim2D()
45 {
46 };
47
48 WaveSim2D2Enviroments::
49 WaveSim2D2Enviroments(int LatticeLines,
50 int LatticeColumns, int InputLin, int InputCol,
51 bool type, int LineTransition, double Refraction1,
52 double Refraction2) :
```

```
53 WaveSim2D(LatticeLines , LatticeColumns ,
54 InputLin , InputCol , type)
55 {
56      assert(LineTransition >0 &&
57      LineTransition <NumberOfLines &&
58      Refraction1 >=1  && Refraction2 >=1);
59          //coerencia dos dados
60      Frontier=LineTransition ;
61      RefractionIndex1=Refraction1 ;
62      RefractionIndex2=Refraction2 ;
63 };
64
65 //////////////////////////////////////////////////
66 //protected middle: tem 2 ambientes
67 //com diferentes indices de refracao
68 //////////////////////////////////////////////////
69
70 void WaveSim2D2Enviroments::middle(Lattice2D *Result)
71 {
72      double F0, F1, F2, F3, F4;
73          //variaveis para armazenar dados
74      double RefractionNow = RefractionIndex1 ;
75          //indice de refracao no momento
76      for(int i=1; i<NumberOfLines-1; i++)
77      {
78          if(i>=Frontier)
79          {
80              RefractionNow =RefractionIndex2 ;
81          }
82          for(int j=1;j<NumberOfColumns-1;j++)
83          {
84              refractionCalc(F0, F1, F2, F3, F4,
85                  RefractionNow , i , j);
86              Result->plusCell(F0, 0 , 0 , 0 , 0 , i , j);
87          Result->plusCell(0 , F1 , 0 , 0 , 0 , i , (j+1));
```

```
88          Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
89          Result->plusCell(0, 0, 0, F3, 0, i, j-1);
90          Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
91          }
92      }
93  };
94  //refractionCalc faz os calculos do meio
95  //para um dado indice de refracao
96  void WaveSim2D2Enviroments ::
97  refractionCalc(double &F0, double &F1,
98  double &F2, double &F3,
99  double &F4, double RefractionIndex,
100 int PositionX, int PositionY)
101 {
102     double f0, f1, f2, f3, f4;
103     //variaveis auxiliares
104     double A = pow(RefractionIndex, 2); //
105     double B = 1/(2*A);          //Valores internos
106     double C = 2*sqrt(A-1);      //da matriz
107     double D = 1-2*A;                    //
108     double E = 2*A-4;                    //
109     getValues(f0, f1, f2, f3, f4,
110     PositionX, PositionY);
111     F0=B*(E*f0 + C*f1 + C*f2 + C*f3 + C*f4);//
112     F1=B*(C*F0 + f1 + f2 + D*f3 + f4);      //
113     F2=B*(C*F0 + f1 + f2 + f3 + D*f4);
114     //Multiplicacao de
115     F3=B*(C*F0 + D*f1 + f2 + f3 + f4);
116     //matrizes na raca
117     F4=B*(C*F0 + f1 + D*f2 + f3 + f4);      //
118     return;
119 };
```

C.8 Classe *WaveSim2DSample*

C.8.1 WaveSim2DSample.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34
35 #ifndef WAVESIM2DSAMPLE_H
36 #define WAVESIM2DSAMPLE_H
37
38 #include<iostream>
39 #include "CA/WaveSim2D2Enviroments.h"
40
41 using namespace std;
42
43 class WaveSim2DSample : public WaveSim2D2Enviroments
44 {
45     public:
46         ////////////////
47         //construtores//
48         ////////////////
49         WaveSim2DSample();
50             //erro, nao foi fornecido
51             //dados suficientes para a
52             //construcao
53         WaveSim2DSample(int LatticeLines,
54         int LatticeColumns, int InputLin, int InputCol,
55         bool type, int LineTransition,
56         double Refraction1, double Refraction2);
57             //Cria um objeto do novo tipo
58             //mas este com a transicao de
59             //ambientes
60         ////////////////
61         //metodos///////
62         ////////////////
63         double* returnSampleLine(int SampleFromLine);
64             //retorna um vetor
65             //de amostra da Linha indicada
66 };
67 #endif
```

C.8.2 WaveSim2DSample.cpp

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34
```

```
35 #include "WaveSim2DSample.h"
36
37 WaveSim2DSample::WaveSim2DSample() :
38 WaveSim2D2Enviroments()
39 {
40 };
41 i
42 WaveSim2DSample::WaveSim2DSample(int LatticeLines,
43 int LatticeColumns, int InputLin, int InputCol, bool type,
44 int LineTransition, double Refraction1,
45 double Refraction2) :
46 WaveSim2D2Enviroments(LatticeLines,
47 LatticeColumns, InputLin, InputCol, type, LineTransition,
48 Refraction1, Refraction2)
49 {
50 };
51
52 //////////////////////////////////////////////////////
53 //public returnSampleLine,
54 retorna um ponteiro para um vetor na linha
55 //com os valores de F
56 //////////////////////////////////////////////////////
57 double* WaveSim2DSample::
58 returnSampleLine(int SampleFromLine)
59 {
60     double *LocalSample;
61         //armazena a amostra dentro deste escopo
62     double f0, f1,f2,f3,f4;
63         //variaveis locais para auxilio de calculo
64     assert(SampleFromLine>0 &&
65     SampleFromLine<NumberOfLines);
66         //coerencia dos dados
67     LocalSample = new double[NumberOfColumns];
68     assert(LocalSample!=NULL);
69     //conseguiu alocar o vetor;
```

```
70      for(int i=0;i<NumberOfColumns;i++)
71      {
72          getValues(f0, f1, f2,
73          f3, f4, SampleFromLine, i);
74          LocalSample[i]=f0+f1+f2+f3+f4;
75              //equivalente a funcao calcF
76      }
77      return(LocalSample);
78  };
```

C.9 Classe *WaveSim2DReflect*

C.9.1 WaveSim2DReflect.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34
35 #ifndef WAVESIM2DREFLECT_H
36 #define WAVESIM2DREFLECT_H
37 #include<iostream>
38 #include "WaveSim2DSample.h"
39
40 using namespace std;
41
42 class WaveSim2DReflect : public WaveSim2DSample
43 {
44     protected:
45         //////////////
46         //atributos /////
47         //////////////
48         int ReflectionLine;
49
50         //////////////
51         //metodos ///////
52         //////////////
53         void middle(Lattice2D *Result);
54         //tem um corpo (linha) refletora
55     public:
56         //////////////
57         //construtores //
58         //////////////
59         WaveSim2DReflect();
60             //erro, nao foi fornecido dados
61             //suficientes para a construcao
62         WaveSim2DReflect
63         (int LatticeLines, int LatticeColumns,
64         int InputLin, int InputCol, bool type,
65         int LineTransition, double Refraction1,
66         double Refraction2, int LineReflection);
67             //Cria um objeto do novo tipo
```

```
68              //mas este com a transicao de
69              //ambientes
70          ////////////////
71          //metodos///////
72          ////////////////
73  };
74  #endif
```

C.9.2 WaveSim2DReflect.cpp

```
1   // CAPropag2D: 2D Wave and Diffusion simulator
2    //using Cellular Automata -*- C++ -*-
3
4   // Copyright (C) 2003 Heitor Honda Federico.
5   //
6   //This file is part of the CAPropag2D library
7   //
8   // CAPropag2D is free
9   // software; you can redistribute it and/or modify it
10  //under the  terms of the GNU General Public License
11  //as published by the Free Software Foundation;
12  //either version 2, or (at your option) any later
13  //version.
14
15  // CAPropag2D  is distributed in the hope that it will
16  //be useful, but WITHOUT ANY WARRANTY; without
17  //even the implied warranty of MERCHANTABILITY or
18  //FITNESS FOR A PARTICULAR PURPOSE.  See the
19  // GNU General Public License for more details.
20
21  // You should have received a copy of the GNU General
22  //Public License along with this library; see the
23  //file COPYING.  If not, write to the Free Software
24  //Foundation, 59 Temple Place - Suite 330,
25  //Boston, MA 02111-1307, USA.
```

```
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34
35 #include "WaveSim2DReflect.h"
36
37 //////////////////////////////////////////////////
38 //Construtor: faz o mesmo que em WaveSim2D,
39 //com a diferenca /////
40 //que existe uma linha de transicao ////////////////
41 //////////////////////////////////////////////////
42
43 WaveSim2DReflect::WaveSim2DReflect() :
44 WaveSim2DSample()
45 {
46 };
47 WaveSim2DReflect::WaveSim2DReflect(int LatticeLines,
48 int LatticeColumns, int InputLin, int InputCol, bool type,
49 int LineTransition, double Refraction1, double Refraction2,
50 int LineReflection) : WaveSim2DSample(LatticeLines,
51 LatticeColumns, InputLin, InputCol, type, LineTransition,
52 Refraction1, Refraction2)
53 {
54     assert(LineReflection>0 && LineReflection<NumberOfLines);
55     //coerencia dos dados
56     ReflectionLine = LineReflection;
57 };
58
59 //////////////////////////////////////////////////////////
60 //protected middle: tem 2 ambientes com diferentes indices de
```

```
61 refracao e tem um corpo (linha) refletor
62 //////////////////////////////////////////////////////////
63 void WaveSim2DReflect::middle(Lattice2D *Result)
64 {
65     double F0, F1, F2, F3, F4;
66         //variaveis para armazenar dados
67     double RefractionNow = RefractionIndex1;
68         //indice de refracao no momento
69     for(int i=1; i<NumberOfLines-1; i++)
70     {
71         if(i!=ReflectionLine)
72         {
73             if(i>=Frontier)
74             {
75                 RefractionNow =RefractionIndex2;
76             }
77             for(int j=1;j<NumberOfColumns-1;j++)
78             {
79                 refractionCalc(F0, F1, F2, F3, F4,
80                         RefractionNow, i, j);
81                 Result->plusCell(F0, 0, 0, 0, 0, i, j);
82             Result->plusCell(0, F1, 0, 0, 0, i, (j+1));
83             Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
84             Result->plusCell(0, 0, 0, F3, 0, i, j-1);
85             Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
86             }
87         }
88         else
89         {
90             for(int j=1;j<NumberOfColumns-1;j++)
91             {
92                 getValues(F0, F1, F2, F3, F4, i, j);
93             Result->plusCell(0, -F3, 0, 0, 0, i, (j+1));
94             Result->plusCell(0, 0, 0, 0, -F2, (i+1), j);
95             Result->plusCell(0, 0, 0, -F1, 0, i, j-1);
```

```
96                  Result->plusCell(0, 0, -F4, 0, 0, (i-1), j);
97                  }
98              }
99          }
100 };
```

C.10 Classe *WaveSim2DRandomReflect*

C.10.1 WaveSim2DRandomReflect.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 /// You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
```

```
33
34
35 #ifndef WAVESIM2DRANDOMREFLECT_H
36 #define WAVESIM2DRANDOMREFLECT_H
37 #include<iostream>
38 #include "WaveSim2DReflect.h"
39
40 using namespace std;
41
42 class WaveSim2DRandomReflect :
43 public WaveSim2DReflect
44 {
45     protected:
46         //////////////
47         //atributos /////
48         //////////////
49         //////////////
50         //metodos ///////
51         //////////////
52         void middle(Lattice2D *Result,
53         int *ReflectVector);
54             //tem um corpo (linha) refletora
55     public:
56         //////////////
57         //construtores //
58         //////////////
59         WaveSim2DRandomReflect();
60             //erro, nao foi fornecido dados
61             //suficientes para a construcao
62         WaveSim2DRandomReflect
63         (int LatticeLines, int LatticeColumns,
64         int InputLin, int InputCol, bool type,
65         int LineTransition, double Refraction1,
66         double Refraction2, int LineReflection);
67             //Cria um objeto do novo tipo
```

```
68              //mas este com a transicao de
69              //ambientes
70          ////////////////
71          //metodos///////
72          ////////////////
73          void generate
74          (int NumberOfSteps, int *ReflectVector);
75  };
76  #endif
```

C.10.2 WaveSim2DRandomReflect.cpp

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
```

```
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34
35 #include "WaveSim2DRandomReflect.h"
36
37 //////////////////////////////////////////////
38 //Construtor: faz o mesmo que em
39 //WaveSim2D, com a diferenca /
40 //que existe uma linha de transicao//////////
41 //////////////////////////////////////////////
42
43 WaveSim2DRandomReflect::WaveSim2DRandomReflect() :
44 WaveSim2DReflect()
45 {
46 };
47
48 WaveSim2DRandomReflect::
49 WaveSim2DRandomReflect(int LatticeLines,
50 int LatticeColumns, int InputLin,
51 int InputCol, bool type,
52 int LineTransition, double Refraction1,
53 double Refraction2, int LineReflection) :
54 WaveSim2DReflect(LatticeLines,
55 LatticeColumns, InputLin, InputCol, type, LineTransition,
56 Refraction1, Refraction2, LineReflection)
57 {
58 };
```

```
59 //////////////////////////////////////////////////////
60 //Avanca o numero de passos requeridos pelo áusurio/
61 //////////////////////////////////////////////////////
62 void WaveSim2DRandomReflect::generate(int NumberOfSteps,
63 int *ReflectVector)
64 {
65     Lattice2D *aux;
66         //ponteiro auxiliar para inverter grids
67     Lattice2D *Result = new Lattice2D(NumberOfLines,
68         NumberOfColumns);
69         //grade auxiliar para armazenar resultados parciais
70     bool Freq = true;
71         //faz o controle de excitacoes
72         //caso excitado em frequencia
73     //setInput();
74     for(int i=0; i<NumberOfSteps; i++)
75     {
76         if(Freq)
77         {
78             setInput();
79             if(!Frequency)
80             {
81                 Freq=false;
82             }
83         }
84         Result->inputZero();
85         leftBorder(Result);
86         rightBorder(Result);
87         topBorder(Result);
88         bottomBorder(Result);
89         middle(Result,ReflectVector);
90         //corners(Result);
91         aux=Lattice;
92         Lattice=Result;
93         Result=aux;
```

```
94      //   cout<<"ok: "<<i<<endl;
95      }
96      delete Result; //destroi a matriz auxiliar
97      aux=NULL;    //seta o ponteiro auxiliar para NULL
98  };
99
100 ////////////////////////////////////////////////////
101 //protected middle: tem 2 ambientes com diferentes
102 //indices de refracao e uma area de possivel reflecao
103 ////////////////////////////////////////////////////
104 void WaveSim2DRandomReflect::
105 middle(Lattice2D *Result, int *ReflectVector)
106 {
107     double F0, F1, F2, F3, F4;
108         //variaveis para armazenar dados
109     double RefractionNow = RefractionIndex1;
110         //indice de refracao no momento
111     for(int i=1; i<NumberOfLines-1; i++)
112     {
113         if(i!=ReflectionLine)
114         {
115             if(i>=Frontier)
116             {
117                 RefractionNow =RefractionIndex2;
118             }
119             for(int j=1;j<NumberOfColumns-1;j++)
120             {
121                 refractionCalc(F0, F1, F2, F3, F4,
122                     RefractionNow, i, j);
123                 Result->plusCell(F0, 0, 0, 0, 0, i, j);
124             Result->plusCell(0, F1, 0, 0, 0, i, (j+1));
125             Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
126             Result->plusCell(0, 0, 0, F3, 0, i, j-1);
127             Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
128             }
```

```
129             }
130             else
131             {
132                 for(int j=1;j<NumberOfColumns-1;j++)
133                 {
134                     if(ReflectVector[j-1]==1)
135                     {
136                         getValues(F0, F1, F2, F3, F4, i, j);
137             Result->plusCell(0, -F3, 0, 0, 0, i, (j+1));
138             Result->plusCell(0, 0, 0, 0, -F2, (i+1), j);
139             Result->plusCell(0, 0, 0, -F1, 0, i, j-1);
140             Result->plusCell(0, 0, -F4, 0, 0, (i-1), j);
141                     }
142                     else
143                     {
144                         refractionCalc(F0, F1, F2, F3, F4,
145                             RefractionNow, i, j);
146                 Result->plusCell(F0, 0, 0, 0, 0, i, j);
147                 Result->plusCell(0, F1, 0, 0, 0, i, (j+1));
148                 Result->plusCell(0, 0, 0, 0, F4, (i+1), j);
149                 Result->plusCell(0, 0, 0, F3, 0, i, j-1);
150                 Result->plusCell(0, 0, F2, 0, 0, (i-1), j);
151                     }
152                 }
153             }
154     }
155 };
```

C.11 CAPropag.h

```
1  // CAPropag2D: 2D Wave and Diffusion simulator
2   //using Cellular Automata -*- C++ -*-
3
4  // Copyright (C) 2003 Heitor Honda Federico.
5  //
6  //This file is part of the CAPropag2D library
7  //
8  // CAPropag2D is free
9  // software; you can redistribute it and/or modify it
10 //under the  terms of the GNU General Public License
11 //as published by the Free Software Foundation;
12 //either version 2, or (at your option) any later
13 //version.
14
15 // CAPropag2D  is distributed in the hope that it will
16 //be useful, but WITHOUT ANY WARRANTY; without
17 //even the implied warranty of MERCHANTABILITY or
18 //FITNESS FOR A PARTICULAR PURPOSE.  See the
19 // GNU General Public License for more details.
20
21 // You should have received a copy of the GNU General
22 //Public License along with this library; see the
23 //file COPYING.  If not, write to the Free Software
24 //Foundation, 59 Temple Place - Suite 330,
25 //Boston, MA 02111-1307, USA.
26
27
28 //Heitor Honda Federico
29 //sardaukar@uol.com.br
30 //Rua Saint Tropez 565, Jardim Mediterraneo
31 //Cotia, Sao Paulo Brazil
32 //CEP:06708-700
33
34 #ifndef CAPROPAG_H
```

```
35 #define CAPROPAG_H
36 #include "WaveSim2DCustomBarrier.h"
37 #include "WaveSim2D2Enviroments.h"
38 #include "DiffusionSim2D.h"
39 #endif
```

C.12 Classe *System*

C.12.1 System.h

```
1  #ifndef SYSTEM_H
2  #define SYSTEM_H
3
4  #include<cmath>
5  #include "WaveSim2DRandomReflect.h"
6  //#include "WaveSim2DReflect.h"
7
8  using namespace std;
9
10
11 class System
12 {
13     private:
14         int Steps;
15         int SampleFromLine;
16         double *Ideal;
17
18         int SysLatticeLines;
19         int SysLatticeColumns;
20         int SysInputLin;
21         int SysInputCol;
22         bool SysType;
23         int SysLineTransition;
24         double SysRefraction1;
25         double SysRefraction2;
26         int SysLineReflection;
27
28     public:
29         ////////////////
30         //atributos ///////
31         ////////////////
32
```

```
33          ////////////////
34          //construtores////
35          ////////////////
36          System();
37          System(bool BeStill);
38          //fica parado caso true e sai caso false
39              //construtor default -- falta parametros
40          System(int LatticeLines, int LatticeColumns,
41          int InputLin, int InputCol, bool type,
42          int LineTransition, double Refraction1,
43          double Refraction2, int LineReflection,
44          int StepsToGo, int SampleLine);
45              //cria a amostra considerando
46              //um corpo reflexivo (linha)
47          ~System();
48          ////////////////
49          //metodos////////
50          ////////////////
51          virtual double eval(int *Genetico, int Size);
52  };
53  #endif
```

C.12.2 System.cpp

```
1   #include "System.h"
2
3   ////////////////////////////////////////////////////////
4   //Construtor, determina a amostra que vai ser procurada///
5   ////////////////////////////////////////////////////////
6   System :: System()
7   {
8       cerr<<"Entre com dimensoes"<<endl;
9       exit(1);
10  };
11  System :: System(bool BeStill)
```

```
12 {
13     if(BeStill)
14     {
15     }
16     else
17     {
18         cout<<"saindo"<<endl;
19         exit(1);
20     }
21 };
22 System :: System(int LatticeLines, int LatticeColumns,
23 int InputLin, int InputCol,
24  bool type, int LineTransition, double Refraction1,
25  double Refraction2, int LineReflection, int StepsToGo,
26  int SampleLine)
27 {
28     assert(StepsToGo>0 && SampleLine>0); //coerencia
29     Steps = StepsToGo;
30     SampleFromLine = SampleLine;
31     SysLatticeLines = LatticeLines;
32     SysLatticeColumns = LatticeColumns;
33     SysInputLin = InputLin;
34     SysInputCol = InputCol;
35     SysLineTransition = LineTransition;
36     SysRefraction1 = Refraction1;
37     SysRefraction2 = Refraction2;
38     SysType = type;
39     SysLineReflection = LineReflection;
40
41     WaveSim2DReflect *Wav = new WaveSim2DReflect
42     (SysLatticeLines, SysLatticeColumns, SysInputLin,
43     SysInputCol, SysType, SysLineTransition, SysRefraction1,
44     SysRefraction2, SysLineReflection);
45     Wav->generate(Steps);
46     Ideal = Wav->returnSampleLine(SampleFromLine);
```

```
47      delete Wav;
48      return;
49  };
50  System :: ~System()
51  {
52      delete []Ideal;
53  };
54  //////////////////////////////////////////////////////////
55  //Eval:retorna um valor de fenotipo por comparacao      //
56  //////////////////////////////////////////////////////////
57  double System :: eval(int *Genetico, int Size)
58  {
59      double *Sample;
60      double Resu=0;
61      WaveSim2DRandomReflect *Wav =
62      new WaveSim2DRandomReflect(SysLatticeLines,
63      SysLatticeColumns, SysInputLin,
64      SysInputCol, SysType, SysLineTransition,
65      SysRefraction1, SysRefraction2, SysLineReflection);
66      Wav->generate(Steps, Genetico);
67      Sample = Wav->returnSampleLine(SampleFromLine);
68      delete Wav;
69
70      for(int i=0; i<SysLatticeColumns;i++)
71      {
72          Resu=Resu+(pow((Sample[i]-Ideal[i]),2))*100;
73      }
74      delete []Sample;
75
76  //  int pause;
77  //  cout<<"pause: "<<-Resu<<endl;
78  //  cin>>pause;
79
80      return(-Resu);
81  };
```

C.13 Classe *SystemLine*

C.13.1 SystemLine.h

```
1  #ifndef SYSTEMLINE_H
2  #define SYSTEMLINE_H
3
4  #include "System.h"
5  #include "GA/RandomFree.h"
6  //#include "WaveSim2DReflect.h"
7
8  using namespace std;
9
10
11 class SystemLine: public System
12 {
13     protected:
14     int Seed;
15     //variavel para guardar a
16     //semente de numeros aleatorios
17     int SampleFromLine2;
18     //local do segundo sensor
19     double *Ideal2;
20     //segundo sensor
21     double ErrorSize;
22     //valor para multiplicar o erro
23     bool IdealGaussianErrors;
24     //se existe ou nao erros
25     //gaussianos nas amostras ideais
26     bool SamplesGaussianErrors;
27     //se existe um erro gaussiano
28     //no modelo implementado
29     public:
30         ////////////////////
31         //atributos///////
32         ////////////////////
```

```
33
34          ////////////////////
35          //construtores ////
36          ////////////////////
37          SystemLine();
38          //construtor default -- falta parametros
39          SystemLine(bool BeStill);
40          SystemLine(int LatticeLines, int LatticeColumns,
41          int InputLin, int InputCol,
42           bool type, int LineTransition, double
43           Refraction1, double Refraction2, int LineReflection,
44           int StepsToGo,  int SampleLine, int SampleLine2,
45           int *Reflect, bool SamplesGaussian,
46           bool IdealGaussian, double Error);
47          //cria a amostra considerando
48          //um corpo reflexivo (linha)
49          ~SystemLine();
50          ////////////////////
51          //metodos /////////
52          ////////////////////
53          void getSeed();
54          //le uma semente para o gerador de
55          //numeros aleatorios no arquivo seed
56          void setSeed();
57          //coloca uma nova semente no arquivo seed
58          double eval(int *Genetico, int Size);
59  };
60  #endif
```

C.13.2 SystemLine.cpp

```
1  #include "SystemLine.h"
2  ////////////////////////////////////////////////
3  //Construtores: muda o construtor pois agora
4  //a  linha nao eh totalmente reflexiva
```

```
5  //////////s////////////////////////////////
6  SystemLine::SystemLine():System()
7  {
8  };
9  SystemLine::SystemLine(bool BeStill):System(BeStill)
10 {
11 };
12 SystemLine::SystemLine(int LatticeLines,
13 int LatticeColumns, int InputLin, int InputCol,
14 bool type, int LineTransition, double Refraction1,
15 double Refraction2, int LineReflection,
16 int StepsToGo, int SampleLine,
17 int SampleLine2, int *Reflect, bool SamplesGaussian,
18 bool IdealGaussian, double Error):System(true)
19 {
20      assert(StepsToGo>0 &&
21      SampleLine>0 && SampleLine2>0); //coerencia
22      assert(Error>0);
23      Steps = StepsToGo;
24      SampleFromLine = SampleLine;
25      SampleFromLine2 = SampleLine2;
26      SysLatticeLines = LatticeLines;
27      SysLatticeColumns = LatticeColumns;
28      SysInputLin = InputLin;
29      SysInputCol = InputCol;
30      SysLineTransition = LineTransition;
31      SysRefraction1 = Refraction1;
32      SysRefraction2 = Refraction2;
33      SysLineReflection = LineReflection;
34      IdealGaussianErrors=IdealGaussian;
35      //Caso true, sao adicionados
36      //erros seguindo uma distribuicao normal
37      //de media 0 e desvio padrao 1,
38      //multiplicados por 0.01 a todas as leituras
39      //dos dados ideiais
```

```
40      SamplesGaussianErrors=SamplesGaussian;
41      //Mesmo para todas as amostras geradas
42      ErrorSize=Error;
43      WaveSim2DRandomReflect *Wav =
44      new WaveSim2DRandomReflect(SysLatticeLines,
45      SysLatticeColumns, SysInputLin,
46      SysInputCol, SysType, SysLineTransition,
47      SysRefraction1, SysRefraction2, SysLineReflection);
48      Wav->generate(Steps, Reflect);
49      Ideal = Wav->returnSampleLine(SampleFromLine);
50      Ideal2 = Wav->returnSampleLine(SampleFromLine2);
51      ////////////Introduz erros normais caso seja pedido
52      if(IdealGaussianErrors)
53      {
54          getSeed(); //le um semente de numeros aleatorios
55          setSeed(); //escreve uma nova semente
56          for(int i=0; i < LatticeColumns-2; i++)
57          {
58              Ideal[i] = Ideal[i] +
59              ErrorSize*randomGaussian(Seed);
60          }
61      }
62      ////////////////////////////////////////////////
63      delete Wav;
64      return;
65  };
66  SystemLine::~SystemLine()
67  {
68      delete []Ideal;
69  };
70  ////////////////////////////////////////////////////
71  //getseed le a semente do arquivo seed e coloca outra
72  //setseed colota a semente no arquivo seed
73  //(sao usados no constructor e destructor)          //
74  ////////////////////////////////////////////////////
```

```
75 void SystemLine::getSeed(){
76         ifstream in("seed");
77         assert(in!=NULL);
78         in>>Seed;
79         in.close();
80         double H = randomNumber(Seed);
81         //çãinicializao da semente—ver man
82 };
83 void SystemLine::setSeed(){
84         ofstream out("seed");
85         int k;
86         k=(int)(randomNumber(Seed)*10000)*(-1);
87         out<<k;
88         out.close();
89 };
90
91 //////////////////////////////////////////////////////////
92 //Eval: retorna um valor de fenotipo por comparacao     //
93 //////////////////////////////////////////////////////////
94 double SystemLine :: eval(int *Genetico, int Size)
95 {
96     double *Sample;
97     double *Sample2;
98     double Resu=0;
99     WaveSim2DRandomReflect *Wav =
100    new WaveSim2DRandomReflect(SysLatticeLines,
101    SysLatticeColumns, SysInputLin,
102    SysInputCol, SysType, SysLineTransition,
103    SysRefraction1, SysRefraction2, SysLineReflection);
104    Wav->generate(Steps, Genetico);
105    Sample = Wav->returnSampleLine(SampleFromLine);
106    Sample2 =  Wav->returnSampleLine(SampleFromLine2);
107    delete Wav;
108    if(SamplesGaussianErrors)
109    {
```

```
110             for(int i=0; i<SysLatticeColumns;i++)
111             {
112                 Resu=Resu+(pow((Sample[i]+
113             ErrorSize*randomGaussian(Seed)-Ideal[i]),2))+
114                 (pow((Sample2[i]+ErrorSize*
115                 randomGaussian(Seed)-Ideal2[i]),2));
116             }
117         }
118         else
119         {
120             for(int i=0; i<SysLatticeColumns;i++)
121             {
122                 Resu=Resu+(pow((Sample[i]-Ideal[i]),2))+
123                 (pow((Sample2[i]-Ideal2[i]),2));
124             }
125         }
126         delete []Sample;
127         delete []Sample2;
128
129 //      int pause;
130 //      cout<<"pause: "<<-Resu<<endl;
131 //      cin>>pause;
132
133         return(-Resu);
134 };
```

C.14 Função *int main()*

```
1  #include<iostream>
2  #include<fstream>
3  #include "GA/GAOptim.h"
4  #include "CA/CAPropag.h"
5  #include "SystemLine.h"
6  #include "SystemCircular.h"
7
8  using namespace std;
9
10 void select();
11 //seleciona que tipo de simulacao se deseja
12 void testeGrande();
13 //teste para entradas grandes >
14 //5 ...1000, coloca uma saida grafica figura.ppm
15 void testePequeno();
16 //teste para entradas pequenas 3-5, coloca
17 //valores numericos e uma saida grafica figura.ppm
18 void refracao();
19 //teste para dois meios diferentes
20 void sample();
21 //amostra dados de uma linha no arquivo saida
22 void reflect();
23 //reflete em uma dada linha
24 void data();
25 //gera dados para o opengl
26 void circularBarrier();
27 //barreira circular
28 void customBarrier();
29 //gera uma barreira de acordo com um vetor
30 void findBody();
31 //procura sites refletores em uma
32 linha --- demonstracao do GA
33 void findBodyInLine();
34 //procura sites refletores
```

```
35 //em uma linha onde nem todos sao refletores
36 void findCircle();
37 //procura um circulo como barreira
38 void findCircleParameters();
39 //sabe-se queé um circulo e procura-se os parametros
40 void testBarrier();
41 //testa uma barreira colocada manualmente
42
43
44 int main()
45 {
46     select();
47     return 0;
48 };
49
50 void select()
51 {
52     int choice; //armazena a escolha do usuario
53     cout<<"(0) testeGrande():
54 ____teste para entradas grandes >5 ...1000,
55 ____coloca uma saida grafica figura.ppm."<<endl;
56     cout<<"(1) testePequeno(): teste para entradas
57 ____grandes >5 ...1000, coloca uma
58 ____saida grafica figura.ppm."<<endl;
59     cout<<"(2) refracao(): teste para
60 ____dois meios diferentes com refracao >=1."<<endl;
61     cout<<"(3) sample(): amostra
62 ____dados de uma linha no arquivo saida."<<endl;
63     cout<<"(4) reflect(): reflete
64 ____em uma dada linha."<<endl;
65     cout<<"(5) data(): imprime os dados
66 ____da simulacao em arquivos."<<endl;
67     cout<<"(6) circularBarrier: simulacao com
68 ____uma barreira circular, gera animacao."<<endl;
69     cout<<"(7) customBarrier: simulacao com
```

```
70  ␣␣␣␣uma␣dada␣por␣um␣vetor,␣gera␣uma␣animacao."<<endl;
71      cout<<"(8)␣findBody():␣procura␣sites
72  ␣␣␣␣refletores␣em␣uma␣linha␣---
73  ␣␣␣␣demonstracao␣do␣GA."<<endl;
74      cout<<"(9)␣findBodyInLine():
75  ␣␣␣␣mesmo,␣mas␣o␣sites␣refletores
76  ␣␣␣␣sao␣controlados␣no␣main."<<endl;
77      cout<<"(10)␣findCircle():␣procura
78  ␣␣␣␣uma␣barreira␣circular."<<endl;
79      cout<<"(11)␣findCircleParameters:
80  ␣␣␣␣sabe-se␣que␣eh␣um␣circulo␣e␣procura-se
81  ␣␣␣␣os␣parametros."<<endl;
82      cout<<"(12)␣testBarrier():␣entrada␣manual
83  ␣␣␣␣de␣barreiras."<<endl;
84      cout<<"Escolha␣um␣tipo␣de␣teste:␣";
85      cin>>choice;
86
87
88      switch(choice)
89      {
90          case 0:
91              testeGrande();
92              break;
93          case 1:
94              testePequeno();
95              break;
96          case 2:
97              refracao();
98              break;
99          case 3:
100             sample();
101             break;
102         case 4:
103             reflect();
104             break;
```

```
105             case 5:
106                 data();
107                 break;
108             case 6:
109                 circularBarrier();
110                 break;
111             case 7:
112                 customBarrier();
113                 break;
114             case 8:
115                 findBody();
116                 break;
117             case 9:
118                 findBodyInLine();
119                 break;
120             case 10:
121                 findCircle();
122                 break;
123             case 11:
124                 findCircleParameters();
125                 break;
126             case 12:
127                 testBarrier();
128                 break;
129             default:
130                 cout<<"Escolha␣naoé␣␣valida!!"<<endl;
131         cerr<<"Erro,␣escolha␣de␣teste␣invalida!!"<<endl;
132                 exit(1);
133         }
134
135         return;
136 };
137
138 void testeGrande()
139 {
```

```
140     int Steps, Lines, Columns, InputX, InputY;
141     double Graph;
142     bool Frequencia;
143     cout<<"\nA entrada é em
144 frequencia? (true=1, false=0): ";
145     cin>>Frequencia;
146     cout<<"\nEntre com o numero de linhas da grade: ";
147     cin>>Lines;
148     cout<<"\nEntre com o numero de colunas da grade: ";
149     cin>>Columns;
150     cout<<"\nEntre a coordenada x da excitacao: ";
151     cin>>InputX;
152     cout<<"\nEntre a coordenada y da excitacao: ";
153     cin>>InputY;
154     cout<<"\nEntre com o numero de passos que quer simular: ";
155     cin>>Steps;
156     cout<<"\nEntre com a precisao do
157 grafico que deseja(true=0.1 false=0.0001): ";
158     cin>>Graph;
159     cout<<"\ncomecando... "<<endl;
160     WaveSim2D *Wav =
161     new WaveSim2D(Lines,Columns,
162     InputX-1, InputY-1, Frequencia);
163     Wav->generate(Steps);
164     Wav->printToFile(Graph);
165     delete Wav;
166     return;
167 };
168
169 void testePequeno()
170 {
171     int Steps, Lines, Columns, InputX, InputY;
172     double Graph;
173     bool Frequencia;
174     cout<<"\nA entrada é em frequencia?
```

```
175 ␣␣␣␣(true=1,␣false=0):␣";
176       cin>>Frequencia;
177       cout<<"\nEntre␣com␣o␣numero␣de␣linhas␣da␣grade:␣";
178       cin>>Lines;
179       cout<<"\nEntre␣com␣o␣numero␣de␣colunas␣da␣grade:␣";
180       cin>>Columns;
181       cout<<"\nEntre␣a␣coordenada␣x␣da␣excitacao:␣";
182       cin>>InputX;
183       cout<<"\nEntre␣a␣coordenada␣y␣da␣excitacao:␣";
184       cin>>InputY;
185       cout<<"\nEntre␣com␣o␣numero␣de␣passos
186 ␣␣␣␣que␣quer␣simular:␣";
187       cin>>Steps;
188       cout<<"\nEntre␣com␣a␣precisao␣do␣grafico
189 ␣␣␣␣que␣deseja(true=0.1␣false=0.0001):␣";
190       cin>>Graph;
191       cout<<"\ncomecando...␣"<<endl;
192       WaveSim2D *Wav =
193       new WaveSim2D(Lines,Columns,
194       InputX-1, InputY-1, Frequencia);
195       Wav->generate(Steps);
196       Wav->printToFile(Graph);
197       Wav->print();
198       delete Wav;
199       return;
200 };
201 void refracao()
202 {
203       int Steps, Line, Lines, Columns, InputX, InputY;
204       double n1, n2, Graph;
205       bool Frequencia;
206       cout<<"\nA␣entradaé␣␣em␣frequencia?␣(true=1,␣false=0):␣";
207       cin>>Frequencia;
208       cout<<"\nEntre␣com␣o␣numero␣de␣linhas␣da␣grade:␣";
209       cin>>Lines;
```

```
210     cout<<"\nEntre com o numero de colunas da grade: ";
211     cin>>Columns;
212     cout<<"\nEntre a coordenada x da excitacao: ";
213     cin>>InputX;
214     cout<<"\nEntre a coordenada y da excitacao: ";
215     cin>>InputY;
216     cout<<"\nEntre com o numero de passos
217 ____que quer simular: ";
218     cin>>Steps;
219     cout<<"\nEntre com o valor do
220 ____primeiro indice de refracao: ";
221     cin>>n1;
222     cout<<"\nEntre com o valor do
223 ____segundo indice de refracao: ";
224     cin>>n2;
225     cout<<"\nEntre com a Linha de fronteira: ";
226     cin>>Line;
227     cout<<"\nEntre com a precisao do
228 ____grafico que deseja(true=0.1 false=0.0001): ";
229     cin>>Graph;
230     cout<<"\ncomecando... "<<endl;
231     WaveSim2D2Enviroments *Wav =
232     new WaveSim2D2Enviroments(Lines, Columns, InputX-1,
233     InputY-1, Frequencia, Line, n1, n2);
234     Wav->generate(Steps);
235     Wav->printToFile(Graph);
236     delete Wav;
237     return;
238 };
239
240 void sample()
241 {
242     int Steps, Line, Lines, Columns, InputX,
243     InputY, SampleFromLine;
244     double n1, n2, Graph, *SampleLine;
```

```
245     bool Frequencia;
246     ofstream FileOut("amostra");
247     assert(FileOut!=NULL); //abriu o arquivo
248     cout<<"\nA entrada é
249 em frequencia? (true=1, false=0): ";
250     cin>>Frequencia;
251     cout<<"\nEntre com o numero de
252 linhas da grade: ";
253     cin>>Lines;
254     cout<<"\nEntre com o numero de colunas da grade: ";
255     cin>>Columns;
256     cout<<"\nEntre a coordenada x da excitacao: ";
257     cin>>InputX;
258     cout<<"\nEntre a coordenada y da excitacao: ";
259     cin>>InputY;
260     cout<<"\nEntre com o numero de
261 passos que quer simular: ";
262     cin>>Steps;
263     cout<<"\nEntre com o valor do primeiro
264 indice de refracao: ";
265     cin>>n1;
266     cout<<"\nEntre com o valor do segundo
267 indice de refracao: ";
268     cin>>n2;
269     cout<<"\nEntre com a Linha de fronteira: ";
270     cin>>Line;
271     cout<<"\nEntre com a precisao do
272 grafico que deseja(true=0.1 false=0.0001): ";
273     cin>>Graph;
274     cout<<"\nEntre com a linha de amostra: ";
275     cin>>SampleFromLine;
276     cout<<"\ncomecando... "<<endl;
277     WaveSim2DSample *Wav =
278     new WaveSim2DSample(Lines, Columns, InputX-1,
279     InputY-1, Frequencia, Line, n1, n2);
```

```
280     Wav->generate(Steps);
281     Wav->printToFile(Graph);
282     SampleLine = Wav->returnSampleLine(SampleFromLine);
283     delete Wav;
284     for(int i=0;i<Columns;i++)
285     {
286         cout<<SampleLine[i]<<" ";
287         FileOut<<i<<"\t"<<SampleLine[i]<<endl;
288     }
289     FileOut.close();
290     delete SampleLine;
291     return;
292 };
293
294 void reflect()
295 {
296     int Steps, Line, Lines, Columns, InputX,
297     InputY, SampleFromLine, ReflectLine;
298     double n1, n2, Graph, *SampleLine;
299     bool Frequencia;
300     ofstream FileOut("amostra");
301     assert(FileOut!=NULL); //abriu o arquivo
302     cout<<"\nA entradaé  em
303 frequencia? (true=1, false=0): ";
304     cin>>Frequencia;
305     cout<<"\nEntre com o numero
306 de linhas da grade: ";
307     cin>>Lines;
308     cout<<"\nEntre com o numero
309 de colunas da grade: ";
310     cin>>Columns;
311     cout<<"\nEntre a coordenada x da excitacao: ";
312     cin>>InputX;
313     cout<<"\nEntre a coordenada y da excitacao: ";
314     cin>>InputY;
```

```
315         cout<<"\nEntre␣com␣o␣numero␣de
316         passos que quer simular: ";
317         cin>>Steps;
318         cout<<"\nEntre␣com␣o␣valor␣do␣primeiro
319 ␣␣␣␣indice␣de␣refracao:␣";
320         cin>>n1;
321         cout<<"\nEntre␣com␣o␣valor␣do␣segundo
322 ␣␣␣␣indice␣de␣refracao:␣";
323         cin>>n2;
324         cout<<"\nEntre␣com␣a␣Linha␣de␣fronteira:␣";
325         cin>>Line;
326         cout<<"\nEntre␣com␣a␣precisao␣do
327 ␣␣␣␣grafico␣que␣deseja(true=0.1␣false=0.0001):␣";
328         cin>>Graph;
329         cout<<"\nEntre␣com␣a␣linha␣de␣amostra:␣";
330         cin>>SampleFromLine;
331         cout<<"\nEntre␣com␣a␣linha␣de␣reflexao:␣";
332         cin>>ReflectLine;
333         cout<<"\ncomecando...␣"<<endl;
334         WaveSim2DReflect *Wav =
335         new WaveSim2DReflect(Lines, Columns, InputX-1,
336         InputY-1, Frequencia, Line, n1, n2, ReflectLine);
337         Wav->generate(Steps);
338         Wav->printToFile(Graph);
339         SampleLine = Wav->returnSampleLine(SampleFromLine);
340         delete Wav;
341         for(int i=0;i<Columns;i++)
342         {
343             cout<<SampleLine[i]<<"␣";
344             FileOut<<i<<"\t"<<SampleLine[i]<<endl;
345         }
346         FileOut.close();
347         delete SampleLine;
348         return;
349 };
```

```
350 void data()
351 {
352     int Steps, Line, Lines, Columns, InputX, InputY, Type;
353     double n1, n2, Graph, P0, P1, P2, P3;
354     bool Frequencia;
355     WaveSim2D2Enviroments *Wav;
356     DiffusionSim2D *Diff;
357     cout<<"\nA entrada é em
358 frequencia? (true=1, false=0): ";
359     cin>>Frequencia;
360     cout<<"\nEntre com o numero
361 de linhas da grade: ";
362     cin>>Lines;
363     cout<<"\nEntre com o numero
364 de colunas da grade: ";
365     cin>>Columns;
366     cout<<"\nEntre a coordenada x da excitacao: ";
367     cin>>InputX;
368     cout<<"\nEntre a coordenada y da excitacao: ";
369     cin>>InputY;
370     cout<<"\nEntre com o numero de
371 passos que quer simular: ";
372     cin>>Steps;
373
374     cout<<"\nO processo eh de difusao
375 ou propagacao de ondas?"<<endl;
376     cout<<"\t(0) Difusao."<<endl;
377     cout<<"\t(1) Propagacao de ondas;"<<endl;
378     cout<<"Tipo de processo:";
379     cin>>Type;
380
381     switch(Type)
382     {
383         case 0:
384             /*cout<<"\nEntre com o valor de P0: ";
```

```
385             cin>>P0;
386             cout<<"\nEntre com o valor de P1: ";
387             cin>>P1;
388             cout<<"\nEntre com o valor de P2: ";
389             cin>>P2;
390             cout<<"\nEntre com o valor de P3: ";
391             cin>>P3;*/
392             P0=0.25;
393             P1=0.25;
394             P2=0.25;
395             P3=0.25;
396             cout<<"\ncomecando... "<<endl;
397             Diff = new
398             DiffusionSim2D(Lines, Columns,
399             InputX-1, InputY-1, Frequencia,
400             P0, P1, P2, P3);
401             Diff->generate(Steps, "instante");
402             delete Diff;
403             break;
404         case 1:
405             cout<<"\nEntre com o valor
406 ____________do primeiro indice de refracao: ";
407             cin>>n1;
408             cout<<"\nEntre com o valor
409 ____________do segundo indice de refracao: ";
410             cin>>n2;
411             cout<<"\nEntre com a Linha de fronteira: ";
412             cin>>Line;
413             cout<<"\ncomecando... "<<endl;
414             Wav =
415             new WaveSim2D2Enviroments(Lines,
416             Columns, InputX-1, InputY-1,
417             Frequencia, Line, n1, n2);
418             Wav->generate(Steps, "instante");
419             delete Wav;
```

```
420                 break;
421             default:
422                 cout<<"\nOpcao␣invalida!!!!!"<<endl;
423                 cerr<<"\nOpcao␣invalida!!!!!"<<endl;
424                 exit(1);
425         }
426         return;
427 }
428 void circularBarrier()
429 {
430         int Steps, Line, Lines,
431         Columns, Type, CenterX, CenterY, Radius;
432         bool Frequencia;
433         cout<<"\nA␣entradaé␣␣em
434 ␣␣␣␣frequencia?␣(true=1,␣false=0):␣";
435         cin>>Frequencia;
436         cout<<"\nEntre␣com␣o␣numero␣de␣linhas␣da␣grade:␣";
437         cin>>Lines;
438         cout<<"\nEntre␣com␣o␣numero␣de␣colunas␣da␣grade:␣";
439         cin>>Columns;
440         cout<<"\nEntre␣com␣o␣numero␣de
441 ␣␣␣␣passos␣que␣quer␣simular:␣";
442         cin>>Steps;
443         cout<<"\nEntre␣a␣␣linha␣do␣centro
444 ␣␣␣␣da␣barreira␣circular:␣";
445         cin>>CenterX;
446         cout<<"\nEntre␣a␣coluna␣do
447 ␣␣␣␣centro␣da␣barreira␣circular:␣";
448         cin>>CenterY;
449         cout<<"\nEntre␣com␣o␣raio␣da␣barreira␣circular:␣";
450         cin>>Radius;
451         cout<<"\ncomecando...␣"<<endl;
452         WaveSim2DCustomBarrier *Wav =
453         new WaveSim2DCustomBarrier
454         (Lines, Columns,1,1, Frequencia);
```

```
455     Wav->setCircularBarrier
456     (Radius , CenterX -1, CenterY -1);
457     Wav->generate(Steps , "instante");
458     delete Wav;
459     return;
460 }
461
462 void customBarrier()
463 {
464     int Steps,Lines , Columns, Type, CenterX , CenterY , Radius;
465     bool Frequencia;
466     int *Barrier;
467     cout<<"\nA entradaé  em frequencia? (true=1, false=0): ";
468     cin>>Frequencia;
469     cout<<"\nEntre com o numero de linhas da grade: ";
470     cin>>Lines;
471     cout<<"\nEntre com o numero de colunas da grade: ";
472     cin>>Columns;
473     cout<<
474     "\nEntre com o numero de passos que quer simular: ";
475     cin>>Steps;
476     cout<<"\ncomecando... "<<endl;
477     Barrier = new int[(Lines -2)*(Columns-2)]
478     assert (Barrier!=NULL);
479     for(int i=0; i<(Lines -2)*(Columns-2); i++)
480     {
481         Barrier[i]=i%2;
482     }
483     WaveSim2DCustomBarrier *Wav =
484     new WaveSim2DCustomBarrier
485     (Lines , Columns,1,1, Frequencia);
486     Wav->setCustomizedBarrier
487     (Barrier , (Lines -2)*(Columns-2));
488     Wav->generate(Steps , "instante");
489     delete Wav;
```

```
490         return;
491 }
492
493 void findBody()
494 {
495         int Steps, Line, Lines, Columns,
496         InputX, InputY, SampleFromLine, ReflectLine;
497         double n1, n2, Graph, *SampleLine;
498         bool Frequencia;
499         ofstream FileOut("amostra");
500         assert(FileOut!=NULL); //abriu o arquivo
501         cout<<
502         "\nA entradaé  em frequencia? (true=1, false=0): ";
503         cin>>Frequencia;
504         cout<<"\nEntre com o numero de linhas da grade: ";
505         cin>>Lines;
506         cout<<
507         "\nEntre com o numero de colunas da grade: ";
508         cin>>Columns;
509         cout<<"\nEntre a coordenada x da excitacao: ";
510         cin>>InputX;
511         cout<<"\nEntre a coordenada y da excitacao: ";
512         cin>>InputY;
513         cout<<
514         "\nEntre com o numero de
515 passos que quer simular: ";
516         cin>>Steps;
517         cout<<"\nEntre com o valor do
518 primeiro indice de refracao: ";
519         cin>>n1;
520         cout<<
521         "\nEntre com o valor do segundo
522 indice de refracao: ";
523         cin>>n2;
524         cout<<"\nEntre com a Linha de fronteira: ";
```

```
525     cin>>Line;
526     cout<<"\nEntre com a precisao do
527 ____grafico que deseja(true=0.1 false=0.0001): ";
528     cin>>Graph;
529     cout<<"\nEntre com a linha de amostra: ";
530     cin>>SampleFromLine;
531     cout<<"\nEntre com a linha de reflexao: ";
532     cin>>ReflectLine;
533     cout<<"\ncomecando... "<<endl;
534     assert(InputX-1>=0 && InputX-1<Lines &&
535     InputY-1>0 && InputY-1<Columns);
536     System *Sys = new System
537     (Lines, Columns, InputX-1, InputY-1, Frequencia,
538     Line, n1, n2, ReflectLine, Steps, SampleFromLine);
539     Optimizer<System> *optim = new Optimizer<System>
540     (Sys, 10, Columns-2, 0.5, 0.05,2, 1.0, "saida");
541     optim->stepsOfOptimization(6000, 50, true, true);
542     delete optim;
543     delete Sys;
544     return;
545 };
546 void findBodyInLine()
547 {
548     int Steps, Line, Lines, Columns,
549     InputX, InputY, SampleFromLine, SampleFromLine2,
550     ReflectLine, ReflectType;
551     double n1, n2, Graph, *SampleLine;
552     bool Frequencia;
553     ofstream FileOut("amostra");
554     assert(FileOut!=NULL); //abriu o arquivo
555     cout<<
556     "\nA entradaé em frequencia? (true=1, false=0): ";
557     cin>>Frequencia;
558     cout<<
559     "\nEntre com o numero de linhas da grade: ";
```

```
560       cin>>Lines;
561       cout<<
562       "\nEntre␣com␣o␣numero␣de␣colunas␣da␣grade:␣";
563       cin>>Columns;
564       cout<<
565       "\nEntre␣a␣coordenada␣x␣da␣excitacao:␣";
566       cin>>InputX;
567       cout<<
568       "\nEntre␣a␣coordenada␣y␣da␣excitacao:␣";
569       cin>>InputY;
570       cout<<
571       "\nEntre␣com␣o␣numero␣de␣passos
572 ␣␣␣␣que␣quer␣simular:␣";
573       cin>>Steps;
574       cout<<
575       "\nEntre␣com␣o␣valor␣do
576 ␣␣␣␣primeiro␣indice␣de␣refracao:␣";
577       cin>>n1;
578       cout<<
579       "\nEntre␣com␣o␣valor␣do
580 ␣␣␣␣segundo␣indice␣de␣refracao:␣";
581       cin>>n2;
582       cout<<
583       "\nEntre␣com␣a␣Linha␣de␣fronteira:␣";
584       cin>>Line;
585       cout<<
586       "\nEntre␣com␣a␣precisao␣do␣grafico␣que
587 ␣␣␣␣deseja(true=0.1␣false=0.0001):␣";
588       cin>>Graph;
589       cout<<"\nEntre␣com␣a␣linha␣de␣amostra␣1:␣";
590       cin>>SampleFromLine;
591       cout<<"\nEntre␣com␣a␣linha␣de␣amostra␣2:␣";
592       cin>>SampleFromLine2;
593       cout<<"\nEntre␣com␣a␣linha␣de␣reflexao:␣";
594       cin>>ReflectLine;
```

```
595     cout<<"\nOs␣tipos␣de␣barreiras␣ãso:␣"<<endl;
596     cout<<"\t(0)␣buraco␣no␣meio;"<<endl;
597     cout<<"\t(1)␣alterna␣zeros␣e␣uns;"<<endl;
598     cout<<"\t(2)␣corpo␣imerso;"<<endl;
599     cout<<"\t(3)␣barreira␣completa."<<endl;
600     cout<<"------>Entre␣com␣a␣a␣opcao␣desejada:␣";
601     cin>>ReflectType;
602     assert(InputX-1>=0 && InputX-1<Lines
603     && InputY-1>0 && InputY-1<Columns);
604     int *Reflect = new int[Columns-2];
605     assert(Reflect!=NULL);
606     int i=0;
607     switch(ReflectType)
608     {
609         case 0:
610         //////////buraco no meio//////////////
611             i=0;
612             for(i=0;i<(Columns-2)/3; i++)
613             {
614             Reflect[i]=1;
615             }
616             for(;i<2*(Columns-2)/3;i++)
617             {
618                 Reflect[i]=0;
619             }
620             for(;i<Columns-2;i++)
621             {
622                 Reflect[i]=1;
623             }
624             break;
625         case 1:
626         ///////alterna 0 e 1//////////////////
627             i=0;
628             for(i=0;i<Columns-2;i++)
629             {
```

```
630                 Reflect[i]=i%2;
631             }
632             break;
633         case 2:
634         ////////// corpo imerso ////////////////
635             i=0;
636             for(i=0;i<(Columns-2)/3; i++)
637             {
638                 Reflect[i]=0;
639             }
640             for(;i<2*(Columns-2)/3;i++)
641             {
642                 Reflect[i]=1;
643             }
644             for(;i<Columns-2;i++)
645             {
646                 Reflect[i]=0;
647             }
648             break;
649         case 3:
650         ////// barreira completa ///////////////
651             i=0;
652             for(i=0; i<Columns-2; i++)
653             {
654                 Reflect[i]=1;
655             }
656             break;
657         default:
658             cout<<"Escolha␣naoé␣␣valida!!"<<endl;
659     cerr<<"Erro,␣escolha␣de␣teste␣invalida!!"<<endl;
660             exit(1);
661     }
662     ////////////////////////////////////////
663     cout<<"Comecando␣..."<<endl;
664     cout<<endl;
```

```
665     for(i=0;i<Columns-2;i++)cout<<Reflect[i]<<" ";
666     cout<<endl;
667     SystemLine *Sys = new SystemLine
668     (Lines, Columns, InputX-1, InputY-1, Frequencia,
669     Line, n1, n2, ReflectLine, Steps, SampleFromLine,
670     SampleFromLine2,
671                         Reflect, true, true, 0.0001);
672     Optimizer<SystemLine> *optim =
673     new Optimizer<SystemLine>
674     (Sys, 100, Columns-2, 0.5, 0.05,2, 1.0, "saida");
675     optim->stepsOfOptimization(6000, 10, true, true);
676     delete optim;
677     delete Sys;
678
679 };
680
681
682 void findCircle()
683 {
684     int Steps, Line, Lines, Columns, Type,
685     CenterX, CenterY, Radius, SampleFromLine,
686     SampleFromLine2;
687     bool Frequencia;
688     Chromossome<SystemCircular> *Found;
689
690     cout<<
691     "\nA entrada é em frequencia? (true=1, false=0): ";
692     cin>>Frequencia;
693     cout<<
694     "\nEntre com o numero de linhas da grade: ";
695     cin>>Lines;
696     cout<<
697     "\nEntre com o numero de colunas da grade: ";
698     cin>>Columns;
699     cout<<
```

```
700     "\nEntre␣com␣o␣numero␣de␣passos␣que␣quer␣simular:␣";
701     cin>>Steps;
702     cout<<
703     "\nEntre␣a␣␣linha␣do␣centro␣da␣barreira␣circular:␣";
704     cin>>CenterX;
705     cout<<
706     "\nEntre␣a␣coluna␣do␣centro␣da␣barreira␣circular:␣";
707     cin>>CenterY;
708     cout<<
709     "\nEntre␣com␣o␣raio␣da␣barreira␣circular:␣";
710     cin>>Radius;
711     cout<<
712     "\nEntre␣com␣a␣linha␣de␣amostra␣1:␣";
713     cin>>SampleFromLine;
714     cout<<
715     "\nEntre␣com␣a␣linha␣de␣amostra␣2:␣";
716     cin>>SampleFromLine2;
717     cout<<
718     "\ncomecando...␣"<<endl;
719
720
721     SystemCircular *Sys = new SystemCircular
722     (Lines, Columns, Frequencia, Steps,
723     SampleFromLine, SampleFromLine2,
724     Radius, CenterX-1, CenterY-1, false, false, 0.0001);
725     Optimizer<SystemCircular> *optim =
726     new Optimizer<SystemCircular>
727     (Sys, 100, (Lines-2)*(Columns-2), 0.5,
728     0.0005, 2, 1.0, "desempenhoGA");
729     optim->stepsOfOptimization(6000, 50, true, true);
730
731     Found = optim->returnBestFit();
732
733     WaveSim2DCustomBarrier *Wav = new
734     WaveSim2DCustomBarrier(Lines, Columns,1,1, true);
```

```
735     Wav->setCustomizedBarrier
736     (Found->GenBox,(Lines-2)*(Columns-2));
737     Wav->generate(100, "encontrado");
738     delete Wav;
739     WaveSim2DCustomBarrier *Wav2 =
740     new WaveSim2DCustomBarrier(Lines,
741     Columns,1,1, Frequencia);
742     Wav2->setCircularBarrier(Radius, CenterX, CenterY);
743     Wav2->generate(Steps, "ideal");
744     delete Wav2;
745     delete optim;
746     delete Sys;
747
748 }
749
750 void findCircleParameters()
751 {
752     int BestRaio=2, BestLine=2, BestColumn=2, Seed;
753     double *Sensor1, *Sensor2;
754     double *Ideal1, *Ideal2;
755     double ErrorSize = 0.00001;
756     double Resu=-10000;
757     //recolho as amostra
758     Seed = getSeed();
759     setSeed(Seed);
760     WaveSim2DCustomBarrier *Wav =
761     new WaveSim2DCustomBarrier(31+16,
762     31+16, 1, 1, true);
763     Wav->setCircularBarrier(5, 14+8, 14+8);
764     Wav->generate(150, "ideal");
765     Ideal1 = Wav->returnSampleLine(7);
766     Ideal2 = Wav->returnSampleLine(39);
767     for(int i=0; i < 31+16; i++)
768     {
769         Ideal1[i] = Ideal1[i] +
```

```
770         ErrorSize*randomGaussian(Seed);
771         Ideal2[i] = Ideal2[i] +
772         ErrorSize*randomGaussian(Seed);
773     }
774
775     delete Wav;
776     //varro as solucoes
777
778     for(int i=0; i<31; i++)
779     {
780         for(int j=0; j<31; j++)
781         {
782             for(int k = 2; k<7; k++)
783             {
784     double Eval=0;
785     WaveSim2DCustomBarrier
786     *Wav2 =
787     new WaveSim2DCustomBarrier
788     (31+16, 31+16, 1, 1, true);
789     Wav2->setCircularBarrier(k, i+8, j+8);
790     Wav2->generate(150);
791     Sensor1 = Wav->returnSampleLine(7);
792     Sensor2 = Wav->returnSampleLine(39);
793     delete Wav2;
794     Eval = 0;
795     for(int w=0; w<31+16;w++)
796     {
797     Eval=Eval+
798     (pow((Sensor1[w]+ErrorSize*
799     randomGaussian(Seed)-Ideal1[w]),2));
800     +(pow((Sensor2[w]+ErrorSize*
801     randomGaussian(Seed)-Ideal2[w]),2));
802                 }
803                 Eval = -Eval;
804                 if(Eval>Resu)
```

```
805                 {
806             BestRaio = k;
807             BestLine = i;
808             BestColumn = j;
809             Resu = Eval;
810             system("clear");
811             cout<<"Melhor Resultado:"<<endl;
812             cout<<"\tRaio = "<<k<<"."<<endl;
813             cout<<"\tLinha = "<<i<<"."<<endl;
814             cout<<"\tColuna = "<<j<<"."<<endl;
815             cout<<"\tDiferenca = "<<Resu<<"."<<endl;
816         }
817         else
818         {   system("clear");
819             cout<<"analisado"<<endl;
820             cout<<"\tRaio = "<<k<<"."<<endl;
821             cout<<"\tLinha = "<<i<<"."<<endl;
822             cout<<"\tColuna = "<<j<<"."<<endl;
823             cout<<"Ainda o melhor Resultado:"<<endl;
824             cout<<"\tRaio = "<<BestRaio<<"."<<endl;
825             cout<<"\tLinha = "<<BestLine<<"."<<endl;
826             cout<<"\tColuna = "<<BestColumn<<"."<<endl;
827             cout<<"\tDiferenca = "<<Resu<<"."<<endl;
828                 }
829                 delete []Sensor1;
830                 delete []Sensor2;
831             }
832         }
833     }
834     delete []Ideal1;
835     delete []Ideal2;
836     WaveSim2DCustomBarrier *Wav2 =
837     new WaveSim2DCustomBarrier
838     (31+16, 31+16, 1, 1, true);
839     Wav2->setCircularBarrier
```

```
840         (BestRaio , BestLine+8, BestColumn+8);
841         Wav2->generate(150, "solucao");
842         delete Wav2;
843 }
844
845 void testBarrier()
846 {
847         int Steps,Lines, Columns,
848         Type, CenterX, CenterY, Radius;
849         bool Frequencia;
850         int *Barrier;
851         cout<<"\nA entradaé  em frequencia?
852 (true=1, false=0): ";
853         cin>>Frequencia;
854         cout<<"\nEntre com o numero
855 de linhas da grade: ";
856         cin>>Lines;
857         cout<<"\nEntre com o numero de
858 colunas da grade: ";
859         cin>>Columns;
860         cout<<"\nEntre com o numero de
861 passos que quer simular: ";
862         cin>>Steps;
863         cout<<"\ncomecando... "<<endl;
864         Barrier = new int[(Lines-2)*(Columns-2)]
865         assert (Barrier!=NULL);
866         for(int i=0; i<(Lines-2)*(Columns-2); i++)
867         {
868             cout<<"Entre com a barreira:";
869             cin>>Barrier[i];
870         }
871         WaveSim2DCustomBarrier *Wav =
872         new WaveSim2DCustomBarrier(Lines,
873         Columns,1,1, Frequencia);
874         Wav->setCustomizedBarrier(Barrier,
```

```
875         (Lines -2)*(Columns -2));
876         Wav->generate(Steps , "instante");
877         delete Wav;
878         return ;
879 }
```

Apêndice D

Visualizador de superfícies usando OpenGL

Este programa processa uma série de quadros gerados um por arquivo de maneira a gerar uma animação tridimensional de uma propagação de ondas ou um processo de difusão.

D.1 Graphics.h

```
1  #include<GL/gl.h>
2  #include<GL/glu.h>
3  #include<GL/glut.h>
4  #include<stdio.h>
5  #include<stdlib.h>
6  #include<math.h>
7  #include<string.h>
8
9
10 /*propriedades da perspectiva*/
11 static double FrustrumFOV = 30.0,
12                FrustrumASPECT = (1.0/1.0),
13                FrustrumNEAR = 1.5,
14                FrustrumFAR = 10.0;
15
16 /*propriedades da superficie a ser plotada*/
```

```
17 static GLfloat SurfaceDiffuseMaterial[5][4] =
18 {
19       { 0.7, 0.7, 0.7, 1.0 },
20       { 0.8, 0.8, 0.3, 1.0 },
21       { 0.4, 1.0, 0.3, 1.0 },
22       { 0.5, 0.9, 0.5, 1.0 },
23       { 1.0, 1.0, 1.0, 1.0 },
24 };
25 static GLfloat SurfaceSpecularMaterial[5][4] =
26 {
27       { 0.8, 0.8, 0.8, 1.0 },
28       { 0.8, 0.8, 0.3, 1.0 },
29       { 0.4, 0.9, 0.3, 1.0 },
30       { 0.5, 0.9, 0.5, 1.0 },
31       { 1.0, 1.0, 1.0, 1.0 },
32 };
33 static GLfloat SurfaceShininess[5][1] =
34 {
35       { 60.0 },
36       { 50.0 },
37       { 40.0 },
38       { 70.0 },
39       { 95.0 },
40 };
41 /*propriedades das luzes */
42 static
43 GLfloat AmbientLight[] = { 0.0, 0.0, 0.0, 1.0 },
44         DiffuseLight[] = { 1.0, 1.0, 1.0, 1.0 },
45     SpecularLight[] = { 1.0, 1.0, 1.0, 1.0 },
46     LModelAmbient[] = { 0.1, 0.1, 0.1, 1.0 },
47     LightPosition[] = { 1.0, 1.0, 1.0, 0.0};
48
49 /*Camera de observacao*/
50 static double Camera[3]={2.0,1.5,3.0} ;
51
```

```
52 /*Dados da superficie a ser desenhada*/
53
54 struct PlotSurface
55 {
56     int NumberOfPointsX,
57     /*armazena o numero de pontos X vs Z*/
58         NumberOfPointsZ;
59     GLfloat X0,
60     /*armazena o inicio do x*/
61             Z0,
62             /*armazena o inicio do z*/
63             XMax,
64             /*armazena o final do x*/
65             ZMaX,
66             /*armazena o final do z*/
67             MaxHeight,
68             /*armazena maxima elevacao em y*/
69             MinHeight,
70             /*armazena minima elevacao em y*/
71             **Heights,
72             /*armazena as elevacoes*/
73             DeltaR,
74             /*armazena o valor dos pontos da grade*/
75             DeltaT,
76             /*armazena o valor do delta do tempo*/
77             ***NormalVectors;
78             /*tem os vetores normais
79             da superficie*/
80
81 };
82
83 char  DataFileBaseName[80];
84 /*guarda a base do nome*/
85 int FrameNumber;
86 /*guarda o numero da frame a ser plotada*/
```

```
87  GLfloat DeltaR=0.01;
88  int FramesNumber;
89  /* guarda o numero de frames exstentes*/
90  struct PlotSurface *SurfaceToPlot;
91
92  void allocVectorHeights(struct PlotSurface *Surface);
93  /*aloca os vetores ánecessrios para armazenar as alturas*/
94  void freeVectorHeights(struct PlotSurface *Surface);
95  /*libera os vetores ánecessrios para armazenar as alturas*/
96  void allocNormalVectors(struct PlotSurface *Surface);
97  /*aloca os vetores normais*/
98  void freeNormalVectors(struct PlotSurface *Surface);
99  /*libera os vetores normais*/
100 void calcNormalVectors(struct PlotSurface *Surface);
101 /*calcula os vetores normais*/
102 void readFile(char *FileName, struct PlotSurface *Surface);
103 /*le o arquivo com os dados de heights e o X x Z*/
104 void prepareSurface
105 (char *FileName, struct PlotSurface *Surface);
106 /*prepara a supercie para plotar*/
107 void freeSurface(struct PlotSurface *Surface);
108 /*libera a memoria usada pela Superficie*/
109 void colorSurfaceSet(int i);
110 /*seleciona uma nova cor da superficie*/
111 void colorSurfaceSelect(GLfloat Height);
112 /*seleciona uma cor de acordo com a altura*/
113 void plotSurface(struct PlotSurface *Surface);
114 /*desenha a superficie */
115 void enterData();
116 /*dados ánecessrios ao controle do sistema*/
117 void init(void);
118 void display(void);
119 void reshape(int w, int h);
120 void keyboard(unsigned char key, int x, int y);
```

D.2 Graphics.c

```
1  #include "Graphics.h"
2
3  int main(int argc, char** argv)
4  {
5      /*struct PlotSurface Surface;*/
6      FrameNumber = 0; /*inicio da simulacao */
7      strcpy(DataFileBaseName, argv[1]);
8      printf("\nEntre com o delta da malha:");
9      scanf("%f", &DeltaR);
10     printf
11     ("\nEntre com o numero de frames existentes:");
12     scanf("%d", &FramesNumber);
13     glutInit(&argc, argv);
14     glutInitDisplayMode
15     (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
16     glutInitWindowSize (500, 500);
17     glutInitWindowPosition (100, 100);
18     glutCreateWindow (argv[0]);
19     init();
20     glutDisplayFunc(display);
21     glutReshapeFunc(reshape);
22     glutKeyboardFunc(keyboard);
23     glutMainLoop();
24     printf("\n\n%f!!!!", SurfaceToPlot->DeltaR);
25     freeSurface(SurfaceToPlot);
26     return 0;
27 }
28
29 void allocVectorHeights(struct PlotSurface *Surface)
30 {
31     int i;
32     if(Surface->NumberOfPointsX <= 0 ||
33     Surface->NumberOfPointsZ <=0)
34     {
```

```
35          printf("Dados␣inconsistentes");
36          exit(1);
37      }
38      Surface->Heights = (GLfloat **)
39      malloc(((Surface->NumberOfPointsX))*sizeof(GLfloat *));
40      if(Surface->Heights!=NULL)
41      {
42          for(i=0; i<=Surface->NumberOfPointsX; i++)
43          {
44              Surface->Heights[i] = (GLfloat *)
45              malloc(((Surface->NumberOfPointsZ))*
46              sizeof(GLfloat));
47              if(Surface->Heights[i]==NULL)
48              {
49                  printf("Erro␣de␣alocacao
50  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣de␣memoria");
51                  exit(1);
52              }
53          }
54      }
55      else
56      {
57          printf("Erro␣de␣alocacao␣de␣memoria");
58          exit(1);
59      }
60      return;
61  }
62  void freeVectorHeights(struct PlotSurface *Surface)
63  {
64      int i;
65      for(i=0; i<Surface->NumberOfPointsX; i++)
66      {
67          free(Surface->Heights[i]);
68      }
69      free(Surface->Heights);
```

```
70      Surface->Heights=NULL;
71      return;
72  }
73
74  void readFile(char *FileName,struct PlotSurface *Surface)
75  {
76      FILE *FileIn;
77      int i, j;
78
79      FileIn = fopen(FileName, "r");
80      if(FileIn==NULL)
81      {
82          printf("ãNo␣consegui␣abrir
83  ␣␣␣␣␣␣␣␣o␣arquivo␣de␣entrada!");
84          /*Teste de abertura do */
85          exit(1);
86          /*arquivo*/
87      }
88      fscanf(FileIn , "%d", &Surface->NumberOfPointsX);
89      fscanf(FileIn , "␣%d", &Surface->NumberOfPointsZ);
90
91      if(Surface->NumberOfPointsX <= 0 ||
92      Surface->NumberOfPointsZ <=0)
93      {
94          printf("Dados␣inconsistentes");
95          exit(1);
96      }
97
98      allocVectorHeights(Surface);
99
100     for(i=0; i<Surface->NumberOfPointsX; i++)
101     {
102         for(j=0; j<Surface->NumberOfPointsZ; j++)
103         {
104             fscanf(FileIn , "␣%f",
```

```
105                 &Surface->Heights[i][j]);
106             }
107         }
108         fclose(FileIn);
109         return;
110 }
111
112 void allocNormalVectors(struct PlotSurface *Surface)
113 {
114         int i, j;
115         if(Surface->NumberOfPointsX <= 0 ||
116         Surface->NumberOfPointsZ <=0)
117         {
118             printf("Dados inconsistentes");
119             exit(1);
120         }
121
122         Surface->NormalVectors = (GLfloat ***)
123         malloc((Surface->NumberOfPointsX - 1) * sizeof(GLfloat **));
124         if(Surface->NormalVectors!=NULL)
125         {
126             for(i=0; i<Surface->NumberOfPointsX - 1; i++){
127                 Surface->NormalVectors[i] = (GLfloat **)
128         malloc((Surface->NumberOfPointsZ - 1) * sizeof(GLfloat *));
129                 if(Surface->NormalVectors[i]!=NULL)
130                 {
131                     for(j=0; j<(Surface->NumberOfPointsZ - 1);j++)
132                     {
133                         Surface->NormalVectors[i][j] = (GLfloat *)
134                         malloc( 6 * sizeof(GLfloat));
135                         if(Surface->NormalVectors[i][j]==NULL)
136                     {
137                     printf("Nao consegui alocar memoria\n");
138                             exit(1);
139                         }
```

```
140                     }
141                 }
142                 else
143                 {
144                     printf("Nao consegui alcar memoria\n");
145                     exit(1);
146                 }
147             }
148         }
149         else
150         {
151             printf("Nao consegui alcar memoria\n");
152             exit(1);
153         }
154 }
155 void freeNormalVectors(struct PlotSurface *Surface)
156 {
157     int i, j;
158     for(i=0; i<Surface->NumberOfPointsX-1;i++)
159     {
160         for(j=0; j<(Surface->NumberOfPointsZ-1);j++)
161         {
162             free(Surface->NormalVectors[i][j]);
163         }
164         free(Surface->NormalVectors[i]);
165     }
166     free(Surface->NormalVectors);
167     Surface->NormalVectors=NULL;
168     return;
169 }
170 void calcNormalVectors(struct PlotSurface *Surface)
171 {
172     int i, j;
173     GLfloat v1[3], v2[3];
174     Surface->DeltaR = DeltaR;
```

```
175     for(i=0; i<(Surface->NumberOfPointsX - 1); i++)
176     {
177         for(j=0; j<(Surface->NumberOfPointsZ - 1); j++)
178         {
179             /*primeiro triangulo*/
180             v1[0]= - Surface->DeltaR;
181             v1[1]= 0;
182             v1[2]=Surface->Heights[i][j]-
183             Surface->Heights[i+1][j];
184             v2[0]=Surface->DeltaR;
185             v2[1]=0;
186             v2[2]=Surface->Heights[i+1][j] -
187             Surface->Heights[i][j+1];
188             /*produto vetorial*/
189             Surface->
190             NormalVectors[i][j][0]=v1[1]*v2[2]-v1[2]*v2[1];
191             Surface->
192             NormalVectors[i][j][1]=v1[2]*v2[0]-v1[0]*v2[2];
193             Surface->
194             NormalVectors[i][j][2]=v1[0]*v2[1]-v1[1]*v2[0];
195             /*segundo triangulo*/
196             v1[0]= 0;
197             v1[1]= 0;
198             v1[2]=Surface->
199             Heights[i+1][j]-Surface->Heights[i+1][j+1];
200             v2[0]=Surface->
201             DeltaR;
202             v2[1]=0;
203             v2[2]=Surface->
204             Heights[i+1][j+1] - Surface->Heights[i][j+1];
205             /*produto vetorial*/
206             Surface->
207             NormalVectors[i][j][3]=v1[1]*v2[2]-v1[2]*v2[1];
208             Surface->
209             NormalVectors[i][j][4]=v1[2]*v2[0]-v1[0]*v2[2];
```

```
210                 Surface->
211                 NormalVectors[i][j][5]=v1[0]*v2[1]-v1[1]*v2[0];
212             }
213         }
214         return;
215 }
216 void prepareSurface
217 (char *FileName, struct PlotSurface *Surface)
218 {
219         readFile(FileName, Surface);
220         allocNormalVectors(Surface);
221         calcNormalVectors(Surface);
222 }
223 void freeSurface(struct PlotSurface *Surface)
224 {
225         freeVectorHeights(Surface);
226         freeNormalVectors(Surface);
227         free(Surface);
228 }
229 void colorSurfaceSet(int i)
230 {
231 glMaterialfv
232 ( GL_FRONT_AND_BACK, GL_DIFFUSE, SurfaceDiffuseMaterial[i] );
233 glMaterialfv
234 (GL_FRONT_AND_BACK,GL_SPECULAR, SurfaceSpecularMaterial[i] );
235 glMaterialfv
236 ( GL_FRONT_AND_BACK, GL_SHININESS, SurfaceShininess[i] );
237
238 }
239 void colorSurfaceSelect(GLfloat Height)
240 {
241         if(Height<0.01)
242         {
243             colorSurfaceSet(0);
244         }
```

```
245     else
246     {
247         if (Height <0.05)
248         {
249             colorSurfaceSet (1);
250         }
251         else
252         {
253             if (Height <0.1)
254             {
255                 colorSurfaceSet (2);
256             }
257             else
258             {
259                 if (Height <0.2)
260                 {
261                     colorSurfaceSet (3);
262                 }
263                 else
264                 {
265                     if (Height <0.3)
266                     {
267                         colorSurfaceSet (4);
268                     }
269                     else
270                     {
271                         if (Height <0.5)
272                         {
273                             colorSurfaceSet (4);
274                         }
275                         else
276                         {
277                             colorSurfaceSet (5);
278                         }
279                     }
```

```
280                     }
281                 }
282             }
283         }
284 }
285 void plotSurface(struct PlotSurface *Surface)
286 {
287     int i, j;
288     for(i=0;i<Surface->NumberOfPointsX-1; i++)
289     {
290         for(j=0; j<Surface->NumberOfPointsZ-1;j++)
291         {
292             glNormal3f(Surface->NormalVectors[i][j][0],
293             Surface->NormalVectors[i][j][1],
294             Surface->NormalVectors[i][j][2]);
295             glBegin(GL_TRIANGLES);
296                 colorSurfaceSelect(Surface->Heights[i][j]);
297                 glVertex3f(i*Surface->DeltaR,
298                 Surface->Heights[i][j],j*Surface->DeltaR);
299                 colorSurfaceSelect(Surface->Heights[i+1][j]);
300                 glVertex3f((i+1)*Surface->DeltaR,
301                 Surface->Heights[i+1][j],j*Surface->DeltaR);
302                 colorSurfaceSelect(Surface->Heights[i][j+1]);
303                 glVertex3f(i*Surface->DeltaR,
304         Surface->Heights[i][j+1],(j+1)*Surface->DeltaR);
305             glEnd();
306             glNormal3f(Surface->NormalVectors[i][j][3],
307             Surface->NormalVectors[i][j][4],
308             Surface->NormalVectors[i][j][5]);
309             glBegin(GL_TRIANGLES);
310         colorSurfaceSelect(Surface->Heights[i+1][j]);
311                 glVertex3f((i+1)*Surface->DeltaR,
312                 Surface->Heights[i+1][j],j*Surface->DeltaR);
313         colorSurfaceSelect(Surface->Heights[i+1][j+1]);
314                 glVertex3f((i+1)*Surface->DeltaR,
```

```
315             Surface->Heights[i+1][j+1],(j+1)*Surface->DeltaR);
316                     colorSurfaceSelect
317                     (Surface->Heights[i][j+1]);
318                     glVertex3f(i*Surface->DeltaR,
319             Surface->Heights[i][j+1],(j+1)*Surface->DeltaR);
320                 glEnd();
321             }
322         }
323 }
324 void init(void)
325 {
326     glEnable(GL_LIGHTING);
327     glEnable(GL_LIGHT0);
328     glEnable(GL_DEPTH_TEST);
329     glEnable(GL_NORMALIZE);
330
331     glLightfv(GL_LIGHT0, GL_POSITION,  LightPosition);
332     glLightfv(GL_LIGHT0, GL_AMBIENT,  AmbientLight);
333     glLightfv(GL_LIGHT0, GL_DIFFUSE,  DiffuseLight);
334     glLightfv(GL_LIGHT0, GL_SPECULAR,  SpecularLight);
335 glLightModelfv(GL_LIGHT_MODEL_AMBIENT, LModelAmbient);
336
337     glClearColor (0.0, 0.0, 0.0, 0.0);
338     glShadeModel (GL_SMOOTH);
339 }
340
341 void display(void)
342 {
343     char FileName[80];
344     char Number[5];
345     SurfaceToPlot =
346 (struct PlotSurface *)
347 malloc(sizeof(struct PlotSurface));
348     sprintf(Number, "%d", FrameNumber);
349     strcpy(FileName, DataFileBaseName);
```

```
350         strcat(FileName, ".");
351         strcat(FileName,Number);
352         prepareSurface(FileName, SurfaceToPlot);
353
354         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
355
356         /*cria e posiciona a luz*/
357         glLightfv(GL_LIGHT0, GL_POSITION,  LightPosition);
358
359         /*faz a projecao*/
360         glMatrixMode(GL_PROJECTION);
361         glLoadIdentity();
362         gluPerspective
363 (FrustrumFOV, FrustrumASPECT, FrustrumNEAR, FrustrumFAR);
364
365         /*posiciona a camera*/
366         glMatrixMode(GL_MODELVIEW);
367         glLoadIdentity();
368         gluLookAt
369 (Camera[0], Camera[1], Camera[2],
370 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
371
372         /*desenha a superficie*/
373         glPushMatrix();
374         plotSurface(SurfaceToPlot);
375         glPopMatrix();
376
377         glutSwapBuffers();
378         freeSurface(SurfaceToPlot);
379 }
380
381 void reshape (int w, int h)
382 {
383    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
384    glMatrixMode(GL_PROJECTION);
```

```
385     glLoadIdentity();
386     gluPerspective
387     (FrustrumFOV, FrustrumASPECT,
388     FrustrumNEAR, FrustrumFAR);
389     /*    gluPerspective(60.0,
390     (GLfloat) w/(GLfloat) h, 1.0, 20.0); */
391     glMatrixMode(GL_MODELVIEW);
392     glLoadIdentity();
393     gluLookAt
394     (Camera[0], Camera[1],
395     Camera[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
396 }
397
398 /* ARGSUSED1 */
399 void keyboard (unsigned char key, int x, int y)
400 {
401    switch (key) {
402       case 'n':
403  if(FrameNumber>=0 && FrameNumber<FramesNumber-1)
404          {
405                FrameNumber++;
406                printf("\n%d", FrameNumber);
407                glutPostRedisplay();
408          }
409
410          break;
411       case 'p':
412  if(FrameNumber>0 && FrameNumber<FramesNumber)
413          {
414              FrameNumber--;
415              printf("\n%d", FrameNumber);
416              glutPostRedisplay();
417          }
418          break;
419       case 27: /*dev ser o ESC*/
```

```
420             exit(0);
421             break;
422         default:
423             break;
424     }
425 }
```

Apêndice E

Licença-GPL *General public license*

E.1 Considerações

Como qualquer outro sofware livre licenciado pela *GPL*, publicamos aqui também uma copia da licença para consulta de qualquer interessado. Foi escolhido registrar as bibliotecas como GPL devido a facilidade com que a licença pode ser conseguida e também para ajudar o meio de desenvolvedores de software livre, entre eles pesquisadores. Dessa maneira todos que desejarem podem usar as bibliotecas aqui descritas, do mesmo modo que usam a *iostream* do C++, é a mesma licença.

```
    GNU GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
    59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

    Preamble

The licenses for most software are designed to take away
your freedom to share and change it.  By contrast, the GNU
General Public License is intended to guarantee your freedom
to share and change free software--to make sure the software
is free for all its users.  This General Public License
applies to most of the Free Software Foundation's software
```

and to any other program whose authors commit
to using it. (Some other Free Software Foundation software is
covered by the GNU Library General Public License instead.)
You can apply it to your programs, too.

When we speak of free software, we are referring to freedom,
not price. Our General Public Licenses are designed to
make sure that you have the freedom to distribute copies of
free software (and charge for this service if you wish),
that you receive source code
or can get it if you want it, that you can change the software or
use pieces of it in new free programs; and that you know you can
do these things.

To protect your rights, we need to
make restrictions that forbid
anyone to deny you these rights or to ask you to surrender
the rights. These restrictions translate to
certain responsibilities for you if you distribute copies
of the software, or if you
modify it.

For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights
that you have. You must make sure that they, too, receive or can
get the source code. And you must show them these terms so they
know their rights.

We protect your rights with two steps: (1) copyright the
software, and (2) offer you this license which gives
you legal permission to copy,
distribute and/or modify the software.

Also, for each author's protection and ours,
we want to make certain that everyone understands that there

is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program).

Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on

the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; o

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium

customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or

distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any

such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Softwa Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author

to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITIN WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISIN OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show
> This is free software, and you are welcome to redistribute it
> under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriat

parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliografi a

[1] JEN, ERICA: *Aperiodicity in one-dimensional cellular automata* Theoritical Division , Los Alamos National Laboraty,Los Alamos-USA, 1990.

[2] FISH, ROBERT: *Cyclic cellular automata and related processes* Departament of Mathematics, Colby College ME-USA, 1990.

[3] WOLFRAM, S. *Theory and applications of cellular automata* World Scientif Press, Singapore,1986.

[4] CHOPARD, BASTIEN;DROZ MICHAEL: *Cellular automata modeling of phisical systems* Cambridge, Cambridge University Press, 1998. p. 256-280

[5] WOLFRAM, STEPHEN: *Cellular Automata and complexity* New York, Addison-Wesley Publishing Company, 1994. p. 1-17

[6] GLADROW, DIETER A.WOLF: *Lattice-Gas cellular automata and lattice boltzmann models* Heildelberg Germany, Springer Press, 2000. p. 1-22

[7] JESUS, RICARDO A. DE: *Aplicação de autômatos celulares na propagação de ondas*, Dissertação, São Paulo, 2002

[8] PATTERSON, A.R.: *A first course in fluid dynamics* Cambridge, Cambridge University Press 1983. p. 248-288

[9] LIGHTHILL, JAMES: *Waves in fluids* Cambridge, Cambridge University Press 1978. p.1-88

[10] WILDE, ANDREAS;SCHNEIDER, PETER: *Acoustic system simulation using the Transmission Line Matrix method*Dresden-Germany, 2002.

[11] COULSON, C.A.;JEFFREY, ALAN: *Waves: A mathematical approach to the commom types of wave motion* Longman Scientific & Technical.p.1-34

[12] MITCHEL, MELANIE: *An Introduction to Genetic Algorithms.* MIT Press, 1999.

[13] GOLDBERG, DAVID E.: *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison Wesley Longman, Inc, 1999.

[14] GEN, MITSUO;CHENG, RUNWEI: *Genetic Algorithms & Engineering Desing.* Wiley, 1997.

[15] TARANTOLA, ALBERT: *Inverse Theory*, Elsevier, 1987.

[16] CALLISTER, WILLIAM D. JR: *Materials Science and Engineering* John Wiley & Sons, 1999.

[17] FOLEY, JAMES D. : *Computer Graphics : principles and pratice* Addison-Wesley, 1997.

Bibliografia de Apoio para Desenvolvimento de Linguagem C/C++:

[18] PHOL, IRA: *C++ for C programmers* Addison Wesley, 2001

[19] SEDGWICK, ROBERT: *Algorithms in C++* Addison Wesley:2001

[20] KERNIGHAN, BRIAN W.;RITCHIE, DENNIS M.: *C: A Linguagem de Programação Padrão Ansi* Editora Campus, 1989.

[21] PRESS, WILIAM H.: *Numerical Recipes in C++:The Art of Scientific Computing* Cambridge, 2002.

[22] SCHILDT, HEBERT:*C Completo e Total* Makron Books, 1997

[23] ECKEL, BRUCE: *Thinking in C++* Prentice Hall

[24] JAMSA, KRIS;KLANDER LARS:*Programando em C/C++ "A Bíblia"* Makron Books, 1999.

[25] BUDD, TIMOTHY: *Data Structures in C++ using the Standart Teamplate Library* Addison Wesley, 1998.

Internet:

[26] Slackware Linux *www.slackware.com.* acesso em 2003.

[27] Red Hat Linux *www.redhat.com.* acesso em 2003.

[28] LaTeX 2$_\varepsilon$ *www.ctan.org*. acesso em 2003.

[29] MESA-3D opengl *www.mesa3d.org*. acesso em 2003.

[30] g++ (gcc) *gcc.gnu.org*. acesso em 2003.

[31] Java *www.javasoft.com*. acesso em 2003.