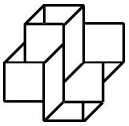


Laboratório Nacional de Computação Científica

Programa de Pós Graduação em Modelagem Computacional

Animação de Fluidos via Modelos do tipo *Lattice Gas* e *Lattice Boltzmann*

Por

Sicilia Ferreira Ponce Pasini Judice

PETRÓPOLIS, RJ - BRASIL

AGOSTO DE 2009

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

ANIMAÇÃO DE FLUIDOS VIA MODELOS DO TIPO *LATTICE GAS* E *LATTICE BOLTZMANN*

Sicilia Ferreira Ponce Pasini Judice

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM MODELAGEM COMPUTACIONAL

Aprovada por:

Prof. Gilson Antonio Giraldi, D.Sc
(Presidente)

Prof. Abimael Fernando Dourado Loula, D.Sc.

Prof. Jauvane Cavalcante de Oliveira, Ph.D

Prof. Antonio Alberto Fernandes de Oliveira, D.Sc.

PETRÓPOLIS, RJ - BRASIL
AGOSTO DE 2009

Judice, Sicilia Ferreira Ponce Pasini

J92a Animação de fluidos via modelos do tipo *lattice gas* e *lattice boltzmann* / Sicilia Ferreira Ponce Pasini Judice. Petropólis, RJ. : Laboratório Nacional de Computação Científica, 2009.

xvi, 127 p. : il.; 29 cm

Orientador: Gilson Antonio Giraldi

Dissertação (M.Sc.) – Laboratório Nacional de Computação Científica, 2009.

1. Animação por computador. 2. Autômato celular. 3. Dinâmica dos fluidos. 4. Processamento eletrônico de dados. I. Giraldi, Gilson Antonio. II. LNCC/MCT. III. Título.

CDD - 006.696

"Há um tempo em que é preciso abandonar as roupas usadas, que já tem a forma do nosso corpo, e esquecer os nossos caminhos, que nos levam sempre aos mesmos lugares. É o tempo da travessia: e, se não ousarmos fazê-la, teremos ficado, para sempre, à margem de nós mesmos."

Fernando Pessoa

Dedicatória

Aos meus pais, Rosangela e Newton, e aos meus irmãos, Giovanni e Hugo, por todo carinho e apoio!

Agradecimentos

Agradeço à SCC (Sociedade de Computação Científica), FAPERJ (Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro) e à FACC (Fundação de Apoio ao Desenvolvimento da Computação Científica) por todo apoio financeiro, sem o qual não seria possível a realização deste trabalho.

Agradeço também ao meu orientador, o professor Gilson Antonio Giraldi, por toda paciência e atenção. Assim como a todos do LNCC que me ajudaram de alguma forma, em especial aos amigos Bruno Barcellos, João Vicente Reis, Daniel Golbert e Adilson Xavier. Ao professor José Karam Filho pelas palavras certas nas horas certas. À Ana Neri, Ana Paula do Nascimento, Angela Garcia, Natália Fernandes e Bárbara de Andrade, por toda ajuda dispensada. Aos amigos que ganhei, em especial ao Marcelo Barros, Priscila Goliatt, Leonardo Goliatt, Eduardo Krempser, Paulo Trenhago e Jaqueline M. da Silva.

Não poderia deixar de agradecer a toda a minha família, em especial aos meus pais e a minha avó Henice Ferreira, fundamentais para minha formação. Agradeço também ao meu namorado Antônio Neto e a minha cunhada Sábatha Alves, pessoas especiais com quem divido não só a rotina, mas também as conquistas e derrotas do dia-a-dia. Aos meus sobrinhos Giulia e Enrico, por me fazerem rir nos momentos em que mais preciso. E a todos os meus amigos que compreendem a minha ausência e que fazem da presença momentos mais que especiais.

Resumo da Dissertação apresentada ao LNCC/MCT como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANIMAÇÃO DE FLUIDOS VIA MODELOS DO TIPO *LATTICE GAS* E *LATTICE BOLTZMANN*

Sicilia Ferreira Ponce Pasini Judice

Agosto , 2009

Orientador: Gilson Antonio Giraldi, D.Sc

Técnicas baseadas em física têm chamado a atenção da comunidade de computação gráfica, em especial para animação de fluidos (gás ou líquidos). As técnicas tradicionais para animação de fluidos são metodologias *top-down* baseadas em malhas $2D/3D$, tais como Diferenças Finitas e Elementos Finitos, em conjunto com equações de fluidos *Navier-Stokes*. Entretanto, tais métodos têm um custo computacional alto. Uma alternativa é o uso de técnicas baseadas em Autômatos Celulares do tipo *Lattice Gas* (LGCA) e o Método de *Lattice Boltzmann* (LBM). A idéia básica desses métodos consiste em obter a dinâmica macroscópica de um fluido a partir do comportamento coletivo de diversas partículas microscópicas. Em geral, tais metodologias *bottom-up* são eficientes do ponto de vista computacional. Neste trabalho, são estudados os aspectos teóricos e práticos da animação computacional de fluidos bidimensionais para computação gráfica, usando um método LGCA chamado FHP, e um método LBM chamado D2Q9. É proposto um modelo de fluido $3D$ baseado nos modelos bidimensionais FHP e D2Q9, bem como em métodos de interpolação. Em seguida, são apresentadas duas aplicações para animação de fluidos através dos métodos mencionados, uma para execução em tempo real e outra para execução *off-line*. Nos resultados dos experimentos computacionais são enfatizados a simplicidade e o potencial dos modelos propostos quando combinados com técnicas eficientes de *rendering*.

Abstract of Dissertation presented to LNCC/MCT as a partial fulfillment of the requirements for the degree of Master of Sciences (M.Sc.)

FLUID ANIMATION THROUGH LATTICE GAS AND LATTICE BOLTZMANN METHODS

Sicilia Ferreira Ponce Pasini Judice

August, 2009

Advisor: Gilson Antonio Giraldi, D.Sc

Physically-based techniques for the animation of fluids (gas or liquids) have taken the attention of the computer graphics community. The traditional fluid animation methods rely on a top down viewpoint that uses 2D/3D mesh based approaches motivated by the Eulerian methods of Finite Element (FE) and Finite Difference (FD), in conjunction with Navier-Stokes equations of fluids. Alternatively, lattice methods comprised by the Lattice Gas Cellular Automata (LGCA) and Lattice Boltzmann (LBM) can be used. The basic idea behind these methods is that the macroscopic dynamics of a fluid is the result of the collective behavior of many microscopic particles. Such bottom-up approaches need low computational resources for both the memory allocation and the computation itself. In this work, we consider animation of fluids for computer graphics applications, using a LGCA method called FHP, and a LBM method called D2Q9, both bidimensional models. We propose $3D$ fluid animation techniques based on the FHP and D2Q9 as well as interpolation methods. Then, we present two animating frameworks based on the mentioned lattice methods, one for a real time implementation and the other for an off-line implementation. In the experimental results we emphasize the simplicity and power of the presented models when combined with efficient techniques for rendering and compare their efficiency.

Sumário

Lista de Figuras

Figura

Lista de Tabelas

Tabela

Lista de Siglas e Abreviaturas

- D2Q9: modelo bidimensional do Método de *Lattice Boltzmann*
- DEM: Modelo Digital de Elevação
- DFC: Dinâmica de Fluidos Computacional
- DTM: Modelo Digital de Terreno
- EDP: Equação Diferencial Parcial
- FHP: modelo de autômato celular do tipo *Lattice Gas*
- FPS: *Frames* por Segundo
- GLUI: biblioteca de interface gráfica
- GLUT: biblioteca de funcionalidades para OpenGL
- GPU: *Graphics Processing Unit*
- HPP: modelo de autômato celular do tipo *Lattice Gas*
- LBM: Método de *Lattice Boltzmann*
- LGCA: Autômato Celular do Tipo *Lattice Gas*
- OpenGL: biblioteca gráfica
- PBRT: biblioteca de *rendering* baseada em modelos físicos
- SIMD: *Single Instruction, Multiple Data*

Capítulo 1

Introdução

Nas ultimas décadas, técnicas baseadas em modelos físicos para animação de elementos naturais, como fluidos (gás ou líquido), têm chamado a atenção dos pesquisadores em computação gráfica (Deusen et al, 2004; Guendelman et al, 2005; Nealen et al, 2006). A motivação reside no potencial existente nas aplicações dessas técnicas bem como na complexidade e beleza dos fenômenos naturais envolvidos. Em particular, técnicas no campo da Dinâmica de Fluidos Computacional (DFC) têm sido implementadas para animação de fluidos em aplicações que envolvem interações fluido-fluido e fluido-superfície (Müller et al, 2003), simuladores de cirurgia virtuais (Müller et al, 2004b), efeitos visuais para a indústria cinematográfica (Witting, 1999) e jogos digitais (Müller et al, 2004a; Monitzer, 2008).

As pesquisas em animação de fluidos se dividem basicamente em três etapas. Primeiramente, temos a busca de novos modelos em DFC que sejam mais eficientes do ponto de vista da computação gráfica, bem como o aprimoramento dos modelos já existentes. O próximo passo é a fase de *rendering* onde são aplicadas técnicas de computação gráfica sobre os campos gerados com o objetivo de criar efeitos visuais, tais como transparência e imagens refletidas na superfície de um líquido. Por fim, a terceira etapa envolve a criação de um aplicativo, onde as técnicas adotadas devem ser incorporadas a um *software*, com interfaces gráficas convenientes, tornando possível o uso destes recursos por artistas gráficos e animadores (ver os trabalhos de Witting (1999) e de Sucontphunt et al (2008) como exemplo).

Os métodos numéricos em DFC, tais como Diferenças Finitas e Elementos Finitos (Hughes, 1987), procuram descrever um sistema contínuo através da discretização das equações que o representam. Entretanto, tais métodos têm um custo computacional alto. Uma outra alternativa nesta área é o uso de técnicas baseadas em Autômatos Celulares do tipo *Lattice Gas* (LGCA) e o Método de *Lattice Boltzmann* (LBM) (Wolfram, 1994; Sarkar, 2000; Kari, 2005). Tais métodos buscam obter a dinâmica macroscópica do fluido através do comportamento coletivo de inúmeras partículas microscópicas. Técnicas baseadas em LGCA seguem esta idéia simplificando a dinâmica através de regras locais simples para as interações e deslocamento das partículas microscópicas. Em contrapartida, o método LBM constrói um modelo cinético simplificado (simplificação da Equação de *Boltzmann*), o qual incorpora a física microscópica de maneira que as propriedades macroscópicas obedeçam as equações desejadas (Chopard e Droz, 1998).

Enquanto a modelagem tradicional em DFC tenta representar um meio contínuo partindo de variáveis macroscópicas e Equações Diferenciais Parciais (EDPs), a modelagem via LGCA segue o caminho inverso, ou seja, a partir da descrição microscópica baseada em regras simples, procura-se obter o comportamento macroscópico (*bottom up*) sem, no entanto, resolver explicitamente nenhum sistema de EDPs (Chopard e Droz, 1998). O comportamento macroscópico é obtido via simulação computacional, baseada no conjunto de regras que regem o sistema, e interpolações simples.

Os LGCAs podem ser vistos como um universo simplificado. O espaço é representado por uma malha uniforme, onde cada célula contém dados *booleanos*, o passo de tempo é discreto e as leis que regem esse universo são expressas por simples regras de colisões locais. São modelos discretos, baseados em partículas cujo movimento está restrito às direções das arestas da malha (*lattice*). Tais métodos têm um custo computacional menor e são mais estáveis comparados aos tradicionais utilizados em simulação de fluidos, devido ao fato de não ser necessário calcular EDPs para se obter o nível desejado de realismo físico (Rothman e Zaleski, 1997).

Dentre a variedade de autômatos do tipo LGCA, optamos pelo modelo FHP pela sua simplicidade e eficiência computacional (Chopard e Droz, 1998). O FHP foi desenvolvido para simular fluidos bidimensionais. O espaço é discretizado em uma malha hexagonal (*lattice*), ou seja, em cada nó da *lattice* existem seis possíveis direções de movimento. Além disso, uma técnica multiescala foi aplicada com o intuito de demonstrar que o modelo FHP é capaz de reproduzir comportamentos *Navier-Stokes* para fluidos $2D$ (Frisch et al, 1986).

Os modelos LGCAs apresentam fraquezas inerentes a sua natureza discreta. Assim, os métodos *Lattice Boltzmann* foram propostos para corrigir alguns desses problemas, através do uso de variáveis reais no lugar das *booleanas*. O modelo LBM foi desenvolvido como uma técnica numérica alternativa e promissora para simulação de fluidos (McNamara e Zanetti, 1988). Este método não é baseado na discretização das equações que representam um sistema contínuo uma vez que o mesmo trabalha com uma versão simplificada da Equação de *Boltzmann* para obter o comportamento macroscópico desejado. McNamara e Zanetti (1988) propõe substituir os dados *booleanos* utilizados nos modelos LGCAs por números reais entre 0 e 1, representando probabilidade de presença de partículas em determinada direção de movimento. O método LBM pode ser usado para simular modelos $2D$ ou $3D$. Dentre os diversos modelos de LBM presentes na literatura (Chopard et al, 1998; Chen e Doolen, 1998; Wei et al, 2004) optamos por um modelo $2D$, conhecido como D2Q9, cujo espaço e tempo são discretos e para cada nó da *lattice* existem nove possíveis direções de movimento.

Uma vez resolvidas numericamente as equações de fluido, passa-se à fase de *rendering*, responsável pela geração de imagens com o nível de realismo desejado. Esta fase envolve tanto modelos físicos quanto conceitos de percepção visual. Sintetizar uma imagem com realismo é gerá-la através da definição dos objetos que a compõem, o que se dá a partir da geometria da cena, das informações dos materiais que constituem os objetos (texturas, coeficiente de reflexão, etc), das condições de iluminação ambiente e da posição de observação da cena (Rogers, 1998).

De acordo com o grau de realismo desejado, as técnicas de *rendering* podem ser classificadas em Realismo Físico e Foto Realismo (Adabala e Manohar, 2002). No Realismo Físico, propriedades físicas presentes na cena precisam ser levadas em conta na hora de gerar a imagem: as luzes presentes na cena, bem como suas intensidades e direções; características das superfícies representadas, como transparência, brilho, reflexão e textura; sombras; dentre outras propriedades. O Foto Realismo tem por objetivo a geração de imagens que sejam indistinguíveis da foto da cena representada, ou seja, imagens que contenham um realismo visual. Neste caso há a possibilidade de se utilizar técnicas intuitivas, sem a necessidade de resolver equações intrincadas provenientes de leis físicas. (Ferwerda, 2003).

Os métodos de *rendering* podem ser executados em tempo real ou não. Para execuções que não ocorrem em tempo real as técnicas de *rendering* são chamadas de *off-line* (Müller et al, 2008). Técnicas em tempo real são normalmente usadas para mídias interativas, assim como jogos digitais e simuladores. Nesse caso, o objetivo do método de *rendering* é manter uma taxa igual ou superior a 30 *frames* por segundo, deixando em segundo plano a preocupação com o foto realismo. Entretanto, técnicas *off-line* são voltadas para mídias não interativas assim como filmes e vídeos, onde o foco principal é a qualidade visual da cena e não o custo computacional do método usado (Müller et al, 2008). O tempo de renderização pode variar de alguns segundos até dias para as cenas mais complexas.

O presente trabalho propõe o uso de métodos do tipo *lattice* (LGCA e LBM) para animação de fluidos. Propomos um modelo de fluido $3D$ baseado nos modelos bidimensionais FHP e D2Q9, bem como em métodos de interpolação. Basicamente, nossa proposta consiste em definir um domínio $3D$ e regularmente distribuir planos independentes ao longo dos eixos da base (x e z). Cada um desses planos representam um modelo bidimensional, que evoluem de forma independente de acordo com as regras inerentes a cada um dos modelos FHP ou D2Q9. Finalmente, implementamos um modelo de interpolação com o intuito de obter um fluxo tridimensional através das simulações bidimensionais realizadas em cada um dos planos.

Como exemplo prático, propomos a aplicação desses métodos em dois casos distintos; um deles envolvendo animação em tempo real e outro animação *off-line*.

Para a aplicação em tempo real utilizamos os métodos apresentados para simular a dinâmica de uma chuva, um sistema de partículas para representar as gotas de chuva, e Modelos Digitais de Terreno (DTMs), sobre os quais a chuva interage. Um método de interação entre a chuva e o terreno também é proposto, fazendo com que a água alimente o fluxo superficial à medida que as partículas da chuva encontram o terreno. Para a visualização da animação em tempo real utilizamos os seguintes métodos de *rendering*: *Point Sprite* para renderizar as partículas da chuva; efeito *Fresnel* (baseado nas leis físicas de reflexão e refração) para visualização da superfície livre da água acumulada sobre o terreno; e *Texture Mapping* para visualizar o modelo digital de terreno. Utilizamos a técnica *Cube Mapping* para criar diversos cenários nos exemplos simulados, como um dia ensolarado, por exemplo.

Para a aplicação *off-line* utilizamos os métodos baseado em *lattice* para a simulação da dinâmica de um gás. Para a visualização dos *frames* gerados na simulação utilizamos uma técnica de *rendering* volumétrico, o *Volume Scattering*, que tem como base os métodos clássicos de traçado de raios (*Whitted's Ray Tracing*) (Whitted, 1980). Sua implementação está disponível na biblioteca PBRT (Pharr e Humphreys, 2004), um sistema de *rendering* baseado em física, que utilizamos neste trabalho.

Para cada uma das aplicações citadas foi desenvolvido um *software*, utilizando a linguagem de programação C/C++, e seguindo o paradigma de programação orientada a objetos. A visualização foi implementada através da OpenGL (Shreiner, 2004; Wright et al, 2007), uma biblioteca já tradicional em aplicações gráficas, e da biblioteca PBRT, baseada em física (Pharr e Humphreys, 2004). Utilizou-se também a biblioteca GLUT (Kilgard, 1996; Baker, 2006) em conjunto com a biblioteca GLUI (Stewart, 2006; Rademacher, 1999) para o desenvolvimento das interfaces gráficas.

Desta forma, as contribuições deste trabalho são: (1) Implementação de um modelo de fluido tridimensional a partir dos modelos bidimensionais FHP e D2Q9 e de técnicas de interpolação; (b) Combinação do modelo de fluido $3D$ com DTMs em uma aplicação em tempo real, utilizando diversas técnicas de *rendering* (*Point Sprite*, Efeito *Fresnel*, *Texture Mapping* e *Cube Mapping*); (3) Combinação do modelo de fluido $3D$ com uma técnica de *rendering* volumétrico (*Volume Scattering*) em uma aplicação *off-line*; (4) Desenvolvimento de aplicativos para facilitar a manipulação dos parâmetros das simulações e animações. Os resultados apresentados neste trabalho estão baseados nas referências (Judice e Giraldi, 2008a,b; Judice et al, 2008, 2009) publicadas utilizando-se as técnicas desenvolvidas.

O texto que segue está organizado da seguinte forma. No Capítulo 2 encontram-se as equações básicas utilizadas em Dinâmica dos Fluidos. No Capítulo 3 introduzimos os conceitos de Autômatos Celulares e apresentamos alguns exemplos. O Capítulo 4 traz uma breve descrição dos métodos tradicionalmente utilizados na modelagem de fluidos, bem como apresenta o FHP, um modelo de Autômato Celular do tipo Lattice-Gas, e o D2Q9, um modelo *Lattice Boltzmann* bidimensional, implementados neste projeto. No Capítulo 5 descrevemos as técnicas de *rendering* aplicadas. O Capítulo 6 descreve as contribuições deste trabalho bem como apresenta as metodologias de implementação dos aplicativos propostos. Finalmente, o Capítulo 7 traz os resultados obtidos e o Capítulo 8 apresenta as conclusões e os trabalhos futuros.

Capítulo 2

Equações de Dinâmica dos Fluidos

O estudo dos fluidos remonta aos trabalhos de **Arquimedes** (287 - 212 a.C.), um matemático, físico e inventor grego que contribuiu para a fundação do que hoje conhecemos como Dinâmica dos Fluidos. No trabalho intitulado *Tratado dos Corpos Flutuantes* ele estabeleceu as leis da Estática e Hidrostática.

Desde então, muitos nomes contribuíram para o desenvolvimento desta área. **Leonardo da Vinci** (1452 - 1519) usou sua observação apurada para descrever fluidos. **Evangeliston Torricelli** (1608 - 1647), discípulo de Galileu, inventou o barômetro, um dispositivo para medir as variações da pressão atmosférica. Os princípios do barômetro foram explorados pelo cientista e filósofo **Blaise Pascal** (1623 - 1662), formando a base para o entendimento da Estática dos Fluidos.

Foi no século XVIII que a base da Mecânica dos Fluidos foi fundada. Partindo dos fundamentos da mecânica clássica, **Sir Isaac Newton** (1642 - 1727), **Daniel Bernoulli** (1700 - 1782), **Jean le Rond d'Alembert** (1717 - 1783) e **Leonard Euler** (1707 - 1783) deram início ao desenvolvimento das equações fundamentais da Mecânica dos Fluidos relacionadas a conservação de massa, momento e energia. O princípio de conservação do momento tornou-se completo com os estudos independentes do francês **C. L. M. H. Navier** (1785 - 1836) e do inglês **Sir George Stokes** (1819 - 1903). As equações obtidas, conhecidas como **Equações de Navier-Stokes**, são o modelo matemático fundamental para a Mecânica dos Fluidos (Shaughnessy et al, 2005).

Desde então esse modelo matemático vem sendo aplicado com o intuito de solucionar diversos problemas da engenharia, física e indústrias de entretenimento (tais como jogos eletrônicos e filmes). O desenvolvimento dos computadores digitais foi de grande impacto nessa área, visto que tornou possível o surgimento da Dinâmica de Fluidos Computacional (DFC). Neste capítulo abordaremos as equações básicas que regem a Dinâmica dos Fluidos.

2.1 Fundamentos

Um dos primeiros conceitos aprendidos por nós é o de classificar os objetos que nos rodeiam em **sólidos** e **fluidos**, sendo o último dividido entre **líquidos** e **gasosos** (Pozrikidis, 2001).

Para a perspectiva da Dinâmica dos Fluidos, a diferença mais importante entre sólidos e fluidos é a facilidade com a qual eles se deformam. Um sólido tem uma forma definida, podendo sofrer alteração apenas quando há mudança nas condições externas. Por outro lado, os fluidos assumem a forma do recipiente onde estão inseridos. Como dito por Filho e Almeida (2003), um fluido se deforma continuamente quando submetido a uma tensão de cisalhamento. Um líquido apresenta uma superfície livre, enquanto o gás não, fazendo com que uma substância gasosa tende a preencher todo o recipiente que o contém (Pozrikidis, 2001; Filho e Almeida, 2003).

Em um nível microscópico, as moléculas encontram-se em constante vibração, o que descreve a energia interna da matéria e, conseqüentemente, determina seu estado. Para um objeto no estado sólido a energia interna é pequena o suficiente, de forma que as moléculas mantém suas posições relativas umas às outras, resultando em seu formato rígido. Em um líquido, entretanto, essa energia é de tal forma que permite certa mobilidade às moléculas, porém não o bastante para separá-las completamente. Esta separação ocorre no estado gasoso, onde as vibrações são fortes a ponto de permitir que as moléculas movam-se livremente (Neto et al, 2005).

Entretanto, quando se está preocupado com o comportamento do material em uma escala macroscópica, sua estrutura atômica não precisa ser levada em conta no modelo matemático. Para tanto, adota-se a **Hipótese do Contínuo**, onde as moléculas individuais são ignoradas e o sistema é considerado como um material contínuo. De acordo com Shaughnessy et al (2005), a Hipótese do Contínuo permite o uso de cálculo diferencial na modelagem e solução dos problemas de mecânica dos fluidos. Cada propriedade do fluido é considerada contínua em função de sua posição e do tempo. Com isso, propriedades físicas do fluido, como densidade e velocidade, podem ser descritas como campos escalares e vetoriais, ambos dependentes do tempo t, por exemplo $\rho(t, x)$ e $u(t, x)$, onde x representa um ponto no espaço (Neto et al, 2005).

Uma vez decidido entre trabalhar com uma abordagem contínua ou não, é necessário selecionar um método apropriado para descrever o fluido e seu comportamento. Segundo Shaughnessy et al (2005), tais métodos se dividem em três principais classificações, que são as descrições **Molecular**, **Lagrangiana** e **Euleriana**. A primeira é uma descrição que trabalha no nível molecular e provê uma base para uma teoria não-contínua de mecânica dos fluidos, enquanto que as duas últimas são abordagens contínuas.

Na Descrição Lagrangiana a entidade fundamental é uma partícula do fluido. Assume-se que o fluido consiste de uma distribuição contínua de pequenas partículas, sendo cada uma delas uma massa pontual. Tal método descreve o comportamento do fluido a partir dos movimentos de cada partícula, acompanhando-a em sua trajetória (Filho e Almeida, 2003; Shaughnessy et al, 2005).

Na Descrição Euleriana, a idéia é descrever o que acontece num ponto fixo ou numa dada região do espaço. Este é o método tradicional em mecânica dos fluidos (Filho e Almeida, 2003; Shaughnessy et al, 2005). A abordagem Euleriana constitui uma descrição do fluido baseada em campos em vez da descrição baseada em partículas utilizada nas abordagens Lagrangiana e Molecular. Na formulação Euleriana, cada propriedade do fluido tem um campo correspondente dependente

da posição e do tempo. Temos, por exemplo, campos vetoriais, tais como o campo de velocidades e campo de aceleração, e campos escalares de densidade, pressão e temperatura (Shaughnessy et al, 2005).

As leis de conservação serão apresentadas usando a formulação Euleriana. Sejam então $f : \Delta \subset \mathbb{R}^n \to \mathbb{R}$, $f = f(x_1, x_2, \ldots, x_n)$ uma função e $F : \Delta \subset \mathbb{R}^n \to \mathbb{R}$, $F(x_1, x_2, \ldots, x_n) = (F_1(x_1, x_2, \ldots, x_n), F_2(x_1, x_2, \ldots, x_n), \ldots, F_n(x_1, x_2, \ldots, x_n))$, um campo vetorial. Se f (respectivamente F_i, $i = 1, \ldots, n$) possui derivadas parciais até a ordem $r \geq 1$ contínuas em Δ, então dizemos que f é de classe C^r (respectivamente, F é de classe C^r). Assim, podemos definir os seguintes operadores:

- Gradiente:

$$\vec{\nabla} f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}). \tag{2.1}$$

- Divergente:

$$\vec{\nabla} \cdot f = \frac{\partial f}{\partial x_1} + \frac{\partial f}{\partial x_2} + \cdots + \frac{\partial f}{\partial x_n}. \tag{2.2}$$

- Laplaciano:

$$\Delta f = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \cdots + \frac{\partial^2 f}{\partial x_n^2}, \tag{2.3}$$
$$\Delta F = (\frac{\partial^2 F_1}{\partial x_1^2} + \frac{\partial^2 F_1}{\partial x_2^2} + \cdots + \frac{\partial^2 F_1}{\partial x_n^2}, \cdots, \frac{\partial^2 F_n}{\partial x_1^2} + \frac{\partial^2 F_n}{\partial x_2^2} + \cdots + \frac{\partial^2 F_n}{\partial x_n^2})$$

- Rotacional de F ($n = 3$):

$$\nabla \times F = (\frac{\partial F_3}{\partial x_2} - \frac{\partial F_2}{\partial x_3}, \frac{\partial F_1}{\partial x_3} - \frac{\partial F_3}{\partial x_1}, \frac{\partial F_2}{\partial x_1} - \frac{\partial F_1}{\partial x_2}). \tag{2.4}$$

- Diferencial de F:

$$DF(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix} \tag{2.5}$$

2.2 Leis de Conservação

Os princípios de conservação mais gerais são as leis de conservação da massa, do movimento e a primeira e segunda leis da termodinâmica (Filho e Almeida, 2003). Supondo que a matéria não pode ser destruída ou criada e desprezando fontes, a quantidade que entra num elemento de volume se iguala a quantidade que sai mais a quantidade acumulada no elemento. Tal princípio pode ser expresso em termos de taxas de transferência:

$$\textit{taxa de entrada} = \textit{taxa de saída} + \textit{taxa de acumulação}.$$

A seguir faremos uma revisão dos princípios de conservação da massa e do movimento. Desenvolveremos inicialmente as expressões gerais destas leis para campos escalares e vetoriais para em seguida aplicá-las ao caso específico da mecânica de fluidos.

2.2.1 Leis de Conservação para Campos Escalares

Seja U uma quantidade escalar por unidade de volume, definida em um volume arbitrário Ω, fixo no espaço e limitado por uma superfície fechada S. A variação da intensidade local de U ocorre devido à atuação do fluxo que expressa a contribuição do meio externo a Ω e das fontes Q.

A forma geral da lei de conservação para a quantidade escalar U pode ser escrita como (Hirsch, 1988):

$$\frac{\partial}{\partial t}\int\limits_{\Omega} U d\Omega + \oint \vec{F}\cdot d\vec{S} = \int\limits_{\Omega} Q_V d\Omega + \oint\limits_{S} \vec{Q}_S \cdot d\vec{S}. \tag{2.6}$$

Na equação (2.6), o primeiro termo à esquerda do sinal de igualdade representa a variação de U dentro do volume Ω por unidade de tempo. Tal variação deve ser igual à contribuição do meio externo devido ao fluxo através da superfície S, dada por:

$$-\oint \vec{F}\cdot d\vec{S}, \tag{2.7}$$

mais as contribuições das fontes volumétricas (Q_V) e superficiais (Q_S) de U, expressas respectivamente pelos termos à direita do sinal de igualdade na equação (2.6).

Supondo continuidade do fluxo e das fontes superficiais, pode-se usar o Teorema de Gauss, também conhecido como teorema da divergência, e reescrever a expressão (2.6) como:

$$\int_{\Omega} \frac{\partial U}{\partial t} d\Omega + \oint_{\Omega} \vec{\nabla} \cdot \vec{F} = \int_{\Omega} Q_V d\Omega + \oint_{\Omega} \vec{\nabla} \cdot \vec{Q}_S d\Omega. \tag{2.8}$$

A equação (2.8) está na forma integral da lei de conservação. A forma diferencial é obtida diretamente desta, supondo um volume qualquer, e é dada por (Giraldi et al, 2005):

$$\begin{aligned} &\frac{\partial U}{\partial t} + \vec{\nabla} \cdot \vec{F} = Q_V + \vec{\nabla} \cdot \vec{Q}_S \quad \Rightarrow \\ &\frac{\partial U}{\partial t} + \vec{\nabla} \cdot (\vec{F} - \vec{Q}_S) = Q_V. \end{aligned} \tag{2.9}$$

2.2.2 Leis de Conservação para Campos Vetoriais

Para o caso em que a quantidade conservada é descrita por uma quantidade vetorial $\vec{U}$, então o fluxo e o termo relativo às fontes superficiais se tornam tensores, $\bar{\bar{F}}$ e $\bar{\bar{Q}}_S$ respectivamente, e o termo correspondente às fontes volumétricas se torna um vetor $\vec{Q}_V$. As equações (2.8) e (2.9) assumem, respectivamente, a forma (Hirsch, 1988):

$$\int_{\Omega} \frac{\partial \vec{U}}{\partial t} d\Omega + \oint_{\Omega} \vec{\nabla} \cdot \bar{\bar{F}} d\Omega = \int_{\Omega} \vec{Q}_V d\Omega + \oint_{\Omega} \vec{\nabla} \cdot \bar{\bar{Q}}_S d\Omega \tag{2.10}$$

$$\frac{\partial \vec{U}}{\partial t} + \vec{\nabla} \cdot (\bar{\bar{F}} - \bar{\bar{Q}}_S) = \vec{Q}_V. \tag{2.11}$$

2.2.3 Equação de Conservação da Massa

A equação diferencial parcial que expressa a conservação de massa é também conhecida como a Equação de Continuidade.

Aplicando a lei de conservação dada pela equação (2.8) para o caso particular

em que U é a densidade volumétrica de massa (ρ), temos:

$$\int_{\Omega} \frac{\partial \rho}{\partial t} d\Omega + \oint_{\Omega} \vec{\nabla} \cdot (\rho \vec{v}) = \int_{\Omega} Q_V d\Omega + \oint_{\Omega} \vec{\nabla} \cdot \vec{Q}_S d\Omega, \tag{2.12}$$

onde $\vec{v}$ é a velocidade do fluido. Na forma diferencial, temos:

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v}) = Q_V + \vec{\nabla} \cdot \vec{Q}_S. \tag{2.13}$$

Na ausência de fontes, a equação de continuidade leva em conta somente a densidade e a velocidade do fluido. Pode ser aplicada a todos os fluidos, compressíveis e incompressíveis, Newtonianos e não-Newtonianos (Shaughnessy et al, 2005). Para fluidos incompressíveis a densidade é constante, ou seja, não é uma função nem das coordenadas espaciais nem do tempo. A equação (2.13) resulta em:

$$\vec{\nabla} \cdot \vec{v} = 0. \tag{2.14}$$

Outro caso particular ocorre quando a densidade é independente apenas do tempo, resultando em:

$$\vec{\nabla} \cdot (\rho \vec{v}) = 0. \tag{2.15}$$

2.2.4 Equação de Conservação do Momento

O **momento linear**, definido por ($\rho \vec{v}$), é uma quantidade vetorial e portanto sua lei de conservação tem a forma geral dada pelas equações (2.10) e (2.11). A questão central agora é determinar os tensores $\bar{\bar{F}}$ e $\bar{\bar{Q}}_S$ e o vetor $\vec{Q}_V$ que aparecem nessas equações (Giraldi et al, 2005; Xavier, 2006).

Pela segunda lei de Newton, sabe-se que os agentes responsáveis pela variação do momento linear de um sistema físico são as forças que agem no mesmo. Tais forças podem ser externas ou internas. Consideremos então uma porção de fluido limitada por uma superfície S.

Como forças externas, podemos ter campos de força tais como o campo

gravitacional e eletromagnético. Seja $\vec{f_e}$ a densidade volumétrica da resultante destas forças (Xavier, 2006). Além das forças externas ao sistema, temos também as forças de natureza interna decorrentes de deformações e tensões internas do mesmo. Neste texto, assume-se que o fluido é Newtoniano, ou seja, que as tensões internas são dadas pela expressão (Hirsch, 1988):

$$\bar{\bar{\sigma}} = -p\bar{\bar{I}} + \bar{\bar{\tau}}, \tag{2.16}$$

onde $\bar{\bar{I}}$ é um tensor unitário, p é a pressão hidrostática e $\bar{\bar{\tau}}$ é o tensor de tensão cujas componentes são dadas por (Hirsch, 1988):

$$\tau_{ij} = \nu[(\partial_i v_j + \partial_j v_i) - \frac{2}{3}(\vec{\nabla} \cdot \vec{v})\delta_{ij}], \tag{2.17}$$

sendo que ν é a viscosidade cinemática (Giraldi et al, 2005).

Integrando (2.16) ao longo de S tem-se a resultante das forças internas ao fluido que atuam sobre o volume Ω. Quanto ao tensor $\bar{\bar{F}}$ este será dado por (Hirsch, 1988):

$$\bar{\bar{F}} = \rho\vec{v} \otimes \vec{v}, \tag{2.18}$$

onde:

$$\vec{v} \otimes \vec{v} = \begin{pmatrix} v_1v_1 & v_1v_2 & v_1v_3 \\ v_2v_1 & v_2v_2 & v_2v_3 \\ v_3v_1 & v_3v_2 & v_3v_3 \end{pmatrix},$$

com $\vec{v} = (v_1, v_2, v_3)$.

Assim, a expressão (2.10) toma a forma:

$$\int_\Omega \frac{\partial(\rho\vec{v})}{\partial t} d\Omega + \oint_\Omega \vec{\nabla} \cdot (\rho\vec{v} \otimes \vec{v}) d\Omega = \int_\Omega \rho\bar{f}_e d\Omega + \oint_S \bar{\bar{\sigma}} d\vec{S}. \tag{2.19}$$

Substituindo $\bar{\bar{\sigma}}$ pela expressão (2.16) obtém-se a forma integral da lei de conservação do momento linear para um fluido Newtoniano.

A expressão diferencial correspondente é obtida diretamente da forma inte-

gral, sendo dada por (Hirsch, 1988):

$$\frac{\partial(\rho\vec{v})}{\partial t} + \vec{\nabla}\cdot(\rho\vec{v}\otimes\vec{v} + p\bar{\bar{I}} - \bar{\bar{\tau}}) = \rho\bar{f}_e. \tag{2.20}$$

Subtraindo a equação (2.13) do lado esquerdo da equação (2.20), multiplicada por $\vec{\rho}$, e supondo-se os termos fontes nulos, encontra-se:

$$\rho\frac{D\vec{v}}{Dt} = -\vec{\nabla}p + \vec{\nabla}\cdot(\bar{\bar{\tau}}) + \rho\bar{f}_e, \tag{2.21}$$

onde $D\vec{v}/Dt$ é a derivada material, dada por:

$$\frac{D\vec{v}}{Dt} = \frac{\partial\vec{v}}{\partial t} + \vec{v}\cdot\vec{\nabla}\vec{v}. \tag{2.22}$$

Quando a expressão (2.17) para o tensor de tensão de um fluido Newtoniano viscoso é substituída em (2.21) obtém-se as chamadas **Equações de Navier-Stokes**. Para coeficientes de viscosidade constantes, esta equação reduz-se a:

$$\rho\frac{D\vec{v}}{Dt} = -\vec{\nabla}p + \nu[\Delta\vec{v} + \frac{1}{3}\vec{\nabla}(\vec{\nabla}\cdot\vec{v})] + \rho\bar{f}_e. \tag{2.23}$$

Para fluidos incompressíveis ($\vec{\nabla}\cdot\vec{v} = 0$), a equação (2.23) reduz-se a:

$$\rho\frac{D\vec{v}}{Dt} = -\vec{\nabla}p + \nu\Delta\vec{v} + \rho\bar{f}_e. \tag{2.24}$$

No caso de um fluido ideal sem tensões internas, e portanto sem viscosidade $\nu = 0$, a equação (2.24) reduz-se à chamada **Equação de Euler**:

$$\rho\frac{D\vec{v}}{Dt} = \rho\frac{\partial\vec{v}}{\partial t} + \rho(\vec{v}\cdot\vec{\nabla})\vec{v} = -\vec{\nabla}p + \rho\bar{f}_e. \tag{2.25}$$

Portanto, encontramos as equações de conservação de massa e de Navier-

Stokes, dadas respectivamente por:

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v}) = Q_V + \vec{\nabla} \cdot \vec{Q}_S \tag{2.26}$$

$$\rho \frac{D\vec{v}}{Dt} = -\vec{\nabla} p + \nu[\Delta \vec{v} + \frac{1}{3}\vec{\nabla}(\vec{\nabla} \cdot \vec{v})] + \rho \bar{f}_e. \tag{2.27}$$

Tais equações formam a base para a descrição do comportamento de um fluido, particularmente para animações em computação gráfica, como veremos no Capítulo 4.

Capítulo 3

Autômatos Celulares

> *In Greek mythology, the machinery of the universe was the Gods themselves. They personally tugged the sun across the sky, delivered rain and thunder, and fed appropriate thoughts into human minds. In more recent conceptions, the universe is created complete with its operating mechanism: once set in motion, it runs by itself. God sits outside of it and can take delight in watching it* (Toffoli e Margolus, 1987).

Wolfram (1994) em sua coleção de artigos sobre Autômatos Celulares, especificamente no artigo intitulado *Cellular Automata* de 1983 (na página 411), introduz o assunto com uma reflexão acerca das leis básicas da física relevantes aos fenômenos do cotidiano. Muitas dessas leis envolvem sistemas complexos, compostos por inúmeros componentes, cada um deles obedecendo a regras simples, para que se possa obter o comportamento complexo do fenômeno estudado.

Para descobrir e analisar a base matemática de sistemas complexos é preciso identificar sistemas matemáticos simplificados que sejam capazes de capturar a essência do processo, e os Autômatos Celulares são fortes candidatos nessa questão (Wolfram, 1994).

Este capítulo tem como objetivo introduzir os Autômatos Celulares. A seção 3.1 conta um pouco sobre a origem dos autômatos bem como um breve histórico de suas aplicações. A seção 3.2 traz a definição formal e fala um pouco acerca das propriedades dos autômatos. A seção 3.4 apresenta alguns exemplos clássicos de Autômatos Celulares.

3.1 Origem e Breve Histórico

Os Autômatos Celulares são modelos matemáticos baseados em regras simples e locais capazes de gerar comportamentos complexos. Foram originalmente introduzidos por John von Neumann, sob o nome de **espaços celulares**, como uma idealização de sistemas biológicos, com o objetivo particular de modelar sistemas capazes de auto-reprodução (Wolfram, 1994; Chopard e Droz, 1998; Sarkar, 2000).

Em 1940 John von Neumann estava envolvido no planejamento dos primeiros computadores digitais. Seu objetivo era obter comportamentos complexos através de regras simples para interações espaciais e evolução temporal. Seria uma máquina análoga ao cérebro humano, cujas unidades de processamento e memória não fossem separadas uma das outras, massivamente paralela e capaz de auto-reprodução (Chopard e Droz, 1998).

Seguindo sugestões de S. Ulam, John von Neumann propôs um universo discreto fundamentado em uma *lattice* bidimensional, cujas células apresentam-se interconectadas localmente umas às outras. Cada célula é caracterizada por um estado interno, representando uma máquina de estado finito. Através de uma regra local de evolução, cada célula atualiza seu estado em função do seu próprio e dos estados de algumas células vizinhas. Todas as células da *lattice* evoluem de acordo com a mesma regra local, o que torna o sistema homogêneo, a exemplo de muitos sistemas físicos e biológicos (Chopard e Droz, 1998; Sarkar, 2000; Kari, 2005). Estes universos celulares propostos por John von Neumann são conhecidos atualmente como Autômatos Celulares (Chopard e Droz, 1998; Kari, 2005).

No autômato celular proposto por von Neumann (1966), para cada célula bidimensional da *lattice* era definida uma vizinhança formada pelos quatro vizinhos mais próximos (norte, sul, leste, oeste) e 29 possíveis estados para cada célula (Chattopadhyay et al, 2000). A teoria de Autômatos Celulares foi consolidada com o trabalho de Burks (1970) e passou por uma considerável simplificação no trabalho de Codd (1968).

As pesquisas de Wolfram (1994) tornaram o autor pioneiro no trabalho com Autômatos Celulares como modelos matemáticos para o estudo do fenômeno da auto-organização. Ele sugeriu o uso de autômatos unidimensionais, cada célula tendo como vizinhança as células da esquerda e da direita e um conjunto de dois possíveis estados (Wolfram, 1994).

Martin et al (1984) utilizou em seu trabalho ferramentas de álgebra polinomial para a dedução de caracterizações de autômatos celulares uniformes (regras idênticas aplicadas a cada célula do autômato) com fronteira periódica. Uma nova era de pesquisa teve início em 1992, com o trabalho sobre caracterização analítica do comportamento de autômatos baseada em ferramentas de álgebra matricial. A técnica proposta por Das et al (1992) é capaz de caracterizar autômatos celulares híbridos (diferentes regras aplicadas a diferentes células) com fronteira periódica ou nula.

Muitos trabalhos tiveram seus interesses voltados para aplicações de autômatos à computação e à biologia. Entretanto, nas últimas três décadas, observa-se também bastante interesse na área da física (Rothman e Zaleski, 1994; Xavier, 2006). Em particular, foram desenvolvidos modelos baseados em autômatos celulares para fluidos, como por exemplo o *Lattice Gas Cellular Automata* (LGCA) (Frisch et al, 1986), modelos eletromagnéticos, como o *Ising Spin Models*, modelos para fenômenos de difusão (Weimar et al, 1991; Weimar e Boon, 1994) (Chang et al, 2004).

De acordo com Wolfram (1994), sistemas físicos que contenham elementos discretos e cujas interações ocorram localmente podem ser modelados através do uso de Autômatos Celulares.

Embora sejam definidos por regras locais simples, autômatos podem exibir em uma escala ampla, comportamentos dinâmicos complexos. Tais regras podem ser vistas como uma simplificação da dinâmica microscópica, a qual reproduz o comportamento macroscópico esperado. Por exemplo, em uma escala microscópica cada célula pode representar uma partícula do sistema (Wolfram, 1994).

Diversos autores vêm direcionando suas linhas de pesquisa para a compreensão e aplicação dos Autômatos Celulares. Segundo Chopard e Droz (1998), algumas dessas linhas envolvem a utilização de modelos de autômatos para a simulação de sistemas físicos e biológicos. O sucesso obtido nessas aplicações reside no fato de que os Autômatos Celulares possuem diversas propriedades fundamentais do mundo físico: eles são massivamente paralelos, homogêneos e todas as interações ocorrem localmente. Outras propriedades físicas como reversibilidade e leis de conservação podem ser programadas através de uma escolha adequada da regra local de evolução (Toffoli e Margolus, 1987; Wolfram, 1994; Chopard e Droz, 1998).

3.2 Definição Formal

De uma maneira formal, um Autômato Celular é uma quádrupla $A = (L; S; N; f)$ onde L é um conjunto de índices ou células; S é o conjunto finito de estados, ou seja, dos possíveis valores que cada célula pode assumir; $N : L \rightarrow L^k$ é o mapeamento que define a vizinhança de cada célula i como uma coleção de k células; e $f : S^k \rightarrow S$ é a função de evolução ou a regra de atualização do autômato (Wolfram, 1994; Giraldi, 2005; Xavier, 2006).

A regra f é responsável pela dinâmica global do Autômato Celular e é aplicada a cada instante de tempo em todas as células. A coleção dos estados das células em qualquer passo de tempo é chamada de configuração ou estado global do autômato (Chopard e Droz, 1998; Xavier, 2006).

3.2.1 Vizinhanças

Uma regra de atualização f de um autômato é uma regra local, uma vez que só depende do estado de células vizinhas. A princípio não há restrição com relação ao tamanho de uma vizinhança. Na prática, entretanto, é comum utilizar somente células adjacentes (Wolfram, 1994; Chopard e Droz, 1998).

Para Autômatos Celulares, duas vizinhanças são consideradas: a **Vizinhança de von Neumann** e a **Vizinhança de Moore** (Chopard e Droz, 1998).

A Vizinhança de von Neumann mais simples consiste de uma célula central, que é exatamente a célula a ser atualizada pela regra f, e seus quatro vizinhos geográficos norte, sul, leste e oeste (Chopard e Droz, 1998). Generalizando, Wolfram (1994) define a Vizinhança de von Neumann de raio r da célula $\vec{x^0}$ como

$$N^v(\vec{x^0}) = \{\vec{x} : \|\vec{x} - \vec{x^0}\| \leq r\}, \tag{3.1}$$

onde $\|\cdot\|$ é a norma da soma.

No caso de Autômatos Celulares bidimensionais a definição (3.1) é dada por

$$N^v(x_1^0, x_2^0) = \{(x_1, x_2) : |x_1 - x_1^0| + |x_2 - x_2^0| \leq r\}. \tag{3.2}$$

A Figura 3.1 mostra uma Vizinhança de von Neumann para autômatos bidimensionais onde, em (a) temos $r = 1$ enquanto que em (b) temos $r = 2$.

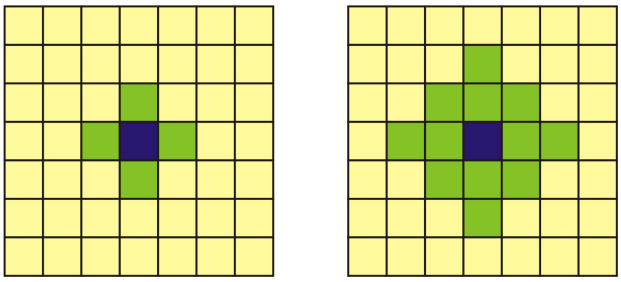


Figura 3.1: Vizinhanças: a) von Neumann de raio 1. b) von Neumann de raio 2. Fonte: Xavier (2006) p.19

Além dos quatro vizinhos geográficos citados na Vizinhança de von Neumann (norte, sul, leste, oeste), a Vizinhança de Moore contém também os quatro vizinhos das diagonais (norte-leste, norte-oeste, sul-leste, sul-oeste) (Chopard e Droz, 1998). Tal vizinhança de raio r para a célula $\vec{x^0}$ é definida por Wolfram (1994) como

$$N^v(\vec{x^0}) = \{\vec{x} : |x_i - x_i^0| \leq r, \quad i = 0, ..., d\}, \tag{3.3}$$

onde d é a dimensão do espaço e x_i é a i-ésima componente espacial.

Para os autômatos bidimensionais ($d = 2$) a definição (3.3) é dada por

$$N^v(x_1^0, x_2^0) = \{(x_1, x_2) : |x_1 - x_1^0| \leq r, |x_2 - x_2^0| \leq r\}. \tag{3.4}$$

A Figura 3.2 mostra uma Vizinhança de Moore para autômatos bidimensionais onde, em (a) temos $r = 1$ e em (b) temos $r = 2$.

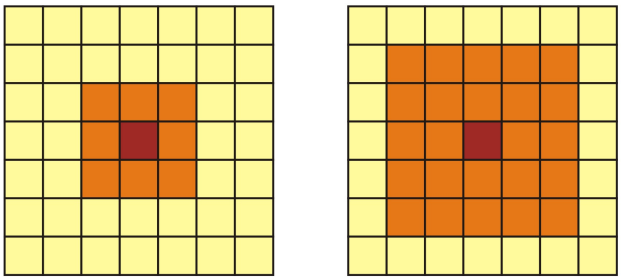


Figura 3.2: Vizinhanças: (a) Moore de raio 1. (b) Moore de raio 2. Fonte: Xavier (2006) p.19

3.2.2 Condição de Fronteira

Na prática se torna inviável trabalhar com uma *lattice* infinita. Uma vez definidos os limites da *lattice*, fica claro que as células presentes na fronteira não terão a mesma vizinhança que as células internas. Por isso se torna necessário identificar quais células pertencem à fronteira e aplicar a elas regras diferentes. Seguindo essa metodologia, é possível definir diversos tipos de fronteiras, cada uma com um comportamento diferente (Wolfram, 1994). Entretanto, ao invés de definir regras distintas para as células da fronteira, outra possibilidade é extender a vizinhança para essas células. Uma solução muito usada é a **condição de fronteira periódica**, a qual supõe que a *lattice* está inserida em uma topologia do tipo *torus*. Tal condição é obtida estendendo-se a *lattice* periodicamente, como mostrado na Figura 3.3. No caso de um autômato bidimensional, a condição de fronteira periódica assume que os lados direito e esquerdo estão conectados, assim como a parte superior e inferior da *lattice* (Chopard e Droz, 1998).

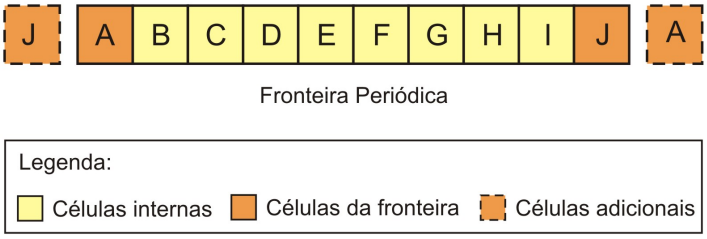


Figura 3.3: Condição de Fronteira Periódica para autômatos unidimensionais. Fonte: Chopard e Droz (1998) e Xavier (2006)

Outros tipos de condições de fronteira adicionam um conjunto de células ao longo da fronteira, como mostra a Figura 3.4 (a)-(c) (Chopard e Droz, 1998). A **condição de fronteira fixa** define valores pré-estabelecidos a essas novas células. A **condição de fronteira adiabática** duplica o valor das células que estão na fronteira para as células adicionais. A **condição de fronteira reflexiva** copia o valor da célula vizinha à célula da fronteira para a célula adicional.

Escolher que tipo de condição de fronteira usar dependerá da natureza do problema a ser modelado. Em alguns casos pode-se usar uma combinação de mais de um tipo de condição, como por exemplo na simulação de um canal longo, onde se pode usar fronteira periódica na horizontal e a reflexiva na vertical (Chopard e Droz, 1998; Xavier, 2006).

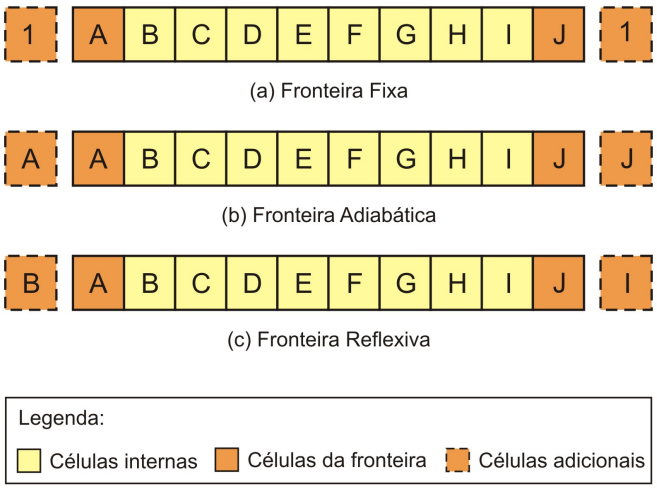


Figura 3.4: Condição de Fronteira: (a) Fixa. (b) Adiabática. (c) Reflexiva. Fonte: Chopard e Droz (1998)

3.3 Classificação

Um Autômato Celular unidimensional com o conjunto de possíveis estados definido por $S = \{0, 1\}$ e vizinhança de raio $r = 1$ é chamado de **elementar** (Kari, 2005). Nesta configuração existem $2^3 = 8$ combinações possíveis na vizinhança de uma dada célula, onde cada uma dessas combinações pode ser mapeada em 0 e 1. Desta forma, existem $2^8 = 256$ Autômatos Celulares Elementares distintos (Kari, 2005). Essas regras elementares foram estudadas e classificadas por Wolfram (1994), que introduziu uma metodologia para nomeá-las: cada uma dessas regras elementares é especificada por uma seqüencia de oito *bits*

$$f(111)\ f(110)\ f(101)\ f(100)\ f(011)\ f(010)\ f(001)\ f(000),$$

onde f é a regra de atualização do autômato. A seqüencia de *bits* é a expansão binária de um inteiro no intervalo de 0...255, chamada de **Nomenclatura de Wolfram** do Autômato Celular (Wolfram, 1994; Kari, 2005).

O valor numérico que identificará a regra é obtido combinando-se os valores resultantes da aplicação da regra de atualização f a cada uma das oito possíveis vizinhanças, formando assim um número binário. A representação desse número na base decimal fornece o valor numérico para a regra (Kari, 2005; Xavier, 2006). Por exemplo, a Figura 3.5 mostra o processo de identificação do autômato celular com $r = 1$, conhecido como Regra 90.

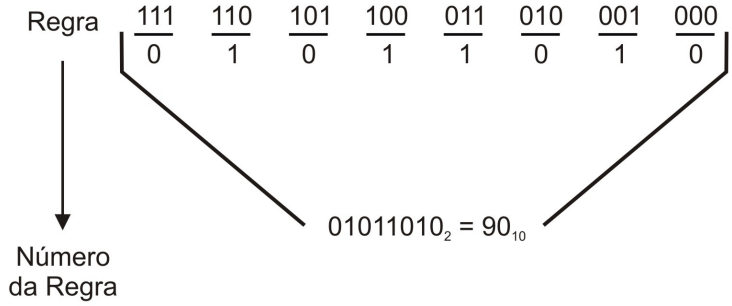


Figura 3.5: Aplicação da Nomenclatura de Wolfram. Fonte: Wolfram (1994) p.419

Exemplos de dinâmicas de Autômatos Celulares unidimensionais são freqüentemente representados em um diagrama espaço-temporal. Linhas horizontais são configurações consecutivas, e a linha superior é a configuração inicial (Kari, 2005). Por exemplo, a Figura 3.6 mostra diagramas da Regra 110 em duas escalas diferentes, onde o preto representa o estado 1 e o branco o estado 0 (Kari, 2005).

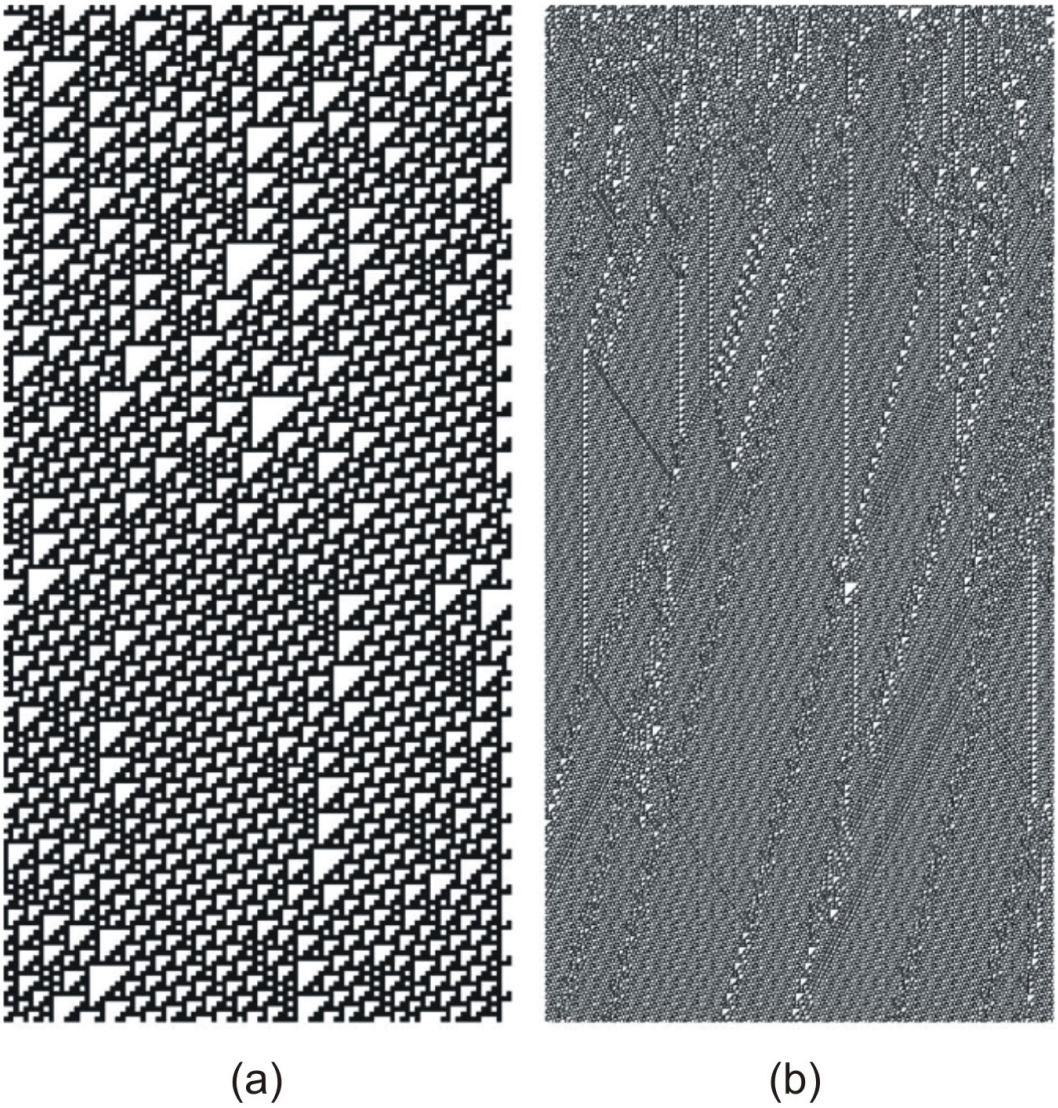


Figura 3.6: Diagramas espaço-temporal da Regra 110. Fonte: Kari (2005) p.9

Wolfram (1994) fez inúmeros experimentos com regras de Autômatos Celulares Elementares utilizando configurações iniciais aleatórias, e baseado nos tipos

de diagramas espaço-temporal observados, ele classificou as regras em quatro categorias (Wolfram, 1994; Chopard e Droz, 1998; Kari, 2005; Xavier, 2006), a saber:

- **Classe 1:** a evolução do autômato leva a um estado homogêneo na qual, por exemplo, todas as células têm valor 0 (Figura 3.7(a));
- **Classe 2:** a evolução do autômato leva a um conjunto de estruturas estáveis ou periódicas, que são simples e separadas (Figura 3.7(b));
- **Classe 3:** a evolução do autômato leva a um padrão caótico (Figura 3.7(c));
- **Classe 4:** a evolução do autômato leva a estruturas complexas (Figura 3.7(d)).

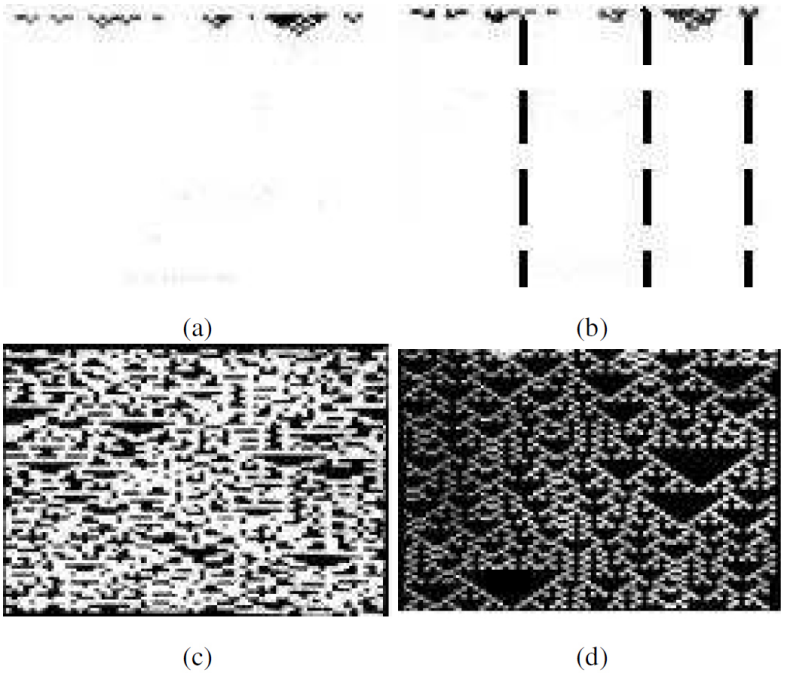


Figura 3.7: Diagramas espaço-temporal: alguns exemplos da classificação de Wolfram. (a) Classe 1. (b) Classe 2. (c) Classe 3. (d) Classe 4. Fonte: Giraldi (2005) p.71

A existência de somente quatro classes implica em uma considerável universalidade no comportamento dos autômatos celulares. Muitas características de um dado autômato dependem apenas da classe na qual ele pertence e não precisamente dos detalhes de sua evolução (Wolfram, 1994; Xavier, 2006).

3.4 Exemplos de Autômatos Celulares

Usando a nomenclatura tradicional (Sarkar, 2000), a cada passo de tempo, a coleção dos estados das células é chamada de **configuração** ou **estado global** do autômato. No passo inicial $t = 0$ tem-se a configuração inicial, e a medida que o tempo incrementa, a regra local é aplicada sobre todas as células, como (genericamente) representado na Figura 3.8 para o caso $1D$. Definições análogas são feitas para o caso $2D$.

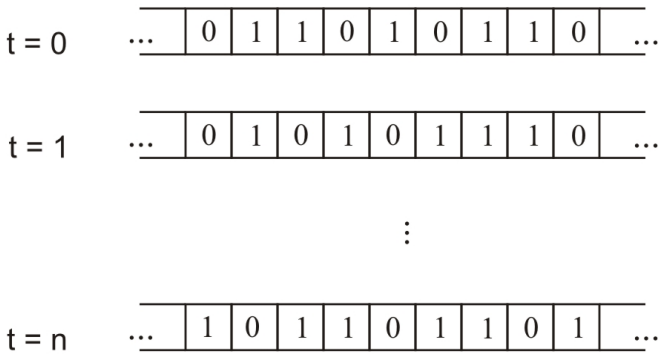


Figura 3.8: Evolução de um Autômato Celular 1D. Fonte: Sarkar (2000) p.82

3.4.1 Regra 90

O Autômato Celular Regra 90 consiste de um vetor unidimensional, cujo conjunto dos possíveis estados é formado por $S = \{0, 1\}$ e a regra de atualização f é a soma binária dos estados das células vizinhas (Wolfram, 1994; Xavier, 2006). Desta forma, dada uma célula x_i no instante t, seu estado a_i^t é definido por:

$$a_i^t = a_{i-1}^{t-1} \oplus a_{i+1}^{t-1}. \tag{3.5}$$

No topo da Figura 3.9 temos as 8 possíveis configurações para a vizinhança da célula a ser atualizada, e o resultado da soma binária efetuada. Logo abaixo podemos ver um diagrama espaço-temporal após 15 passos de evolução.

É interessante notar que, partindo de uma configuração inicial simples, onde somente uma célula inicial tem estado 1 e as demais têm estado 0, após os 15 passos já é possível observar a formação de um padrão não-trivial (Wolfram, 1994; Xavier, 2006).

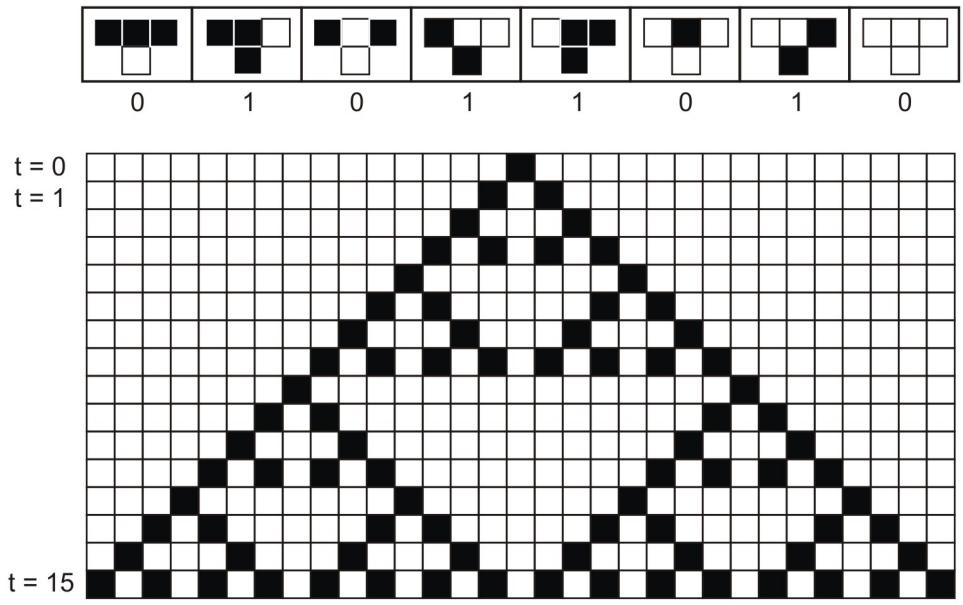


Figura 3.9: Autômato Celular Regra 90. Evolução após 15 passos de tempo. Fonte: Xavier (2006) p.18

3.4.2 O Jogo da Vida

Em 1970 o matemático John Conway propôs o famoso Jogo da Vida (*Game of Life*) (Gardner, 1970). Sua motivação era encontrar um autômato $2D$ que gerasse comportamentos complexos (Chopard e Droz, 1998). O Jogo da Vida é composto por uma *lattice* bidimensional, onde cada célula pode conter os valores 0 ou 1. Para esse conjunto de estados, Conway classificou a célula em viva, no caso de conter o valor 1, ou morta, para o valor 0. Este autômato utiliza a Vizinhança de Moore (Kari, 2005) explicada na seção 3.2.1.

O esquema de atualização f procura simular uma vida artificial, baseando-se nas seguintes regras (Chopard e Droz, 1998; Kari, 2005; Xavier, 2006):

- Uma célula cujo estado é 0 e que tenha em sua vizinhança exatamente 3 células vivas, passa para o estado 1;
- Uma célula cujo estado é 1 e que contenha em sua vizinhança menos de 2 células vivas, passa para o estado 0;
- Uma célula cujo estado é 1 e que tenha em sua vizinhança mais de 3 células vivas, passa para o estado 0.

A Figura 3.10 mostra um exemplo do Jogo da Vida em uma *lattice* 20×20.

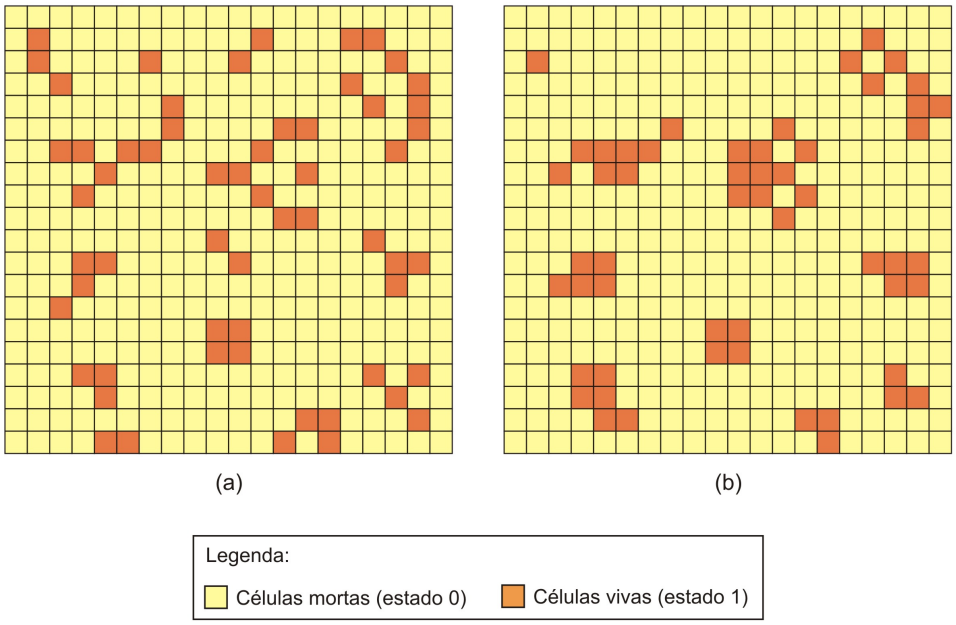


Figura 3.10: Exemplo de Jogo da Vida: (a) Instante inicial. (b) Passo 1.

O Jogo da Vida apresenta um comportamento bastante rico. Durante o processo de evolução do autômato, é possível notar o surgimento de estruturas complexas, como os chamados *gliders*, mostrados na Figura 3.11. Os *gliders* são configurações específicas das células, os quais possuem a propriedade de se locomoverem pelo espaço (Xavier, 2006). Podemos observar na Figura 3.11 que as configurações (e) e (f) são translações das configurações (a) e (b), respectivamente.

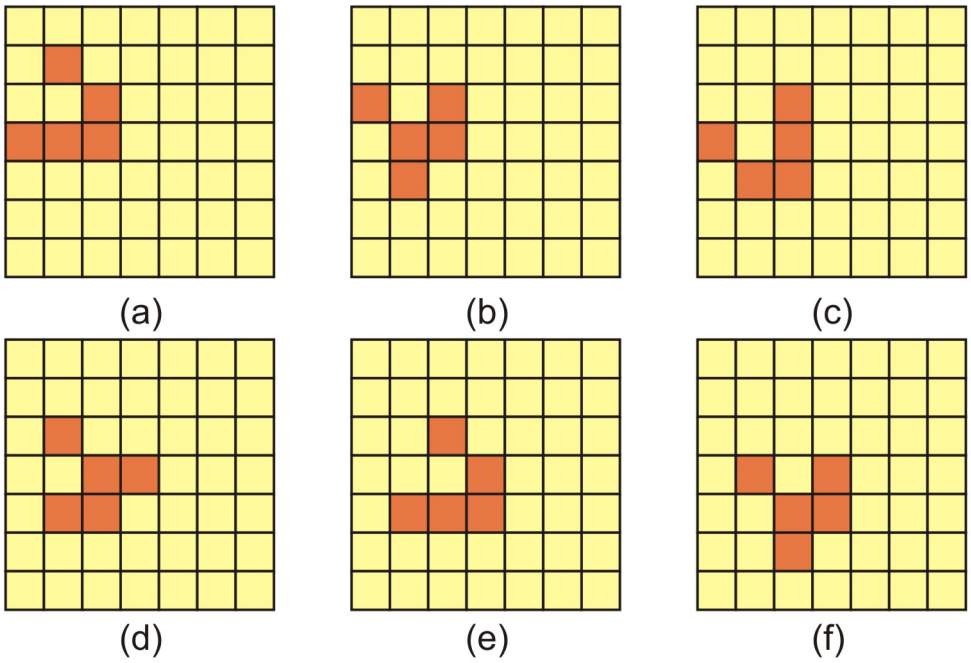


Figura 3.11: *Gliders* no Jogo da Vida: (a) Instante inicial. (b) Passo 1. (c) Passo 2. (d) Passo 3. (e) Passo 4. (f) Passo 5. Fonte: Xavier (2006) p.25

3.4.3 Modelo HPP

O modelo HPP faz parte de um grupo de Autômatos Celulares específicos, chamado *Lattice Gas*. Foi desenvolvido por Jean Hardy, Olivier de Pazzis e Yves Pomeau na década de 70. Sua proposta é modelar fluidos via autômatos celulares, utilizando regras simples e locais que imitam uma dinâmica de partículas. As características essenciais das interações microscópicas reais que são levadas em conta são as leis de conservação do momento linear e conservação do número de partículas (Chopard e Droz, 1998). Nascia então os Autômatos Celulares do tipo *Lattice Gas* (LGCAs).

O HPP é definido em uma *lattice* bidimensional e a idéia básica de sua dinâmica é trabalhar com partículas que se movem ao longo das direções da *lattice* (das arestas), seguindo regras apropriadas. O conjunto dos possíveis estados é composto por $S = \{0, 1\}$ e representa a ausência ou presença de partículas (Chopard e Droz, 1998; Xavier, 2006). A Figura 3.12 mostra um exemplo de uma configuração das partículas do HPP.

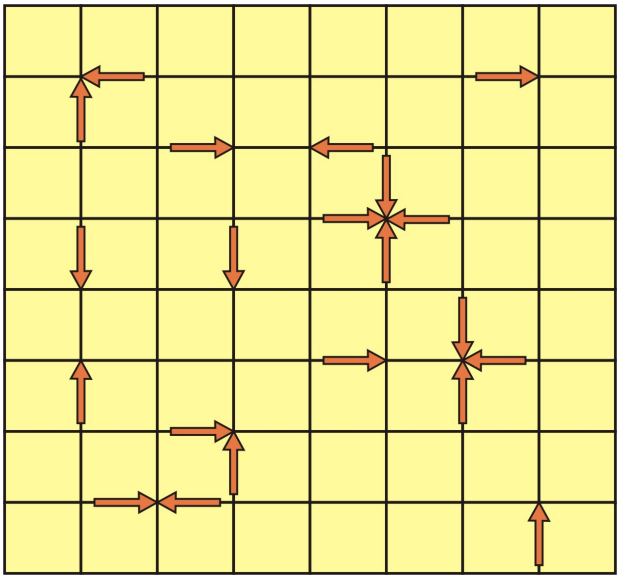


Figura 3.12: Exemplo de uma configuração das partículas do HPP. Fonte: Chopard e Droz (1998) p.39

Pela definição, Autômatos Celulares trabalham com um número finito de *bits* para representar o estado de cada célula. Portanto, para que a dinâmica molecular do HPP seja compatível com a dinâmica dos autômatos, é preciso que haja uma restrição da quantidade de partículas presentes simultaneamente em uma dada célula (Chopard e Droz, 1998).

Tal restrição é chamada de **Princípio de Exclusão** que garante que, dada uma célula da *lattice*, só pode haver no máximo uma partícula incidindo em uma determinada direção de movimento, em um determinado instante de tempo. Dessa forma, quatro *bits* de informação para cada célula são suficientes para se descrever o sistema durante sua evolução (Xavier, 2006).

A regra de atualização do HPP é composta por duas etapas: a colisão e a propagação (Chopard e Droz, 1998; Xavier, 2006). A **fase de colisão** especifica a interação entre partículas entrando no mesmo nó da *lattice*. É nessa fase que as partículas são "rearranjadas"em diferentes direções, com o intuito de assegurar que o princípio de exclusão seja satisfeito, como mostrado na Figura 3.13(b)-(c). Na **fase de propagação**, cada partícula move-se para a célula vizinha, na direção correspondente a sua direção de movimento, como mostrado na Figura 3.13(a).

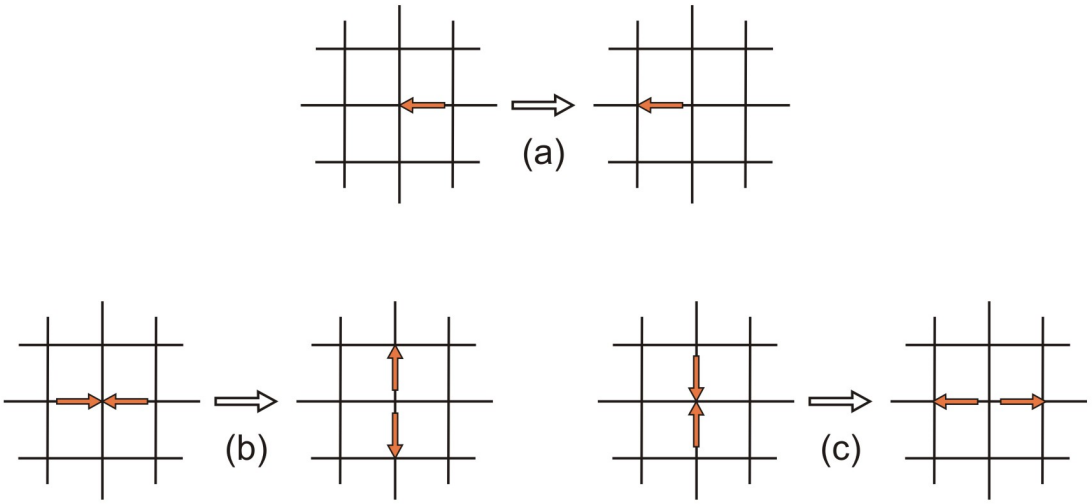


Figura 3.13: Regras do HPP: (a) Propagação. (b)-(c) Colisão. Fonte: Chopard e Droz (1998) p.40

O objetivo da regra do HPP é o de reproduzir certos aspectos das interações reais entre partículas, tais como a conservação do momento e da quantidade de partículas. Observando a Figura 3.13(b)-(c) podemos notar que durante o processo de colisão um par de partículas com momento nulo ao longo de uma dada direção é transformado em outro par de partículas, também com o momento nulo, movendo-se no eixo perpendicular (Chopard e Droz, 1998).

Outra característica importante no modelo HPP é a invariância sob reversão de tempo (reversibilidade). Em um dado instante, se as direções de movimento de todas as partículas forem revertidas, o sistema recuperará toda a sua história. Essa propriedade se torna um limitador quando a intenção é a de usar o modelo para descrever um fluido real, uma vez que neste caso há a presença de viscosidade. Outras limitações estão relacionadas a invariantes algébricos do reticulado adotado (Wolfram, 1994). Uma forma de tratar esses problemas mantendo a filosofia de modelagem via LGCA é dada pelo modelo FHP descrito no Capítulo 4.

Capítulo 4

Métodos para Simulação de Fluidos

Neste trabalho foram utilizados métodos para simulação de fluidos sob o ponto de vista da animação computacional. Os fluidos estão presentes em diversos fenômenos de nosso cotidiano, o que motivou a comunidade de pesquisadores em computação gráfica a utilizar tais métodos para animação.

Técnicas de simulação de fluidos foram desenvolvidas no campo da computação gráfica, onde tanto a abordagem Lagrangiana, não dependente de malhas, quanto a abordagem Euleriana, baseada em malhas, têm sido usadas (Neto et al, 2005). Nos últimos anos, a abordagem Euleriana tem sido mais popular para a simulação de fluidos em geral (Stam, 1999), simulação de água (Enright et al, 2002), e interação com objetos deformáveis (Nixon e Lobb, 2002).

Embora o objetivo principal da animação computacional não seja a precisão física dos fenômenos, é importante garantir estabilidade numérica, pois métodos instáveis geram artefatos visuais que comprometem o objetivo final (Xavier, 2006). O realismo físico e o custo computacional devem ser devidamente ponderados nestas aplicações, considerando-se que o objetivo da animação de fluidos é obter realismo visual.

Este capítulo descreve os métodos de simulação que serão usados nos capítulos seguintes. Uma revisão mais detalhada das técnicas nesta área pode ser encontrada no trabalho de Giraldi et al (2005). Iniciamos com o método de Diferenças Finitas que, apesar de não ser foco de interesse deste trabalho, é um método

tradicional em simulação de fluidos. Em seguida, apresentamos os modelos FHP e o D2Q9, utilizados neste trabalho.

4.1 Diferenças Finitas

Os modelos matemáticos oriundos das equações das seções 2.2.1-2.2.4 precisam ser discretizados se quisermos efetuar simulações numéricas. Conseqüentemente, o meio contínuo inicial do modelo é substituído por um conjunto discreto e finito de pontos no espaço e no tempo nos quais as derivadas presentes nas equações serão aproximadas (Giraldi et al, 2005).

Nos métodos tradicionais de simulação de fluidos, o conjunto discreto e finito de pontos é conectado seguindo uma topologia conveniente, dando origem a uma malha que pode ser de um dos três tipos, a saber:

- **Malha Estruturada:** cada ponto tem o mesmo número de vizinhos;
- **Malha Não-Estruturada:** o número de pontos vizinhos é variável;
- **Malha Mista:** parte da malha é estruturada e parte é não-estruturada.

O método de Diferenças Finitas é baseado nas propriedades da série de Taylor e na aplicação direta das definições de derivadas. É um método mais simples, embora tenha a desvantagem de requerer um alto grau de regularidade da malha. Fundamenta-se na representação de uma função suave f em série de Taylor, na vizinhança de um ponto x (Xavier, 2006):

$$f(x+h) = f(x) + f^{(1)}(x)h + \frac{1}{2!}f^{(2)}(x)h^2 + \frac{1}{3!}f^{(3)}(x)h^3 + \ldots, \qquad (4.1)$$

onde $f^{(i)}(x)$ representa a derivada de ordem i, calculada no ponto x.

A partir desta representação, é possível obter as expressões usuais para a aproximação da primeira derivada de f em x. Seja $x - x_0 = h$, com $h > 0$. Neste caso, relembrando a série de Taylor com resto (Chapra e Canale, 1988), tem-se:

$$f(x+h) = f(x) + f^{(1)}(x)h + \mathcal{O}(h^2), \qquad (4.2)$$

onde $\mathcal{O}(h^2)$ representa o resto decorrente do truncamento da série em (4.1), o qual tem a propriedade

$$\lim_{h\to 0} \frac{\mathcal{O}(h^2)}{h} = \lim_{h\to 0} \mathcal{O}(h) = 0. \tag{4.3}$$

Assim, escrevendo $x = x_0 + h$, e isolando $f^{(1)}(x)$ na expressão (4.2) pode-se escrever:

$$f^{(1)}(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h). \tag{4.4}$$

Desprezando-se o resto no segundo membro desta expressão, obtém-se a aproximação para $f^{(1)}(x)$ em diferenças finitas avançadas:

$$f^{(1)}(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{4.5}$$

De maneira semelhante pode-se, ainda, obter a aproximação de $f^{(1)}(x)$ em diferenças finitas retardadas:

$$f^{(1)}(x) \approx \frac{f(x) - f(x-h)}{h}. \tag{4.6}$$

Aproximações de ordem mais alta para a primeira derivada podem ser obtidas por argumentos semelhantes. De maneira geral, quando procuramos representar a $d-$ésima derivada $f^{(d)}(x)$ estamos procurando uma expressão do tipo:

$$\frac{h^d}{d!} f^{(d)}(x) + \mathcal{O}(h^{d+p}) = \sum_{i=i_{\min}}^{i_{\max}} C_i f(x+ih), \tag{4.7}$$

para alguma escolha dos extremos $i_{\min}$, $i_{\max}$ e dos coeficientes C_i, sendo p a ordem da aproximação desejada. Uma maneira elegante de obter tais quantidades é desenvolvida por Eberly (2003) e reproduzida abaixo.

Formalmente, pode-se escrever a série de Taylor para $f(x+ih)$ da forma:

$$f(x+ih) = \sum_{n=0}^{\infty} i^n \frac{h^n}{n!} f^{(n)}(x). \tag{4.8}$$

Substituindo esta expressão na equação (4.7), obtém-se:

$$\frac{h^d}{d!}f^{(d)}(x) + \mathcal{O}(h^{d+p}) = \sum_{i=i_{\min}}^{i_{\max}} C_i \left(\sum_{n=0}^{\infty} i^n \frac{h^n}{n!} f^{(n)}(x) \right), \tag{4.9}$$

a qual, trocando o ordem dos somatórios no segundo membro, assume a forma:

$$\frac{h^d}{d!}f^{(d)}(x) + \mathcal{O}(h^{d+p}) = \sum_{n=0}^{\infty} \left(\sum_{i=i_{\min}}^{i_{\max}} i^n C_i \right) \frac{h^n}{n!} f^{(n)}(x). \tag{4.10}$$

Finalmente, pode-se isolar a $d-$ésima derivada no primeiro membro dividindo a equação por $h^d/d!$ e usando a série de Taylor com resto:

$$f^{(d)}(x) = \frac{d!}{h^d} \sum_{n=0}^{d+p-1} \left(\sum_{i=i_{\min}}^{i_{\max}} i^n C_i \right) \frac{h^n}{n!} f^{(n)}(x) + \mathcal{O}(h^p), \tag{4.11}$$

onde usou-se o fato que:

$$\frac{d!}{h^d}\mathcal{O}(h^{d+p}) = \mathcal{O}(h^p)$$

Para que a equação (4.11) seja satisfeita, é suficiente que:

$$\sum_{i=i_{\min}}^{i_{\max}} i^n C_i = \begin{cases} 0, & \text{se } 0 \leq n \leq d = p-1 \quad \text{e} \quad n \neq d, \\ 1, & \text{se } n = d. \end{cases} . \tag{4.12}$$

Este é um sistema formado por $d+p$ equações lineares com $i_{\min} + i_{\max} + 1$ incógnitas. Impondo $d + p = i_{\min} + i_{\max} + 1$ tem-se solução única.

4.2 Modelo FHP

O modelo FHP foi introduzido por Frisch et al (1986), sendo mais um tipo de LGCA (vide seção 3.4.3). Embora os LGCAs não possam competir com técnicas tradicionais de dinâmicas de fluido computacional, particularmente para número de Reynolds alto, vários autores tiveram sucesso ao usar tais modelos para simular sistemas complexos, para os quais as técnicas tradicionais são de difícil aplicação (Chopard et al, 1998) (meios porosos (Chen et al, 1991), meios granulares (Károlyi

et al, 1994), dentre outros (Boghosian et al, 1996)). Além disto, a partir da análise da microdinâmica do FHP, através de técnicas multiescala, é possível obter as equações tradicionais da dinâmica de fluidos em escala macroscópica (Frisch et al, 1986). Estes fatos motivam o estudo do FHP como uma metodologia alternativa para simulação de fluidos.

O FHP é um modelo bidimensional e pode ser visto como uma abstração, em uma escala microscópica, de um fluido. O FHP descreve o movimento/interação de partículas em um espaço discretizado em uma *lattice* hexagonal, como visto na Figura 4.1.

A microdinâmica do FHP é dada em termos de variáveis *booleanas* que descrevem o número de ocupação em cada nó da *lattice* a cada passo de interação (i.e. a presença ou não de partícula). As partículas se movem em passos de tempo discretos, com uma velocidade constante em módulo, apontando ao longo de uma das seis direções da *lattice*. Não mais do que uma partícula pode se deslocar para um mesmo nó da lattice, num dado instante, em uma dada direção. Tal restrição é chamada de **Princípio de Exclusão**, o que garante que seis variáveis *booleanas* para cada nó da *lattice* são suficientes para representar a microdinâmica.

Cada partícula se movimenta com uma velocidade constante em módulo, de tal forma que, a cada interação, ela percorre uma aresta da *lattice* e alcança o nó vizinho. Na falta de colisões, as partículas mantém o movimento ao longo da direção especificada por seu vetor velocidade. As colisões ocorrem quando há partículas entrando num mesmo nó no mesmo instante, e resultam em uma nova distribuição local de velocidades das partículas.

Quando exatamente duas partículas incidem em um mesmo nó com velocidades opostas, ambas são desviadas em um ângulo de 60° de forma que, após a colisão, tenha-se uma nova configuração também com momento nulo. Tal desvio pode ocorrer no sentido horário ou anti-horário, como mostra a Figura 4.1. Por razões de simetria, as duas possibilidades são escolhidas aleatoriamente, com igual probabilidade.

Quando exatamente três partículas com velocidades formando um ângulo de 120° colidem, cada uma delas retorna, na direção da aresta inicial, como mostra a Figura 4.1. Desta forma, o momento se mantém nulo, sendo portanto conservado. Para as demais configurações as partículas continuam o seu movimento como se não houvesse colisão.

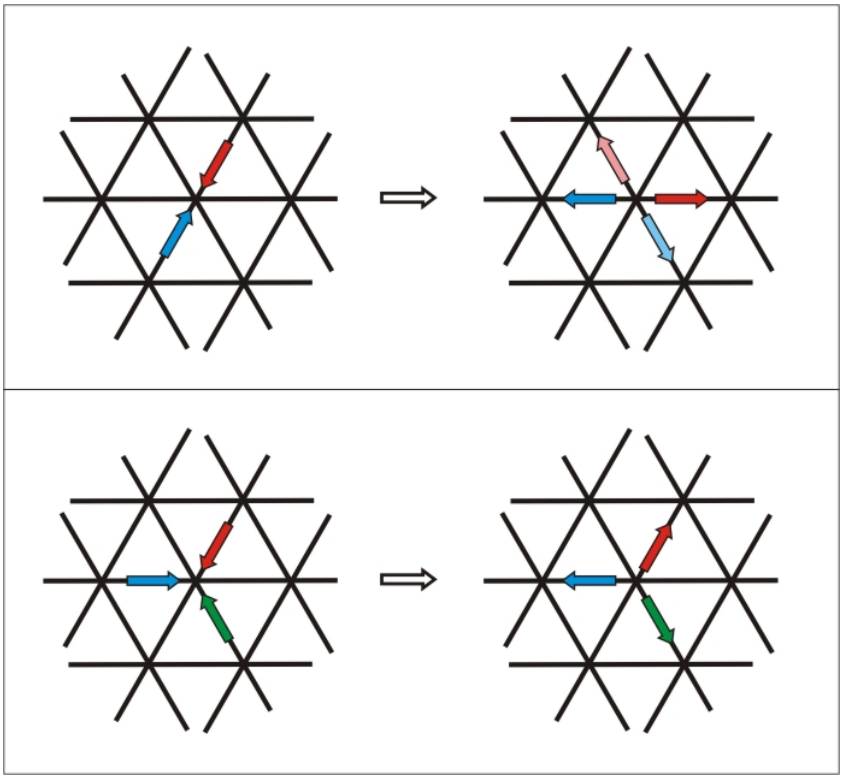


Figura 4.1: Colisões de partículas no modelo FHP. Fonte: Xavier (2006) p.42

4.2.1 A Microdinâmica do FHP

A microdinâmica completa do modelo FHP pode ser expressa por uma equação de evolução para os números de ocupação, definido como o número $n_i(\vec{x}, t)$ de partículas entrando no nó $\vec{x}$ no instante t com velocidade na direção $\vec{c}_i$ (Chopard e Droz, 1998):

$$\vec{c}_i = \left(\cos\left(\frac{2\pi i}{6}\right) , \sin\left(\frac{2\pi i}{6}\right)\right), \tag{4.13}$$

onde $i = 1, ..., 6$ representa as seis direções possíveis para cada nó da *lattice*, como mostra a Figura 4.2.

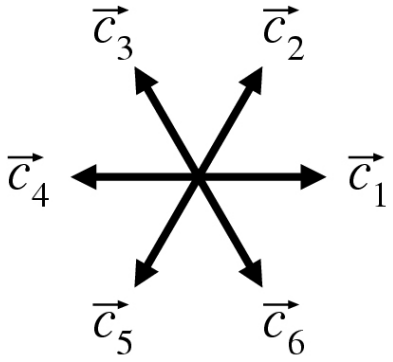


Figura 4.2: Direções de movimento na *lattice* do FHP. Fonte: Xavier (2006) p.43

Os números n_i podem ser 0 ou 1, ou seja, presença ou não de partícula na direção $\vec{c}_i$ da *lattice*. Define-se também o passo de tempo como Δ_t e o deslocamento entre os nós da *lattice* como Δ_x. Assim, as seis possíveis velocidades $\vec{v}_i$ das partículas estão relacionadas às suas direções de movimento por:

$$\vec{v}_i \;=\; \frac{\Delta_x}{\Delta_t}\vec{c}_i. \tag{4.14}$$

Não ocorrendo nenhuma interação entre as partículas, a equação de evolução para n_i pode ser escrita como (Chopard et al, 1998):

$$n_i(\vec{x} + \Delta_x\vec{c}_i,\; t + \Delta_t) \;=\; n_i(\vec{x}, t). \tag{4.15}$$

A equação (4.15) diz que uma partícula entrando no nó $\vec{x}$ com velocidade ao longo de $\vec{c}_i$ continuará em linha reta, de tal forma que, no instante seguinte ela entrará no nó $\vec{x} + \Delta_x\vec{c}_i$ com a mesma direção de movimento (Chopard et al, 1998). Entretanto, quando ocorre uma colisão, uma partícula pode ser removida de sua direção original ou pode ser desviada para uma outra. Vejamos o exemplo da colisão descrita na Figura 4.1, onde exatamente duas partículas concorrem ao mesmo nó com velocidades opostas (topo da figura). Neste caso, antes da colisão, somente n_2 e n_5 possuem valor 1 no nó $\vec{x}$. Após a colisão, temos duas possibilidades: n_1 e n_4 ou n_3 e n_6 terão valor 1.

Generalizando para a colisão entre duas partículas com velocidades opostas, a partícula que se move com velocidade $\vec{v}_i$ antes da colisão se moverá com velocidade $\vec{v}_{i-1}$ ou $\vec{v}_{i+1}$, após a colisão, onde $i = 1, \ldots, 6$ (note que as operações sobre o índice i devem retornar valores entre 1 e 6). A quantidade binária:

$$D_i = n_i n_{i+3}(1 - n_{i+1})(1 - n_{i+2})(1 - n_{i+4})(1 - n_{i+5}) \tag{4.16}$$

indica que uma colisão ocorrerá quando $D_i = 1$.

De fato, se existir uma partícula na direção $\vec{c}_i$ e uma na direção $\vec{c}_{i+3}$ o termo $n_i n_{i+3}$ será 1. Assim basta que não haja nenhuma partícula nas outras direções para garantir $D_i = 1$. Logo $n_i - D_i$ é o número de partículas resultantes na direção $\vec{c}_i$ devido a uma colisão de duas partículas ao longo desta direção (Xavier, 2006).

Contudo, quando $n_i = 0$, uma nova partícula pode aparecer na direção $\vec{c}_i$, como resultado de uma colisão entre n_{i+1} e n_{i+4} ou uma colisão entre n_{i-1} e n_{i+2}. Neste ponto, torna-se conveniente a introdução de uma variável *booleana* aleatória $q(\vec{x}, t)$, cujo valor é usado para decidir se uma partícula será refletida para a direita ($q = 1$) ou para a esquerda ($q = 0$). Dessa forma, o número de partículas criadas na direção $\vec{c}_i$ é dado por (Xavier, 2006):

$$qD_{i-1} + (1 - q)D_{i+1}. \tag{4.17}$$

Para o caso mostrado na Figura 4.1 de uma colisão entre três partículas, estas podem ser espalhadas, com uma delas ocupando a direção $\vec{c}_i$. A quantidade que expressa a ocorrência de uma colisão entre três partículas n_i, n_{i+2} e n_{i+4} pode ser obtida seguindo os mesmos princípios acima, sendo dada por (Chopard e Droz, 1998; Xavier, 2006):

$$T_i = n_i n_{i+2} n_{i+4}(1 - n_{i+1})(1 - n_{i+3})(1 - n_{i+5}). \tag{4.18}$$

Análogo ao caso anterior, o resultado de uma colisão entre três partículas é modificar o número de partículas na direção $\vec{c}_i$. Dessa forma, o número de partículas resultantes na direção $\vec{c}_i$, considerando o desaparecimento de uma partícula devido a colisão na posição i ou o surgimento de uma partícula devido a colisão na posição $i+3$ pode ser expresso por (Chopard et al, 1998; Xavier, 2006):

$$n_i - T_i + T_{i+3}. \tag{4.19}$$

Uma vez estipuladas as regras de colisão, podemos reescrever a equação (4.15) para a microdinâmica do FHP como:

$$n_i(\vec{x} + \Delta_x \vec{c}_i,\ t + \Delta_t) = n_i(\vec{x}, t) + \Omega_i(n(\vec{x}, t)), \tag{4.20}$$

onde Ω_i é o termo de colisão, definido a partir das expressões (4.17) e (4.19):

$$\Omega_i = -D_i + qD_{i-1} + (1-q)D_{i+1} - T_i + T_{i+3}. \tag{4.21}$$

A equação (4.20) pode ser facilmente implementada e, por sua vez, o modelo FHP pode ser eficientemente simulado em um computador. Operadores de colisão mais elaborados podem ser criados via inclusão de colisões entre quatro partículas ou mesmo através da inclusão de uma partícula em repouso (Frisch et al, 1987). Em geral, a única restrição imposta é que o operador de colisão Ω_i conserve a massa $\sum_{i=1}^{6} \Omega_i = 0$ e o momento $\sum_{i=1}^{6} \vec{v}_i \Omega_i = 0$ (Chopard et al, 1998; Xavier, 2006).

4.2.2 As Variáveis Macroscópicas

As quantidades físicas de interesse não são as variáveis *booleanas* n_i, mas sim as quantidades macroscópicas ou os valores médios, tais como a densidade e momento linear em um ponto do sistema (Chopard e Droz, 1998). Seja então a distribuição:

$$N_i(\vec{x}, t) = \langle n_i(\vec{x}, t) \rangle, \quad i = 1, \ldots, 6. \tag{4.22}$$

que corresponde à probabilidade de se ter uma partícula no nó $\vec{x}$, no instante t, com velocidade $\vec{v_i}$ definida pela equação (4.14) (Chopard et al, 1998).

No geral, um modelo LGCA é caracterizado pelo número z de direções para cada nó da *lattice* e a dimensão espacial d. Para uma *lattice* retangular com $d = 2$, temos $z = 4$, enquanto que para uma *lattice* hexagonal temos $z = 6$. Em alguns modelos há o acréscimo de uma direção ($\vec{c_0}$), onde $n_0 \in \{0, 1\}$ será o número de partículas em repouso ($\vec{v_0} = 0$) (Chopard e Droz, 1998).

Seguindo a definição usual de mecânica estatística (Chopard e Droz, 1998), a densidade local de partículas é a soma das médias das variáveis microscópicas de ocupação (expressão 4.22):

$$\rho(\vec{x}, t) = \sum_{i=1}^{z} N_i(\vec{x}, t). \tag{4.23}$$

De forma similar, o momento linear ($\rho\vec{u}$) é dado por (Chopard e Droz, 1998):

$$\rho(\vec{x}, t)\vec{u}(\vec{x}, t) = \sum_{i=1}^{z} \vec{v}_i N_i(\vec{x}, t). \tag{4.24}$$

A partir destas definições e das leis de conservação é possível estimar o comportamento macroscópico do sistema. Para isso, usa-se a expansão multiescala de Chapman-Enskog. Os detalhes desta técnica podem ser encontrados nos trabalhos de Frisch et al (1986) e Xavier (2006). O resultado é a obtenção das Equações de Conservação de Massa e Navier-Stokes a partir da dinâmica microscópica do FHP descrita nesta seção.

4.3 Método de *Lattice Boltzmann*

Apesar de suas vantagens, os LGCAs apresentam certas limitações relacionadas a sua natureza discreta: o surgimento de *ruído*, o que torna necessário o uso de processos envolvendo cálculo de valores médios, e pouca flexibilidade para ajustar os parâmetros físicos e condições iniciais (Xavier, 2006; Chopard e Droz, 1998).

No final dos anos 80, o trabalho de McNamara e Zanetti (1988) mostrou a vantagem de se extender a dinâmica *booleana* dos autômatos para trabalhar diretamente com números reais representando a distribuição N_i, dada pela expressão 4.22. Esta metodologia, chamada método de *Lattice Boltzmann* (LBM) é mais eficiente que a dinâmica *booleana* e provê um modelo computacional mais apropriado para simular fluidos com número de Reynolds alto (Chopard et al, 1998). A equação básica do método é dada por:

$$N_i(\vec{x} + \Delta_x \vec{c}_i,\ t + \Delta_t) \ = \ N_i(\vec{x}, t) + \Omega_i(N), \tag{4.25}$$

onde $\Omega_i(N)$ é o termo de colisão. O cálculo deste termo, é o ponto crítico com relação à equação (4.25), pois este termo aparece como função dos N_i em lugar das variáveis microscópicas n_i da expressão (4.20).

Segundo Chopard et al (1998), a maneira mais natural para se definir o termo de colisão neste caso é tirando-se a média da microdinâmica e fatorando-a em um produto de quantidades médias. Entretanto, para fluidos mais sofisticados, o termo de colisão requer um grande número de operações em ponto flutuante a cada nó da *lattice* em cada instante de tempo, o que aumentaria o custo computacional. Uma solução é utilizar o modelo BGK, descrito a seguir.

4.3.1 Modelo BGK

Os fundamentos teóricos deste modelo se encontram na teoria cinética e estão descritos no Apêndice A. Seguindo esta teoria, o trabalho de Higuera et al (1989a) propôs linearizar o termo de colisão $\Omega_i(N)$, na expressão (4.25), em torno de sua solução de equilíbrio local, o que reduziu o custo computacional e simplificou as operações envolvidas. Desta forma, foi proposto em (Higuera et al, 1989b) a seguinte simplificação para o termo de colisão:

$$\Omega_i(f) = -\frac{1}{\tau}\left(f_i(\vec{x}, t) - f_i^{eq}(\rho, \vec{u}) \right), \tag{4.26}$$

onde seguimos a notação tradicional do LBM e substituímos N_i por f_i, e f_i^{eq} indica a solução de equilíbrio. Portanto, substituindo a expressão (4.26) na equação (4.25), a dinâmica do *Lattice Boltzmann*, em sua forma simples, pode ser escrita através da distribuição dada a seguir:

$$f_i(\vec{x} + \Delta_x \vec{c}_i,\ t + \Delta_t) - f_i(\vec{x}, t) = -\frac{1}{\tau}\Big(f_i(\vec{x}, t) - f_i^{eq}(\rho, \vec{u})\Big), \tag{4.27}$$

onde a quantidade τ é uma relaxação temporal que determinará a viscosidade do fluido (Chopard et al, 1998; Chen e Doolen, 1998).

A função de equilíbrio $f_i^{eq}(\rho, \vec{u})$ é descrita em termos da densidade e velocidade macroscópicas (Chopard et al, 1998; Chen e Doolen, 1998), onde sua forma geral é dada pela expressão:

$$f_i^{eq} \ = \ \rho\omega_i\Big[a \ + \ b(\vec{c}_i \cdot \vec{u}) \ + \ c(\vec{c}_i \cdot \vec{u})^2 \ + \ d\vec{u}^2\Big], \tag{4.28}$$

onde ρ e $\vec{u}$ são as quantidades macroscópicas de densidade e velocidade respectivamente, $\vec{c}_i$ é a direção de movimento, os pesos ω_i e as constantes a_i, b_i, c_i e d_i serão descritas na seção 4.3.2 a seguir.

4.3.2 Modelo D2Q9

O modelo de *Lattice Boltzmann* implementado neste trabalho é um modelo bidimensional, conhecido como D2Q9, com oito direções não nulas de movimento e a possibilidade de ter uma partícula parada. O espaço do modelo D2Q9 é discretizado em uma *lattice* bidimensional, como ilustra a Figura 4.3.

No modelo D2Q9 o passo de tempo é discreto e para cada nó da *lattice* existem nove possíveis direções de movimento, onde $\vec{c}_0$ representa o próprio nó da *lattice*, cuja velocidade é zero (Chopard et al, 1998; Chen e Doolen, 1998). As oito possíveis velocidades $\vec{v}_i$, dadas pela expressão (4.14), estão relacionadas com

as direções $\vec{c}_i$ de movimento, descritas por:

$$\begin{aligned} &\vec{c}_0 = (0,0), \qquad \vec{c}_1 = (1,0)\vec{v}, \qquad \vec{c}_2 = (1,1)\vec{v}, \\ &\vec{c}_3 = (0,1)\vec{v}, \qquad \vec{c}_4 = (-1,1)\vec{v}, \quad \vec{c}_5 = (-1,0)\vec{v}, \\ &\vec{c}_6 = (-1,-1)\vec{v}, \quad \vec{c}_7 = (0,-1)\vec{v}, \quad \vec{c}_8 = (1,-1)\vec{v}. \end{aligned} \tag{4.29}$$

São introduzidas duas velocidades, de acordo com a distância a ser percorrida em um passo de tempo, a saber: as velocidades *lentas* com módulo dado por $\vec{v} = \Delta_x/\Delta_t$ nas direções vertical e horizontal ($\vec{c}_1,\ \vec{c}_3,\ \vec{c}_5,\ \vec{c}_7$) e as velocidades *rápidas* com módulo $\sqrt{2}\vec{v}$ nas direções diagonais ($\vec{c}_2,\ \vec{c}_4,\ \vec{c}_6,\ \vec{c}_8$). Isto ocorre devido às diferentes distâncias que, em um mesmo intervalo de tempo, as distribuições devem viajar até os nós vizinhos.

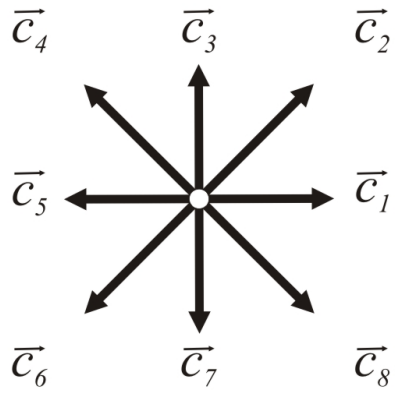


Figura 4.3: Direções de movimento na *lattice* do D2Q9.

A regra de atualização do modelo D2Q9 segue o modelo BGK descrito na equação (4.27). As quantidades macroscópicas ρ e $\vec{u}$ são dadas por:

$$\rho(\vec{x},t) \ = \ \sum_{i=0}^{z} f_i(\vec{x},t) \tag{4.30}$$

$$\vec{u}(\vec{x},t) \ = \ \frac{1}{\rho(\vec{x},t)} \sum_{i=1}^{z} f_i(\vec{x},t)\vec{c}_i, \tag{4.31}$$

onde z representa o número de direções não nulas da *lattice* ($z = 8$).

A forma geral da função de distribuição de equilíbrio é dada pela equação (4.28). He e Luo (1997) obtém os seguintes valores de pesos ω_i e das constantes para o modelo D2Q9:

$$f_i^{eq} \;=\; \rho\omega_i\Bigg[1 \;+\; 3(\vec{c}_i\cdot\vec{u}) \;+\; \frac{9}{2}(\vec{c}_i\cdot\vec{u})^2 \;-\; \frac{3}{2}\vec{u}^2\Bigg], \tag{4.32}$$

onde:

$$\omega_0 = \frac{4}{9}, \quad \omega_i = \frac{1}{9},\ (i=1,3,5,7), \quad \omega_i = \frac{1}{36},\ (i=2,4,6,8).$$

No método D2Q9, a evolução do fluido está baseada nas seguintes etapas (para cada nó $\vec{x}$):

(1) **Inicialização ($t = 0$):**

(a) Define-se valor inicial de densidade macroscópica $\rho(\vec{x}, 0)$

(b) Define-se valor inicial de velocidade macroscópica $\vec{v}(x, 0)$

(c) $f_i(\vec{x}, 0) = f_i^{eq}(\rho, \vec{u})$

(2) **Loop principal ($t = 1$ até t_{max}):**

(a) Cálculo da Colisão $f_i^C(\vec{x}, t)$

(b) Cálculo da Propagação $f_i^P(\vec{x} + \Delta_x\vec{c}_i,\ t + \Delta_t)$

(c) Atualiza densidade (4.30) e velocidade (4.31) macroscópica

(d) $f_i(\vec{x}, t) = f_i^P(\vec{x} + \Delta_x\vec{c}_i,\ t + \Delta_t)$

As etapas de colisão e propagação das distribuições nos nós representam uma divisão da equação de *Lattice Boltzmann* (4.27), a saber:

$$f_i^C(\vec{x}, t) = f_i(\vec{x}, t) - \frac{1}{\tau}\Bigg(f_i(\vec{x}, t) - f_i^{eq}(\rho, \vec{u})\Bigg),\ \text{(colisão)} \tag{4.33}$$

e

$$f_i^P(\vec{x} + \Delta_x\vec{c}_i,\ t + \Delta_t) = f_i^C(\vec{x}, t),\ \text{(propagação)}. \tag{4.34}$$

4.3.3 Condições de Contorno

Diferentes tipos de condição de contorno têm sido introduzidos no campo da Hidrodinâmica para o LBM. *Bounce-back* é o mais simples deles (Rothman e Zaleski, 1997), utilizado em condições de contorno cujas paredes são não escorregadias, ou seja, os nós presentes nas paredes apresentam velocidade nula em todas as direções. Quando as partículas propagam para um nó da fronteira, elas simplesmente retornam pela mesma direção em que estavam.

Pela sua simplicidade, tal método não consegue representar adequadamente problemas de contornos curvos sem a introdução de ruído na solução, principalmente em *lattices* pouco refinadas. Um valor incorreto pode eventualmente causar um valor de densidade negativo, gerando um erro que possivelmente se acumulará ao longo da evolução (Rothman e Zaleski, 1997).

Neste trabalho implementamos a condição de contorno proposta por Zou e He (1997), onde os autores impõem valores de velocidade (não necessariamente nula) nos centros dos nós da fronteira. A idéia básica consiste em introduzir um passo, entre a etapa de propagação (4.34) e colisão (4.33), para calcular as distribuições em algumas direções dos nós da fronteira.

Após a propagação, as f_i's oriundas do interior e da própria fronteira são conhecidas nos nós de fronteira. As f_i's desconhecidas bem como a densidade (ρ) do nó de fronteira são calculadas a partir da equação de conservação de massa (expressão (4.30)), da equação de conservação do momento linear (expressão (4.31)), onde é imposta a velocidade da fronteira, e através da reflexão da parte denominada de não-equilíbrio (Zou e He, 1997), ou seja:

$$f_i - f_i^{eq} = f_{i+z/2} - f_{i+z/2}^{eq}, \tag{4.35}$$

onde z representa o número de direções não nulas da *lattice*. Para o modelo D2Q9 temos $z = 8$.

Para exemplificarmos a condição de contorno implementada, vamos mostrar duas situações de fronteira no modelo D2Q9: nós de fronteira presentes nas paredes (Figura 4.4(a)) e nós de fronteira presentes nos cantos da *lattice* (Figura 4.4(b)).

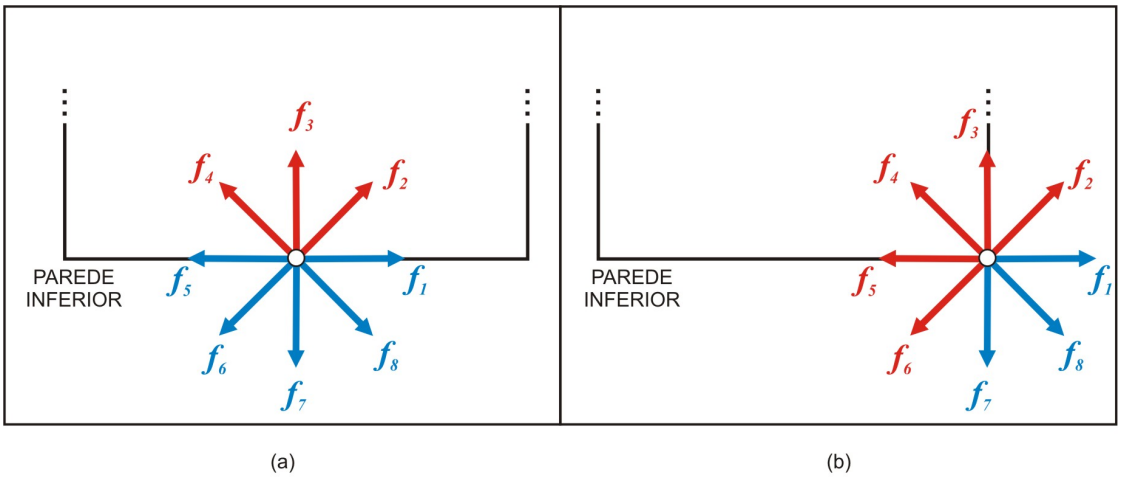


Figura 4.4: Fronteiras do modelo D2Q9. (a) Um exemplo de um nó presente na parede inferior. (b) Um exemplo de um nó presente no canto da *lattice*.

A primeira situação mostra o caso de um nó de fronteira presente na parede inferior. Após a propagação (4.34), as distribuições f_1, f_5, f_6, f_7 e f_8 são conhecidas, pois são oriundas dos nós vizinhos. De acordo com a condição de contorno proposta por Zou e He (1997), vamos impor um valor de velocidade ($\vec{u}$) nos nós de fronteira presentes na parede inferior. Para determinarmos as quatro variáveis desconhecidas (f_2, f_3, f_4 e ρ) vamos utilizar a equação de conservação de massa (4.30):

$$f_2 + f_3 + f_4 \;=\; \rho - (f_0 + f_1 + f_5 + f_6 + f_7 + f_8), \tag{4.36}$$

as equações de conservação do momento linear (4.31) na direção x:

$$f_2 - f_4 \;=\; \rho u_x - f_1 + f_5 + f_6 - f_8, \tag{4.37}$$

e na direção y:

$$f_2 + f_3 + f_4 \;=\; \rho u_y + f_6 + f_7 + f_8. \tag{4.38}$$

A partir das equações (4.36) e (4.38), temos:

$$\rho \;=\; \frac{1}{1-u_y}\Big[f_0+f_1+f_5+2(f_6+f_7+f_8)\Big]. \tag{4.39}$$

Entretanto, f_2, f_3 e f_4 continuam indeterminados. Desta forma, utilizamos o reflexo da parte de não-equilíbrio da distribuição f_7:

$$f_3 - f_3^{eq} = f_7 - f_7^{eq}. \tag{4.40}$$

A partir das equações (4.36)-(4.40) conseguimos calcular as variáveis desconhecidas:

$$f_3 \;= f_7 + \frac{2}{3}\rho u_y \tag{4.41}$$

$$f_4 \;= f_8 + \frac{1}{2}\big(f_1 - f_5\big) - \frac{1}{2}\rho u_x + \frac{1}{6}\rho u_y \tag{4.42}$$

$$f_2 \;= f_6 - \frac{1}{2}\big(f_1 - f_5\big) + \frac{1}{2}\rho u_x + \frac{1}{6}\rho u_y. \tag{4.43}$$

Um procedimento análogo é aplicado aos nós que estão presentes nos cantos da *lattice*. Como exemplo, vamos analisar o nó presente no canto inferior-direito, como mostra a Figura 4.4(b). Após a propagação (4.34), as distribuições f_0, f_1, f_7 e f_8 são conhecidas. Vamos impor então um valor de velocidade ($\vec{u}$), e para determinarmos as seis variáveis desconhecidas (f_2, f_3, f_4, f_5, f_6 e ρ) vamos utilizar a equação de conservação de massa (4.30):

$$f_2+f_3+f_4+f_5+f_6 \;=\; \rho-(f_0+f_1+f_7+f_8), \tag{4.44}$$

as equações de conservação do momento linear (4.31) na direção x:

$$f_2-f_4-f_5-f_6 \;=\; \rho u_x-(f_1+f_8), \tag{4.45}$$

e na direção y:

$$f_2+f_3+f_4-f_6 \;=\; \rho u_y+f_7+f_8. \tag{4.46}$$

Com 3 equações e 6 variáveis desconhecidas não é possível determinar ρ. Desta forma, a densidade no nó de fronteira no canto inferior-direito será determinada através da média das densidades vizinhas. Entretanto, f_2, f_3, f_4, f_5 e f_6 continuam indeterminados. Utilizamos então o reflexo da parte de não-equilíbrio da distribuição f_1 e da distribuição f_7:

$$f_5 - f_5^{eq} = f_1 - f_1^{eq}. \tag{4.47}$$

$$f_3 - f_3^{eq} = f_7 - f_7^{eq} \tag{4.48}$$

A partir das equações (4.44)-(4.48) e do valor estimado de densidade (ρ) conseguimos calcular as variáveis desconhecidas:

$$f_3 = f_7 + \frac{2}{3}\rho u_y \tag{4.49}$$

$$f_5 = f_1 - \frac{2}{3}\rho u_x \tag{4.50}$$

$$f_4 = f_8 - \frac{1}{6}\rho u_x + \frac{1}{6}\rho u_y \tag{4.51}$$

$$f_6 = \frac{1}{2}\big(\rho - f_0\big) - (f_1 + f_7 + f_8) + \frac{1}{3}\rho u_x - \frac{1}{2}\rho u_y \tag{4.52}$$

$$f_2 = \frac{1}{2}\big(\rho - f_0\big) - (f_1 + f_7 + f_8) + \frac{1}{2}\rho u_x - \frac{1}{3}\rho u_y. \tag{4.53}$$

Capítulo 5

Técnicas de *Rendering*

Vimos no Capítulo 4 técnicas de simulação capazes de reproduzir o comportamento de fluidos. Veremos agora as técnicas utilizadas para a visualização da cena final.

Rendering é o processo utilizado para produção de imagens com o nível de realismo desejado, o que envolve tanto modelos físicos quanto conceitos de percepção visual (Rogers, 1998).

Neste capítulo discutiremos técnicas de *rendering* utilizadas neste trabalho. Iniciamos apresentando uma taxonomia para modelos de iluminação, seguindo com uma breve discussão sobre os conceitos de realismo e tempo real. A seção 5.4 apresenta conceitos básicos em modelos de iluminação, com o objetivo de fornecer o modelo de iluminação utilizado nas aplicações em tempo real, descrito na seção 5.5. Finalmente, na seção 5.6.1, descrevemos o método de *Volume Scattering* utilizado na aplicação *off-line*.

5.1 Taxonomia para Modelos de Iluminação

Segundo Adabala e Manohar (2002), os modelos de iluminação podem ser agrupados como mostra a Figura 5.1.

As técnicas **visualmente convincentes** utilizam métodos heurísticos para a geração de imagens aparentemente realistas, não sendo necessário se basear nas leis da física. Por exemplo, Reeves (1983) propôs um modelo de iluminação que utiliza

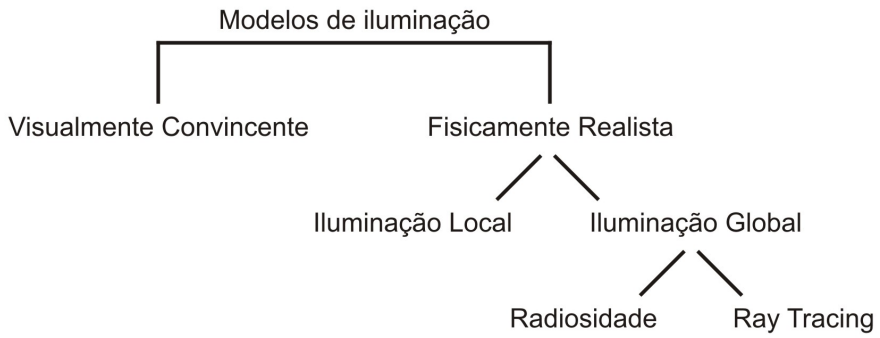


Figura 5.1: Classificação dos modelos de iluminação. Fonte: Adabala e Manohar (2002) p.72

partículas como fontes de luz e ignora demais interações luminosas. O trabalho de Sims (1990) aplica técnica de *motion blur* nas partículas, gerando imagens de quedas d'água, fogo, neve, etc.

Em contrapartida, as técnicas **fisicamente realistas** buscam simular a interação da luz com os objetos em cena através das leis da física, como veremos na seção 5.4. Os **modelos de iluminação local** para fluidos consideram principalmente a interação da luz com o volume, simulando a atenuação da luz ao passar pelo meio participante (Adabala e Manohar, 2002). O trabalho de Blinn (1982) propôs um modelo simples de interação luminosa com nuvens ou superfícies que apresentem poeira, sujeira, levando em consideração meios participantes homogêneos. Avanços da técnica tornaram possível trabalhar em meios heterogêneos bem como simular efeitos que implicam em espalhamento múltiplo da luz (Kajiya e Von Herzen, 1984; Max, 1994).

Os **modelos de iluminação global** citados na Figura 5.1 (Radiosidade e *Ray Tracing*) levam em conta todo o tipo de interação luminosa que ocorre em uma cena do mundo real. Proposto por Goral et al (1984), o algoritmo de **Radiosidade** tem origem nos métodos fundamentais de transferência de calor. A radiosidade é definida como sendo a energia por unidade de tempo e área em cada ponto dos objetos em cena, onde a energia luminosa incidente em uma superfície é refletida com igual intensidade em todas as direções.

Rushmeier e Torrance (1987) desenvolveram um dos primeiros trabalhos nesta área em meio participante. Todas as interações luminosas em uma cena são levadas em conta, a saber, superfície-superfície, volume-volume e superfície-volume. É uma técnica aplicada a meios homogêneos. Extensões foram feitas para o caso de meios heterogêneos (Bhate, 1993).

O ***Ray tracing*** foi desenvolvido em 1968 como um algoritmo para simulação de trajetórias de projéteis balísticos e partículas nucleares (Azevedo e Conci, 2003). A primeira implementação para computação gráfica foi desenvolvida por Appel (1968), como uma ferramenta para o cálculo de sombras. Whitted (1980) propôs um algoritmo *ray tracing* para o cálculo de sombras, reflexões e refrações. Por sua simplicidade, muitos softwares de *rendering* comerciais atualmente são baseados na pesquisa de Whitted.

Extensões do modelo de *Ray tracing* para meio participante e que levam em conta todos os tipos de interações luminosas em um ambiente, incluem os trabalhos de Lafortune et al (1996), Veach e Guibas (1997) e Jensen e Christensen (1998). Esses trabalhos são capazes de capturar interações superfície-volume e cáusticas.

5.2 Conceito de Realismo

A busca pela síntese de imagens realistas tem sido umas das principais direções de pesquisa em computação gráfica. Entretanto, não há metodologia para se medir o realismo. Algumas quantidades físicas podem ser usadas como critério, ou mesmo a percepção visual da cena (Adabala e Manohar, 2002). Segundo Ferwerda (2003), no contexto de imagens geradas por computador é possível distinguir dois tipos de realismo: Realismo Físico e Foto Realismo.

No **Realismo Físico** o modelo tem que representar precisamente a cena envolvida (formas, materiais e propriedades luminosas). A técnica de *rendering* deve calcular a interação luminosa baseado nas leis da física. Além disso, o dispositivo de saída (como os monitores, por exemplo) deve ser capaz de apresentar a cena com a qualidade necessária (Ferwerda, 2003).

No **Foto Realismo** a imagem deve produzir a mesma resposta visual da cena representada, mesmo que não simule a física envolvida no processo. Desta forma, o processo de geração de imagem ganha desempenho comparado ao Realismo Físico que demanda alto custo computacional (Ferwerda, 2003).

5.3 *Rendering* em Tempo Real e *Off-line*

As técnicas de *rendering* podem ser implementadas para aplicações em tempo real ou não. Aplicações cuja execução não ocorre em tempo real são chamadas de *off-line* (Müller et al, 2008). As técnicas de *rendering* para aplicações *off-line* têm como principal objetivo a busca pela qualidade visual nas imagens geradas. Conseqüentemente, o desempenho computacional não é tão importante comparado à necessidade de qualidade na imagem gerada. São técnicas voltadas para mídias não interativas, assim como filmes e vídeos, como os exemplos mostrados na Figura 5.2.

Figura 5.2: Exemplos de aplicações de técnicas de *rendering off-line*: (a) *Shrek*, animação da *Dreamworks* de 2001. (b) *Beowulf*, animação da *Warner Bros.* de 2007.

Os sistemas interativos, como jogos eletrônicos e simuladores, utilizam técnicas de *rendering* em tempo real, como os exemplos mostrados na Figura 5.3. São aplicações que mantém uma taxa igual ou superior a 30 frames por segundo para garantir uma percepção suave e contínua da animação (Müller et al, 2008).

(a) (b)

Figura 5.3: Exemplos de aplicações de técnicas de *rendering* em tempo real: (a) *Mirror's Edge*, jogo eletrônico da *EA Games* de 2008. (b) *The Godfather 2*, jogo eletrônico da *EA Games* de 2008.

5.4 Modelos de Iluminação

Esta seção fornece elementos básicos em modelos de iluminação com o objetivo de facilitar a descrição dos métodos de *rendering* utilizados para tempo real. Quando a luz interage com uma superfície, ela pode ser absorvida, refletida ou transmitida. Uma parcela da energia luminosa incidente é absorvida e convertida em calor. A parcela restante que é refletida ou transmitida é a responsável pela aparência do objeto. Desta forma, se quase toda a energia for absorvida, o objeto aparecerá negro (Rogers, 1998).

A parcela da luz refletida ou transmitida através da superfície de um objeto depende da composição, direção e geometria da fonte de luz; da orientação da superfície; bem como das propriedades do objeto. A reflexão pode ocorrer de maneira difusa ou especular (Rogers, 1998).

5.4.1 Reflexão Difusa

A luz refletida difusamente é espalhada igualmente em todas as direções, tornando irrelevante a posição do observador. É uma reflexão típica de superfícies foscas (Rogers, 1998). A intensidade refletida difusamente pode ser calculada usando-se a lei do cosseno de Lambert, que diz que tal intensidade é proporcional

ao cosseno do ângulo entre a direção da luz e a normal à superfície (Rogers, 1998):

$$I(\lambda) = I_l(\lambda)k_d(\lambda)\cos\theta, \quad 0 \leq \theta \leq \frac{\pi}{2} \tag{5.1}$$

onde $I(\lambda)$ é a intensidade refletida, $I_l(\lambda)$ é a intensidade da fonte de luz pontual, $k_d(\lambda)$ é o coeficiente de reflexão difusa ($0 \leq k_d(\lambda) \leq 1$), λ é o comprimento de onda e θ é o ângulo entre a direção da luz e a normal à superfície, como mostra a Figura 5.4. Um modelo simples de iluminação geralmente assume k_d e I_l constantes (Rogers, 1998).

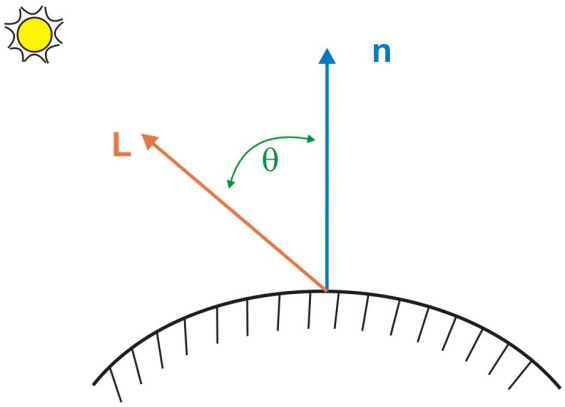


Figura 5.4: Reflexão Difusa, onde: L é a direção da luz incidente e n a normal à superfície. Fonte: Rogers (1998), p.461

Usa-se também uma componente de iluminação ambiente. Esta componente é uma aproximação da quantidade de iluminação indireta presente em uma cena (Rogers, 1998). O modelo de iluminação fica então:

$$I(\lambda) = I_a(\lambda)k_a(\lambda) + I_l(\lambda)k_d(\lambda)\cos\theta, \quad 0 \leq \theta \leq \frac{\pi}{2}, \quad k_a + k_d < 1 \tag{5.2}$$

onde $I_a(\lambda)$ é a intensidade ambiente e $k_a(\lambda)$ é o coeficiente de reflexão difusa ambiente ($0 \leq k_a(\lambda) \leq 1$). Em modelos simples de iluminação k_a e I_a são constantes (Rogers, 1998).

5.4.2 Reflexão Especular

A reflexão especular é a componente responsável pelo brilho do objeto. A intensidade refletida especularmente depende do ângulo de incidência, do comprimento de onda da luz incidente e das propriedades do objeto. A reflexão especular é direcional, ou seja, depende da posição do observador, como mostra a Figura 5.5 (Rogers, 1998; Azevedo e Conci, 2003).

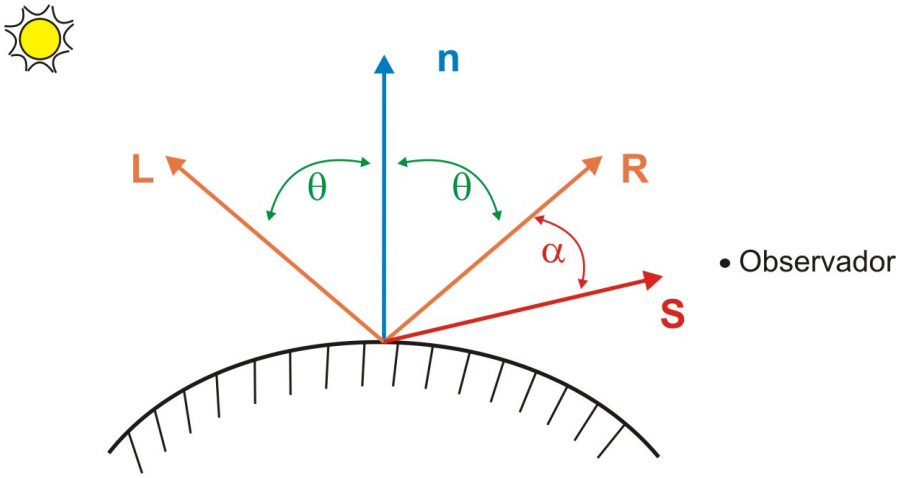


Figura 5.5: Reflexão Especular, onde: L é a direção da luz incidente, R a direção da luz refletida, S a direção do observador e n a normal à superfície. Fonte: Rogers (1998), p.463

O vetor de reflexão R pode ser computado em função dos vetores n e L (Fernando e Kilgard, 2003):

$$R = L - 2n(n \cdot L). \tag{5.3}$$

Um modelo simples para o cálculo da intensidade refletida especularmente é dado por (Rogers, 1998):

$$I_s(\lambda) = I_l(\lambda) w(i, \lambda) \cos^n \alpha, \tag{5.4}$$

onde $I_s(\lambda)$ é a quantidade refletida especularmente, $I_l(\lambda)$ é a intensidade da fonte de luz pontual, $w(i, \lambda)$ é a curva de reflectância em função do ângulo de incidência

i e do comprimento de onda λ, e $n \in \mathbb{N}$. Quanto menor o valor de n maior é o espalhamento da luz em torno da direção S (Figura 5.5), característico de materiais não metálicos, como papel por exemplo (Rogers, 1998).

Considerando as componentes de luz ambiente, difusa, e especular, descritas pelas equações (5.2) e (5.4), foi proposto por Phong (1975) um modelo de iluminação dado por:

$$I(\lambda) = I_a(\lambda)k_a(\lambda) + I_l(\lambda)\big[k_d(\lambda)\cos\theta + w(i,\lambda)\cos^n\alpha\big]. \tag{5.5}$$

Pela complexidade da função $w(i,\lambda)$, modelos simples costumam substituí-la por uma constante k_s (Rogers, 1998):

$$I = I_a k_a + I_l\big(k_d\cos\theta + k_s\cos^n\alpha\big). \tag{5.6}$$

5.5 Rendering em Tempo Real

Para execuções em tempo real utilizamos neste trabalho os seguintes métodos de *rendering*: *Cube Mapping* para criar cenários, como um dia ensolarado por exemplo; *Texture Mapping* para renderizar superfícies; *Point Sprite* para renderizar partículas; e Efeito *Fresnel* para simular a superfície livre do fluido.

Nas próximas seções descreveremos brevemente cada uma dessas técnicas utilizadas.

5.5.1 *Cube Mapping*

A técnica de *Cube Mapping* foi proposta por Greene (1986). Tal técnica é usada para criar ambientes com o auxílio de texturas bidimensionais. O mapa de textura consiste de seis componentes bidimensionais, correspondentes às faces de um cubo (Lengyel, 2004; Wright et al, 2007). Essas componentes representam a visão do mundo a partir de seis direções distintas (x, y e z negativos e positivos), como mostra a Figura 5.6(a). Neste trabalho utilizamos esta técnica para criar diversos cenários, por exemplo um dia ensolarado, como mostra a Figura 5.6(b).

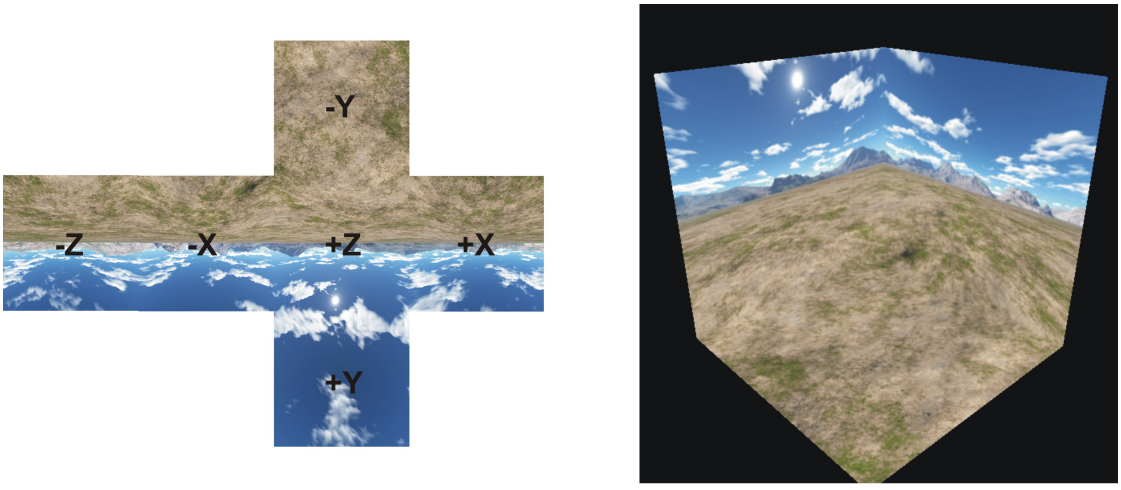


Figura 5.6: (a) Orientação das faces do *Cube Mapping*. (b) Exemplo de *Cube Mapping* na simulação de um dia ensolarado.

O uso de *Cube Mapping* em conjunto com técnicas de *Environment Mapping*(Blinn e Newell, 1976) possibilita simular a reflexão do ambiente na superfície dos objetos em cena. A Figura 5.7 ilustra esse efeito, onde temos uma cena envolta pelos limites do *cube mapping*.

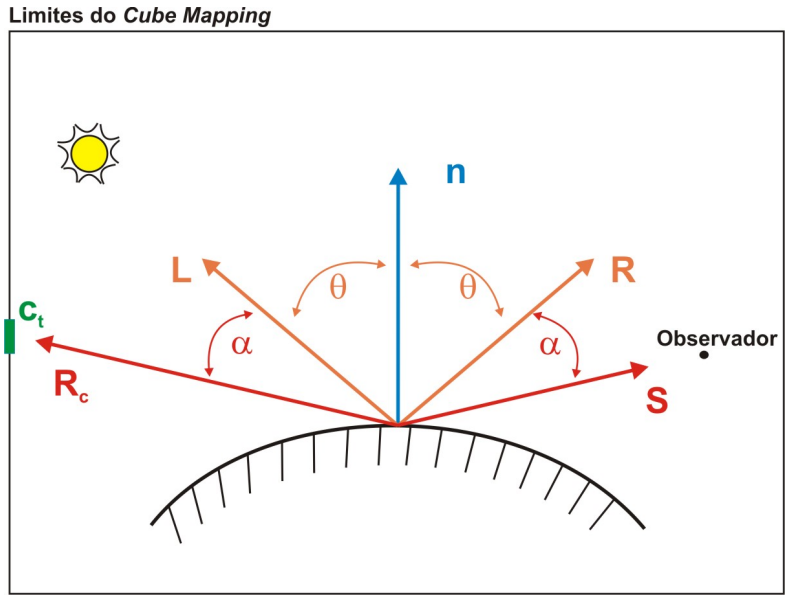


Figura 5.7: Reflexão do ambiente na superfície do objeto, onde: n é o vetor normal à superfície, L é a direção da luz incidente, R a direção da luz refletida especularmente, S a direção do observador, R_c a reflexão do vetor S e c_t é o pixel do *cube mapping*.

No ponto em que queremos calcular a cor, na superfície do objeto, temos a orientação do vetor normal n e o vetor L que representa a direção incidente da fonte de luz no ponto. O vetor S representa a direção de observação vinda do observador até a superfície do objeto, e o vetor R (dado pela equação (5.3)) representa a reflexão especular no ponto. O vetor R_c representa a reflexão do vetor S em relação a normal, e é dado pela equação (5.3) (substituindo o vetor L por S). A interseção desse vetor com o *cube mapping* indica o pixel c_t na textura que é refletido na superfície do objeto e, consequentemente, visualizado pelo observador.

Neste trabalho, utilizamos a influência do *cube mapping* apenas para o *rendering* da superfície livre do fluido. Este cálculo será descrito na seção 5.5.2 abaixo.

5.5.2 Efeito *Fresnel*

O efeito *Fresnel* foi documentado pela primeira vez pelo físico francês Augustin-Jean Fresnel (1788-1827) (Fernando e Kilgard, 2003). A Lei de Fresnel descreve o comportamento da luz quando transita entre meios com diferentes coeficientes de refração, como ilustra a Figura 5.8.

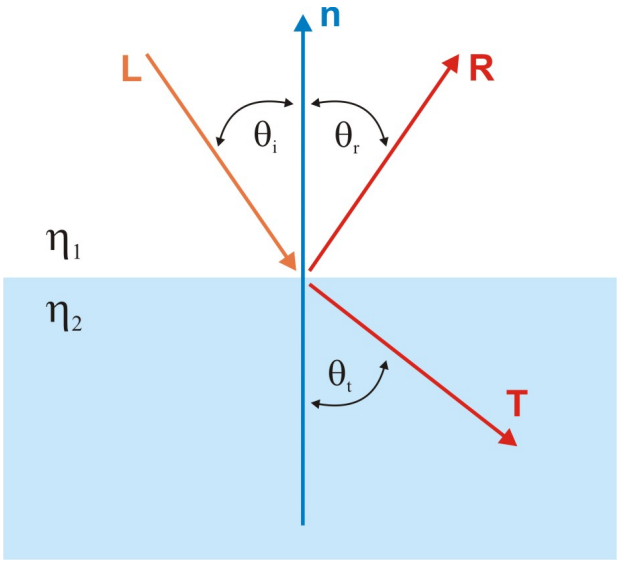


Figura 5.8: Componentes utilizadas na Lei de Fresnel: coeficientes de refração dos meios (η_1 e η_2), direção da luz incidente (L), direção da luz refletida (R), direção da luz refratada (T), normal à superfície (n) e ângulos formados entre os vetores (θ_i, θ_r e θ_t).

A quantidade da luz que é refletida (F_r) em relação à que é refratada (F_t) pode ser determinada a partir dos coeficientes de refração dos materiais (η_1 e η_2), do ângulo de incidência em relação à normal (θ_i) e do ângulo do raio refratado (θ_t):

$$F_r = \left[\frac{\eta_1 \cos(\theta_i) - \eta_2 \cos(\theta_t)}{\eta_1 \cos(\theta_i) + \eta_2 \cos(\theta_t)}\right]^2, \tag{5.7}$$

onde o ângulo θ_t é determinado a partir do ângulo de incidência θ_i, usando a lei de Snell:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_2}{\eta_1}. \tag{5.8}$$

Uma aproximação para a Lei de Fresnel, usada especialmente em aplicações em tempo real, é a aproximação proposta por Schlick (1994):

$$F_r = R_0 + (1 - R_0)(1 + \cos(\theta_i))^5, \tag{5.9}$$

onde θ_i é o ângulo de incidência e R_0 é a reflectância na direção da normal, que pode ser calculada com base na Lei de Fresnel:

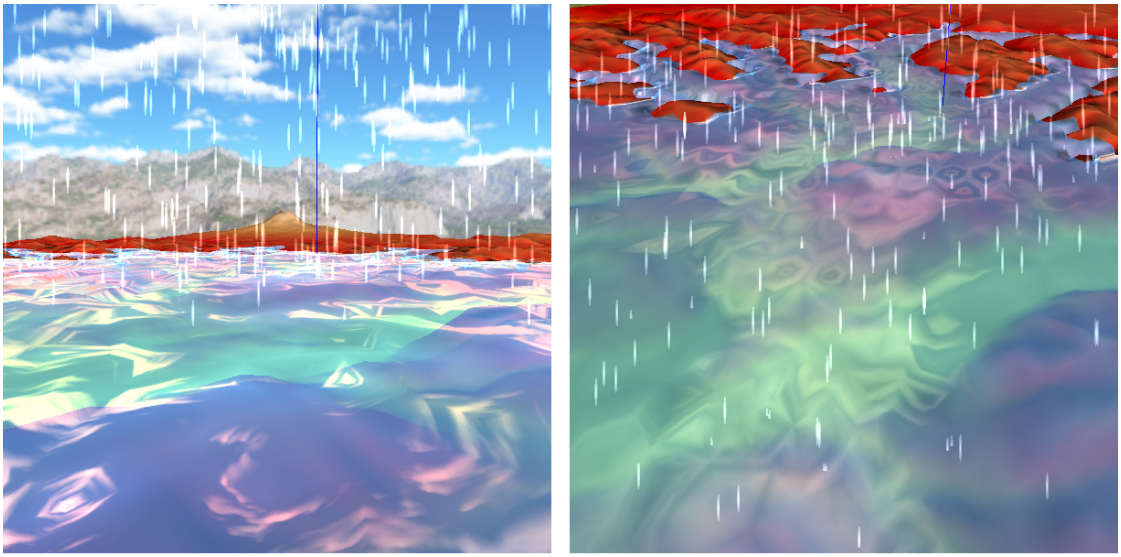
$$R_0 = \frac{(\eta_1 - \eta_2)^2}{(\eta_1 + \eta_2)^2}. \tag{5.10}$$

A observação da reflectância em uma dada superfície depende do ângulo de visão. A Figura 5.9(a) mostra o fluido visto em um ângulo próximo ao nível da superfície livre, onde podemos notar um grau maior de reflexão em comparação à Figura 5.9(b).

Para estimar a cor em cada pixel (c_p), é feita uma interpolação linear, dada por:

$$c_p = I \cdot (1 - F_r) + (c_t \cdot F_r), \tag{5.11}$$

onde I é a intensidade no pixel dada pela expressão (5.6), F_r é o fator de Fresnel dado pela aproximação (5.9) e c_t é a cor da textura do *cube mapping* (vide seção 5.5.1) obtida através do vetor de reflexão dado por (5.3).



(a) (b)

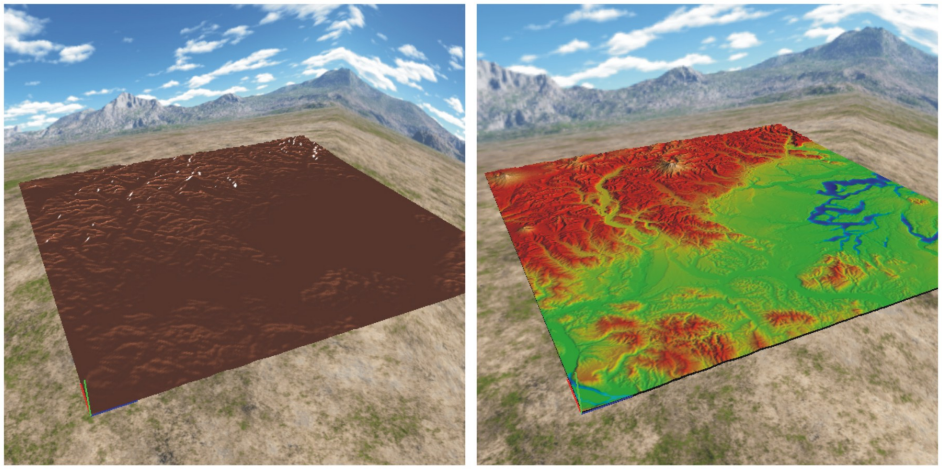
Figura 5.9: Exemplo de Efeito *Fresnel* na superfície livre do fluido observado por dois ângulos distintos.

5.5.3 *Texture Mapping*

Descrever propriedades da superfície de um objeto, tais como rugosidade e padronagem, é um processo cuja complexidade está relacionada ao grau de detalhes envolvido nessas propriedades. Uma alternativa seria acrescentar esses detalhes na geometria da superfície o que poderia tornar o processamento muito custoso. Outra alternativa é modelar estes efeitos com o uso de mapas de textura (Azevedo e Conci, 2003).

A técnica de *Texture Mapping* é tradicional nesta área (Catmull, 1974). Sua idéia consiste em aplicar uma imagem, ou seja, uma textura sobre um objeto na cena. Isso se dá através de um mapeamento que associa um ponto da textura a um ponto sobre a superfície do objeto (Azevedo e Conci, 2003; Fernando e Kilgard, 2003).

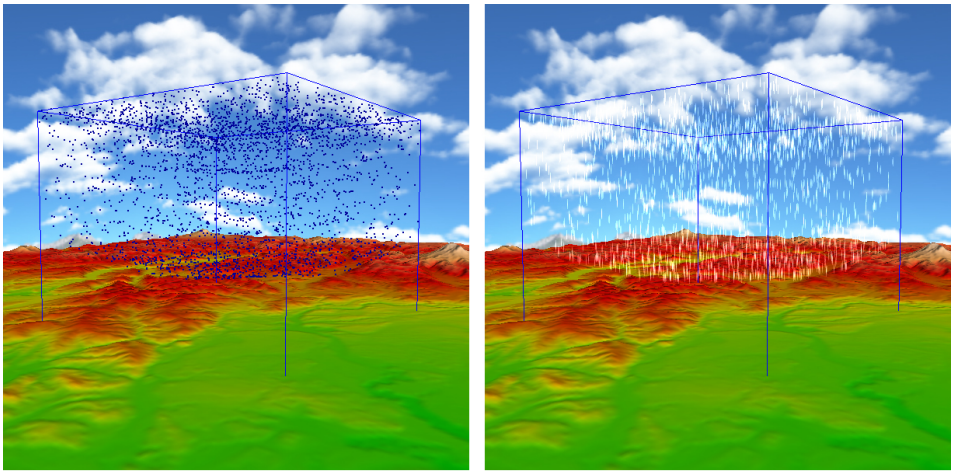
No nosso trabalho utilizamos *Texture Mapping* para renderizar modelos digitais de terreno. A Figura 5.10(a) mostra um exemplo de modelo de terreno sem textura, enquanto a Figura 5.10(b) mostra o mesmo terreno com o uso do *Texture Mapping*.



(a) (b)

Figura 5.10: (a) Exemplo de modelo de terreno sem textura. (b) Modelo de terreno com aplicação de *Texture Mapping*.

Uma alternativa rápida de mapeamento de textura, muito utilizada em sistema de partículas, é a técnica de *Point Sprite*. Tal técnica permite posicionar uma imagem de textura bidimensional na cena a partir da posição de um ponto no espaço. Desta forma, em um sistema de partículas, basta armazenar a localização de cada partícula e a textura a ser mapeada. A Figura 5.11(a) mostra as partículas utilizadas em nosso trabalho para representar uma chuva. Na Figura 5.11(b) temos o mesmo sistema de partículas com aplicação de *point sprite*.



(a) (b)

Figura 5.11: (a) Exemplo de partículas. (b) Mesmo sistema de partículas com aplicação de *Point Sprite*.

5.6 Rendering Off-Line

Para a aplicação *Off-line* utilizamos a técnica de *rendering* volumétrico denominada o *Volume Scattering*, que tem como base os métodos clássicos de traçado de raios (*Whitted's Ray Tracing*) (Whitted, 1980). Sua implementação está disponível na biblioteca PBRT (Pharr e Humphreys, 2004), um sistema de *rendering* baseado em física.

5.6.1 *Volume Scattering*

O método *Volume Scattering* descreve a interação da luz com meios participantes, tais como líquidos ou fumaça. Este método é baseado em métodos clássicos de traçado de raios (*Ray Tracing*) (Whitted, 1980), cujos dados de entrada são: (a) Posição do observador; (b) Posição e orientação do plano de projeção e da janela de visualização; (c) Geometria da cena; (d) Propriedades materiais dos objetos da cena; (e) Propriedades das fontes de luz.

Desta forma, a metodologia do *Ray Tracing* consiste em, para cada pixel da janela de visualização, traçar um raio passando pelo pixel e pela posição do observador. Este raio atua como um raio de luz virtual, refletindo e refratando nos objetos da cena, até um limite pré estabelecido de reflexões. Uma vez terminado o traçado do raio, efetua-se recursivamente o cálculo da radiância total ao longo do raio. Esta radiância total que dará a cor do pixel.

O cálculo da radiância é feito através de um modelo físico de interação entre a luz e os objetos da cena (Rogers, 1998). Para o método *Volume Scattering* o modelo físico tem que levar em conta as interações luminosas em meios participantes. O modelo físico é dado pela equação de transporte (Pharr e Humphreys, 2004):

$$L_i(p,\omega) = T_r(p_0 \rightarrow p)L_o(p_0,-\omega) + \int_0^t T_r(p' \rightarrow p)S(p',-\omega)dt', \qquad (5.12)$$

onde p_0 é o ponto de interseção do raio com uma superfície, e p' são pontos ao longo do raio, como ilustra a Figura 5.12.

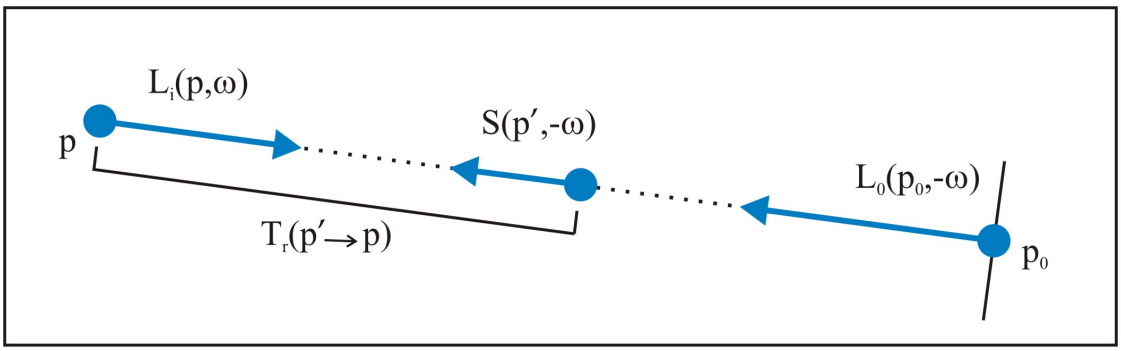


Figura 5.12: Componentes da Equação de Transporte (raio finito). Fonte: Pharr e Humphreys (2004), p. 805

Na equação 5.12, $L_o(p_0, -\omega)$ é a radiância refletida no ponto p_0 de interseção do raio com uma superfície, na direção $-\omega$. Esta radiância sofrerá atenuação ao longo do raio até o ponto de interseção, atenuação essa dada por $T_r(p_0 \rightarrow p)$. O segundo termo da soma em 5.12 nos dá a radiância ao longo do raio através do meio participante, o qual considera as radiâncias adicionadas devido aos efeitos de emissão e *in-scattering*. A radiância através do meio participante é computada levando-se em conta somente os pontos ao longo do raio até encontrar o ponto de interseção com a superfície (Pharr e Humphreys, 2004).

A radiância total adicionada ao longo do raio, devido ao *in-scattering* e efeitos de emissão, é dada pelo termo fonte $S(p', -\omega)$ (Pharr e Humphreys, 2004):

$$S(p', -\omega) = L_{ve}(p', -\omega) + \sigma_s(p', -\omega) \int_{S^2} L(p', \omega_i) f(p', -\omega, \omega_i) d\omega_i, \qquad (5.13)$$

onde $L_{ve}(p', -\omega)$ é a radiância emitida no ponto p' na direção $-\omega$, σ_s é o coeficiente de espalhamento, $L(p', \omega_i)$ é a radiância incidente na direção ω_i, e f é uma função que modela o espalhamento de luz no ponto p'. A integral no segundo membro da equação 5.13 é tomada sobre a esfera unitária S^2 centrada no ponto p', para computar a parcela de radiância refletida na direção $-\omega$.

A atenuação T_r, presente na equação 5.12, nos dá a fração da radiância que é transmitida entre dois pontos ao longo do raio (Pharr e Humphreys, 2004):

$$T_r(p_1 \rightarrow p_2) = e^{-\int_0^d \sigma_t(p_1,\omega)dt}, \tag{5.14}$$

onde d é a distância entre p_1 e p_2, ω é o vetor normalizado de direção entre os pontos, e σ_t é o coeficiente de atenuação, dado por (Pharr e Humphreys, 2004):

$$\sigma_t(p,\omega) = \sigma_a(p,\omega) + \sigma_s(p,\omega), \tag{5.15}$$

onde σ_a é o coeficiente de absorção intrínseco ao meio participante, e σ_s é o coeficiente de espalhamento. Em meios homogêneos o coeficiente de atenuação σ_t é considerado constante, simplificando a equação 5.14 por:

$$T_r(p_1 \rightarrow p_2) = e^{-\sigma_t d}. \tag{5.16}$$

Os parâmetros σ_a, σ_s, bem como a função f e o termo L_{ve}, são características dos materiais da cena. Na seção 7.2 veremos um exemplo de aplicação deste modelo.

Capítulo 6

Modelos Propostos e Implementação

No contexto da computação gráfica, existem três aspectos que devem ser considerados no desenvolvimento de uma animação: realismo físico, realismo visual e eficiência computacional.

Para os casos em que o realismo físico não é o foco principal, técnicas menos precisas, do ponto de vista físico, mas que geram um resultado visual semelhante àqueles obtidos por modelos mais elaborados, são objetos de muito interesse dentro da computação gráfica.

Xavier et al (2005) propõe o uso de modelos LGCAs para a animação de fluidos em computação gráfica, uma vez que tais métodos são eficientes do ponto de vista computacional. Dando continuidade ao trabalho desenvolvido por Xavier et al (2005), o presente trabalho propõe um modelo de fluido tridimensional através de modelos bidimensionais baseados em *lattice* e técnicas de interpolação.

Nas próximas seções apresentaremos as contribuições deste trabalho, bem como descreveremos a metodologia de implementação. Especificamente, as contribuições deste trabalho envolvem a implementação de um modelo de fluido $3D$ baseado em modelos $2D$, combinação do modelo $3D$ com DTMs em uma aplicação em tempo real utilizando diversas técnicas de *rendering* (vide seção 5.5), combinação do modelo $3D$ com a técnica de *rendering Volume Scattering*(vide seção 5.6.1) em uma aplicação *off-line*, bem como desenvolvimento de aplicativos para facilitar a manipulação dos parâmetros das simulações e animações.

6.1 Modelo de fluido 3D

Nas seções 4.2 e 4.3.2 introduzimos respectivamente os modelos FHP e D2Q9, ambos modelos bidimensionais. Nossa proposta é desenvolver um modelo de fluido tridimensional baseado nesses modelos bidimensionais e em técnicas de interpolação (Judice e Giraldi, 2008a,b).

Primeiramente, precisamos determinar um domínio tridimensional onde os modelos serão simulados, como mostra a Figura 6.1(a). A idéia consiste em regularmente distribuir planos ao longo dos eixos da base (no caso, ao longo dos eixos x e z), como mostra a Figura 6.1(b). Cada um desses planos representa um modelo de simulação bidimensional, ou seja, cada um dos planos é um modelo FHP ou um modelo *Lattice Boltzmann* D2Q9. Desta forma, a simulação em cada plano é executada independentemente, seguindo as regras explicadas na seção 4.2 no caso do FHP, ou as regras na seção 4.3.2 para o modelo D2Q9.

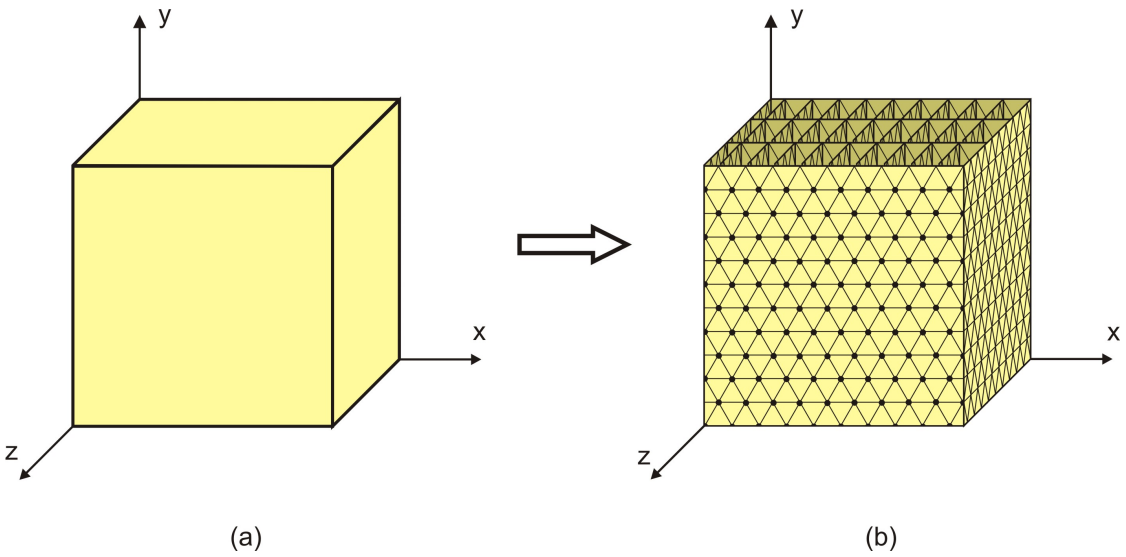


Figura 6.1: Modelo de fluido $3D$. (a) Domínio no espaço tridimensional. (b) Distribuição dos planos bidimensionais.

Especificamente no caso do FHP é preciso fazer uma interpolação simples para gerar o fluxo macroscópico $2D$ em cada plano (como discutido na seção 4.2.2). Nesse passo, novos nós são adicionados com o intuito de completar uma *lattice* retangular em cada plano, como mostra a Figura 6.2.

Para cada um desses novos nós a densidade macroscópica é estimada como a média das densidades de seus quatro vizinhos mais próximos (norte, sul, leste, oeste). No caso do modelo D2Q9 não é necessário adicionar novos nós pois o modelo já trabalha com uma malha retangular, bem como não é preciso estimar as quantidades macroscópicas uma vez que o modelo já atualiza essas quantidades a cada passo de interação.

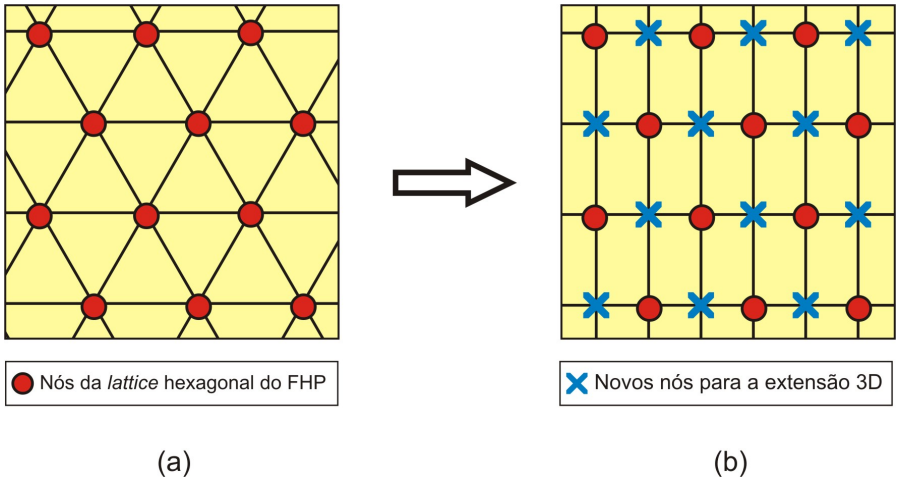


Figura 6.2: (a) Malha do FHP. (b) Extensão da *lattice* hexagonal do FHP (nós em azul).

Assim que a simulação é realizada independentemente em cada plano, é preciso gerar o fluxo macroscópico $3D$. Uma vez que cada nó (x_i, y_i, z_i) da malha gerada pertence aos planos de simulação $x = x_i$ e $z = z_i$, o campo macroscópico de densidade pode ser obtido através de uma interpolação simples de valores de densidade nos planos, ou seja:

$$\rho(x_i, y_i, z_i, t) = \frac{\rho_x(x_i, y_i, z_i, t) + \rho_z(x_i, y_i, z_i, t)}{2}, \tag{6.1}$$

onde ρ_x e ρ_z são as densidades macroscópicas no nó corrente nos planos $x = x_i$ e $z = z_i$ respectivamente. Uma expressão análoga pode ser obtida para estimar o campo macroscópico de velocidade:

$$\vec{u}(x_i, y_i, z_i, t) = \frac{\vec{u_x}(x_i, y_i, z_i, t) + \vec{u_z}(x_i, y_i, z_i, t)}{2}, \tag{6.2}$$

onde $\vec{u_x}$ e $\vec{u_z}$ são as velocidades macroscópicas no nó corrente nos planos $x = x_i$ e $z = z_i$ respectivamente.

A Figura 6.3 apresenta o fluxograma de execução do modelo de fluido $3D$ proposto. A etapa de inicialização do método engloba a definição do domínio $3D$, a distribuição dos planos nesse domínio e a inicialização de cada um desses planos. É na etapa de simulação que os modelos bidimensionais irão evoluir independentemente em cada plano, e os campos macroscópicos $3D$ serão calculados a cada passo de tempo t.

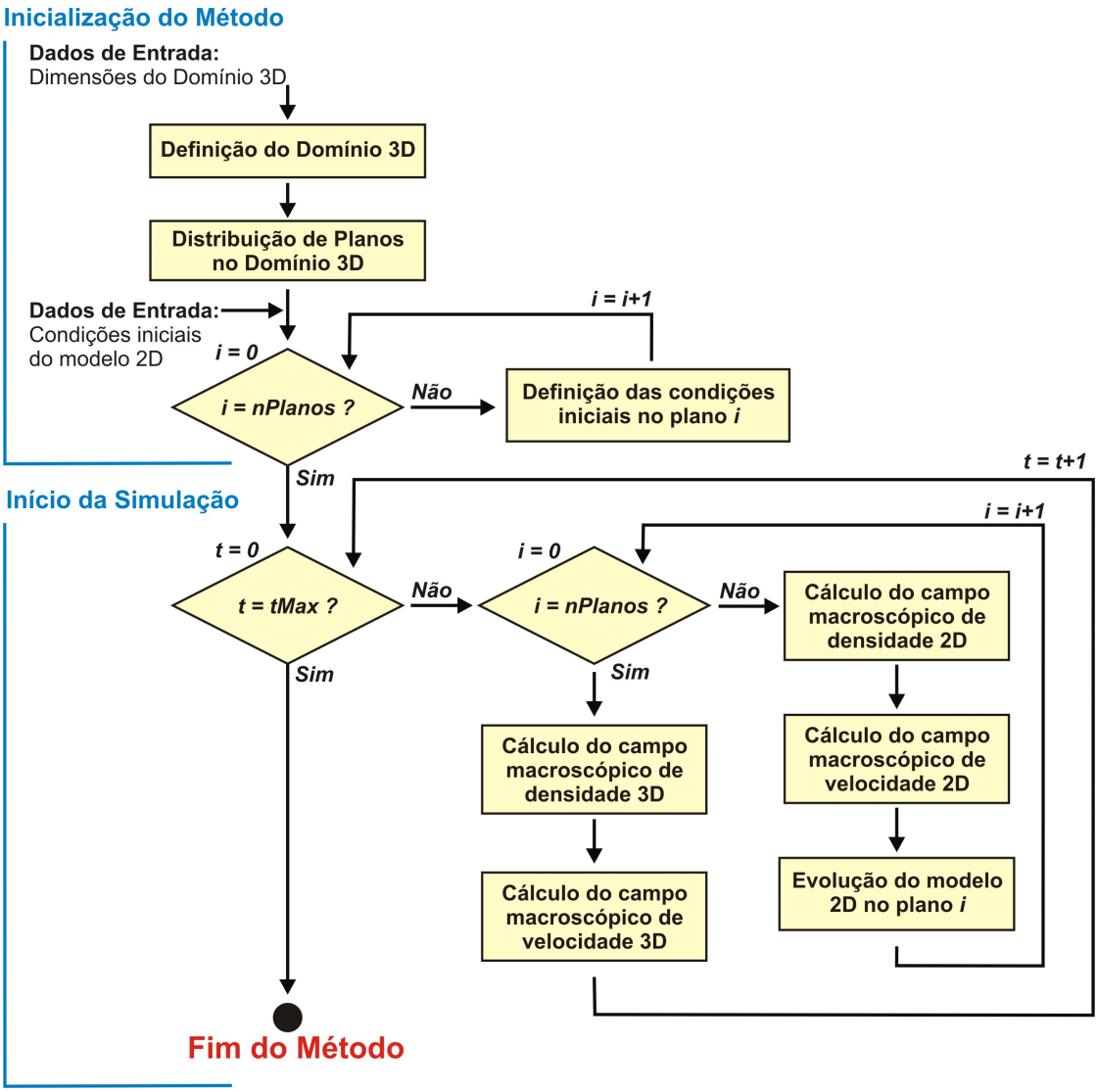


Figura 6.3: Fluxograma de execução do modelo de fluido 3D proposto, onde a variável *nPlanos* indica a quantidade de planos distribuídos e *tMax* a condição de parada.

6.2 Sistema de Partículas

Para a visualização do campo macroscópico de velocidade no domínio tridimensional, descrito na expressão (6.2), propomos a utilização de um conjunto de partículas macroscópicas. Tais partículas residem no domínio tridimensional e suas posições são definidas aleatoriamente.

Para cada partícula macroscópica p, tomamos uma vizinhança $V(p,r)$ de raio r. Desta forma, estimamos a velocidade da partícula $\vec{u}(p,t)$ no instante t através da média das velocidades macroscópicas $\vec{u}$ (equação (6.2)) nos nós da *lattice* interiores à vizinhança $V(p,r)$:

$$\vec{u}(p,t) = \frac{1}{N_V} \sum_{\vec{x} \in V(p,r)} \vec{u}(\vec{x},t), \tag{6.3}$$

onde N_V é o número de nós da *lattice* presente na vizinhança $V(p,r)$.

A propagação de uma dada partícula macroscópica p na posição $\vec{x}(p,t)$ no instante t para uma nova posição no instante seguinte é calculada por:

$$\vec{x}(p,t+\Delta t) = \vec{x}(p,t) + \sigma \vec{u}(p,t), \tag{6.4}$$

onde σ é uma constante.

6.3 Implementação

Os aplicativos desenvolvidos neste trabalho foram implementados usando-se a linguagem de programação C/C++. Para a visualização dos resultados utilizou-se OpenGL (Shreiner, 2004; Wright et al, 2007), uma biblioteca de rotinas gráficas já tradicional em aplicações gráficas. Utilizou-se também a biblioteca GLUT (Kilgard, 1996; Baker, 2006) para a interface gráfica do *software* desenvolvido. Uma grande vantagem da GLUT é permitir o uso das funções gráficas do OpenGL e ainda tornar padronizado o acesso a características específicas de cada ambiente de janelas disponíveis para sistemas operacionais como Unix e Windows.

Para criação de botões, *checkboxes*, *spinners*, entre outros, recorre-se à biblioteca GLUI (Rademacher, 1999; Stewart, 2006), uma extensão da GLUT e, portanto, independente do sistema usado. Dessa forma, o programa desenvolvido pode ser executado em qualquer ambiente com suporte para OpenGL.

A implementação seguiu o paradigma de programação orientada a objetos. Desta forma, criamos classes para representar os modelos bidimensionais FHP e D2Q9 bem como para representar o modelo $3D$ proposto na seção 6.1. Essas classes serão detalhadas nas seções 6.3.1 e 6.3.2.

6.3.1 Classe da Dinâmica Bidimensional

A classe do FHP está representada na Figura 6.4 e contém os atributos necessários para a sua criação e evolução.

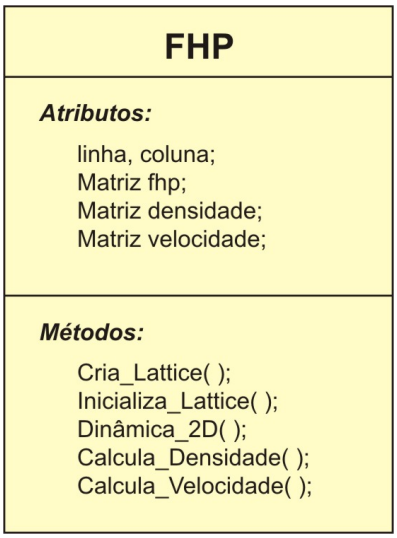


Figura 6.4: Classe do modelo FHP.

Precisamos armazenar as dimensões da *lattice* bidimensional, definida pelas variáveis **linha** e **coluna**. A matriz **fhp** representa a *lattice* $2D$ cuja dimensão é (*linha* $\times$ *coluna*), e cada uma de suas células (fhp$[i][j]$) armazena seis variáveis *booleanas* correspondentes às seis direções da *lattice* hexagonal. As matrizes **densidade** e **velocidade**, também de dimensão (*linha* $\times$ *coluna*), armazenam as quantidades macroscópicas de densidade e velocidade, respectivamente.

Entre os métodos da classe, temos as funções de inicialização: **Cria_Lattice**, responsável por alocar espaço em memória para todas as matrizes, e **Inicializa_Lattice**, responsável por definir os valores iniciais em cada célula de cada matriz. A condição inicial para a matriz fhp é dada através dos números de ocupação n_i (vide seção 4.2.1).

Os demais métodos de classe são responsáveis pela simulação. A função **Dinâmica_2D** atualiza a matriz fhp através das regras explicadas na seção 4.2.1 sobre a microdinâmica do FHP, e está descrita no Algoritmo 1 (correspondente a um único passo de evolução da dinâmica do FHP).

No Algoritmo 1, o primeiro laço de repetição é responsável pela etapa de colisão, dada pelo termo de colisão Ω_i presente na equação de evolução do FHP (expressão (4.20)). O último laço de repetição computa a propagação das partículas, dada pela equação (4.20).

Por fim, as funções **Calcula_Densidade** e **Calcula_Velocidade** são responsáveis por atualizar as matrizes de densidade e velocidade através das expressões (4.23) e (4.24), respectivamente. Apesar da microdinâmica do FHP não depender das variáveis macroscópicas de densidade e velocidade (equação (4.20)), o cálculo delas é importante para estimar os campos macroscópicos no domínio tridimensional.

Algorithm 1 Dinâmica_2D()

for $i = 0$ até *linha* **do**
  for $j = 0$ até *coluna* **do**
    Para cada uma das 6 direções do nó $fhp[i][j]$, verificar as condições de colisão entre partículas.
  end for
end for

for $i = 0$ até *linha* **do**
  for $j = 0$ até *coluna* **do**
    Para cada uma das 6 direções do nó $fhp[i][j]$, efetuar a propagação das partículas.
  end for
end for

A classe do D2Q9 está representada na Figura 6.5. Ela armazena as dimensões da *lattice* bidimensional, definida pelas variáveis **linha** e **coluna**. A matriz **d2q9** representa a *lattice* $2D$ de dimensão ($linha \times coluna$), e cada uma de suas células (d2q9$[i][j]$) armazena nove variáveis ponto flutuante correspondentes às nove direções da *lattice*. As matrizes **densidade** e **velocidade**, também de dimensão ($linha \times coluna$), armazenam as quantidades macroscópicas de densidade e velocidade, respectivamente.

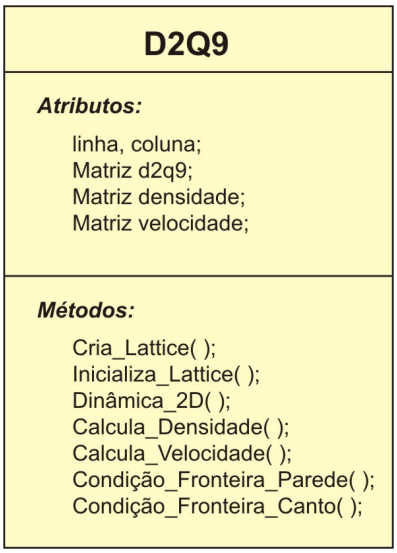


Figura 6.5: Classe do modelo D2Q9.

As funções de inicialização denominadas **Cria_Lattice** e **Inicializa_Lattice** são análogas às da Classe do FHP. A condição inicial da matriz d2q9 é dada da seguinte forma: define-se um valor inicial de densidade (ρ) e velocidade ($\vec{u}$) para cada nó da *lattice* e armazena-se tais valores nas matrizes correspondentes. Em seguida, para cada uma das nove direções de cada célula da matriz d2q9 aplica-se a função de distribuição de equilíbrio descrita pela expressão (4.32), para definir um valor de distribuição inicial nas direções de cada nó.

Os demais métodos de classe são responsáveis pela simulação. A função **Dinâmica_2D** atualiza a matriz d2q9 através das regras explicadas na seção 4.3.2, e está descrita no Algoritmo 2 (correspondente a um único passo de evolução da dinâmica do D2Q9).

No Algoritmo 2, as etapas de colisão e propagação descritas no laço de repetição são dadas pelas equações (4.33) e (4.34), respectivamente. O cálculo da função de distribuição (expressão (4.27)) no laço de repetição é feito para os nós internos da matriz, sem levar em conta os nós presentes nas fronteiras. A função **Condição_Fronteira** trata as condições de fronteiras explicadas na seção 4.3.3.

Por fim, as funções **Calcula_Densidade** e **Calcula_Velocidade** são responsáveis por atualizar as matrizes de densidade e velocidade através das expressões (4.30) e (4.31), respectivamente. É importante lembrar que a regra de evolução do modelo D2Q9, descrita pela função de distribuição (4.27), é dependente das variáveis macroscópicas de densidade e velocidade. Portanto, as funções **Calcula_Densidade** e **Calcula_Velocidade** precisam ser executadas antes do Algoritmo 2, a cada passo de iteração.

Algorithm 2 Dinâmica_2D()

for $i = 1$ até *linha-1* **do**
  for $j = 1$ até *coluna-1* **do**
    Para cada uma das 9 direções do nó $d2q9[i][j]$ calcular a função de distribuição, que envolve etapas de colisão e propagação.
  end for
end for

Condicao_Fronteira();

6.3.2 Classe da Dinâmica Tridimensional

A classe do modelo $3D$ proposto na seção 6.1 está representada na Figura 6.6. Como atributos de classe temos os limites do domínio $3D$, definidos pelas variáveis N_x, N_y e N_z (*largura*, *altura* e *profundidade*); e as variáveis **NumPlanX** e **NumPlanZ** que definem a quantidade de planos distribuídos ao longo dos eixos x e z, respectivamente. Para armazenar os planos, criamos duas estruturas de dados do tipo **lista**, onde: *ListPlanX* é uma lista de tamanho *NumPlanX* que armazena os planos distribuídos ao longo do eixo x, e *ListPlanZ* é uma lista de tamanho *NumPlanZ* que armazena os planos ao longo do eixo z.

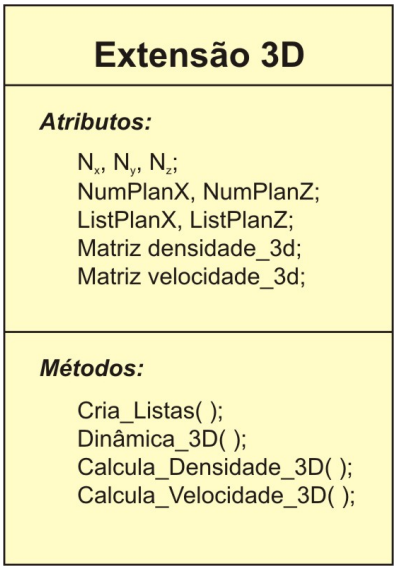


Figura 6.6: Classe do Modelo de Fluido $3D$ Proposto.

O tipo de dado armazenado nas listas vai depender do modelo de simulação bidimensional escolhido. Se a simulação for feita com o modelo FHP, isto significa que cada plano representará uma *lattice* deste modelo e, portanto, as listas armazenarão objetos do tipo definido pela classe implementada para o FHP (Figura 6.4). Em contrapartida, se a simulação for feita com o modelo D2Q9, as listas armazenarão objetos definidos pela classe representada na Figura 6.5.

Por fim, temos ainda como atributos de classe as matrizes denominadas **densidade_3d** e **velocidade_3d**, ambas de dimensão $(N_x \times N_y \times N_z)$, responsáveis por armazenar as quantidades macroscópicas de densidade e velocidade $3D$, dadas pela equação (6.1) e pela equação (6.2), respectivamente.

Entre os métodos de classe, a função de inicialização denominada **Cria_Listas** é a responsável por inicializar o modelo $3D$. Esta etapa se dá através da inicialização de cada plano armazenado nas listas, uma vez que o modelo tridimensional é composto por planos que representam modelos bidimensionais. Para tanto, é necessário um laço de repetição para percorrer cada uma das listas, como mostra o Algoritmo 3. Para cada objeto armazenado em cada lista chamamos sua função **Cria_Lattice** e **Inicializa_Lattice**, definidas na seção 6.3.1.

Algorithm 3 Cria_Listas()

```
for i = 0 até NumPlanX do
   ListPlanX[i].Cria_Lattice();
   ListPlanX[i].Inicializa_Lattice();
end for

for i = 0 até NumPlanZ do
   ListPlanZ[i].Cria_Lattice();
   ListPlanZ[i].Inicializa_Lattice();
end for
```

Os demais métodos de classe são responsáveis pela simulação. A função **Dinâmica_3D** é responsável por gerenciar a evolução em cada plano independentemente e estimar as quantidades macroscópicas no domínio tridimensional. Tal função está descrita no Algoritmo 4, onde t representa o tempo e t_{max} fornece a condição de parada. A função **Calcula_Densidade_3D** corresponde a expressão (6.1) e a função **Calcula_Velocidade_3D** corresponde a expressão (6.2), responsáveis por atualizar as matrizes que armazenam as quantidades macroscópicas tridimensionais.

Algorithm 4 Dinâmica_3D()

```
for t = 0 até t_max do

   for i = 0 até NumPlanX do
      ListPlanX[i].Calcula_Densidade();
      ListPlanX[i].Calcula_Velocidade();
      ListPlanX[i].Dinâmica_2D();
   end for

   for i = 0 até NumPlanZ do
      ListPlanZ[i].Calcula_Densidade();
      ListPlanZ[i].Calcula_Velocidade();
      ListPlanZ[i].Dinâmica_2D();
   end for

   Calcula_Densidade_3D();
   Calcula_Velocidade_3D();

end for
```

6.4 Aplicação em Tempo Real

A proposta da aplicação em tempo real consiste em simular uma chuva sobre um modelo digital de terreno (DTM), bem como a interação da água acumulada sobre diferentes topografias de terreno. Veremos como é feito a interação da chuva com o modelo digital de terreno na seção 6.4.1.

A Figura 6.7 mostra o aplicativo desenvolvido para a animação em tempo real. Na Figura 6.7(a) podemos ver a janela de visualização onde é apresentado um frame da animação gerada através das técnicas descritas na seção 5.5. Este aplicativo destina-se a geração de animações utilizando o modelo $3D$ proposto na seção 6.1 para simular a dinâmica da chuva. Para tornar a animação mais realista, utilizamos o sistema de partículas descrito na seção 6.2 para representar as gotas que compõem a chuva, bem como as técnicas de *rendering* descritas nas seções 5.5.1-5.5.3. Na Figura 6.7(b) temos a janela onde o usuário pode definir parâmetros, escolher o cenário e o modelo de terreno, bem como manipular a evolução do sistema.

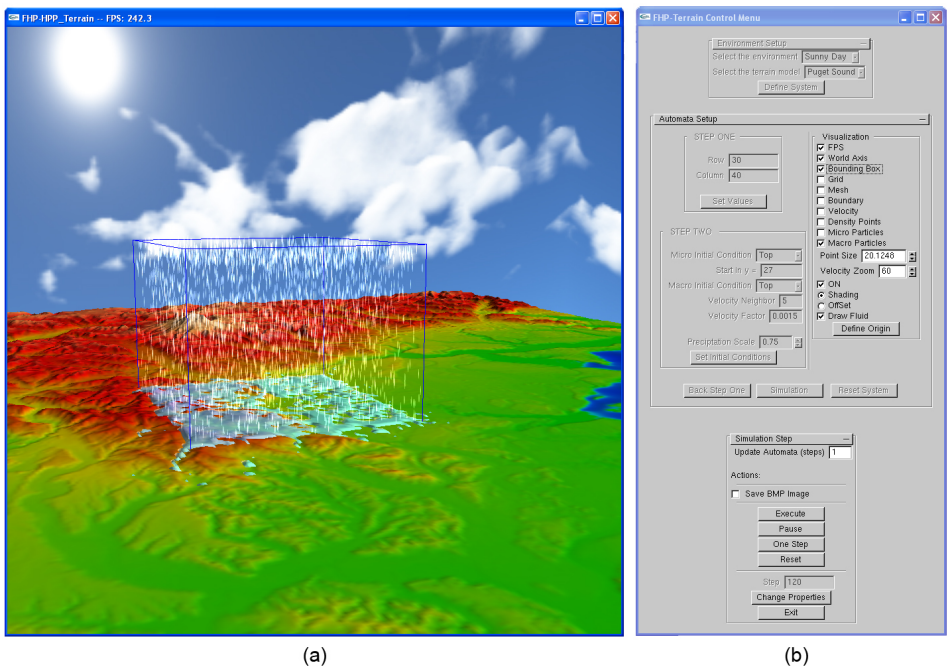


Figura 6.7: Aplicativo desenvolvido para animação em tempo real: (a) Ambiente de Visualização; (b) Janela de interação com o usuário.

6.4.1 Animação de Fluido sobre Modelos de Terreno

O método de animação de fluido sobre modelos digitais de terreno (DTM) utilizado neste trabalho foi proposto por Barcellos et al (2007). Tal método é composto por dois elementos básicos: um Modelo Digital de Elevação (DEM), para representar o terreno; e um modelo de LGCA, para simular a animação do fluido.

O DEM é composto por uma *lattice* regular e por uma função de elevação $\varphi(i, j)$. Para um dado nó (i, j) da *lattice* tal função retorna a elevação do terreno naquele ponto, ou seja, a função retorna o valor da componente y relativa à altura do terreno. O LGCA é um modelo baseado em partícula e, nesse caso, utilizará a *lattice* regular bidimensional do DEM para a simulação do fluido. A Figura 6.8 ilustra as *lattices* e a relação existente entre elas.

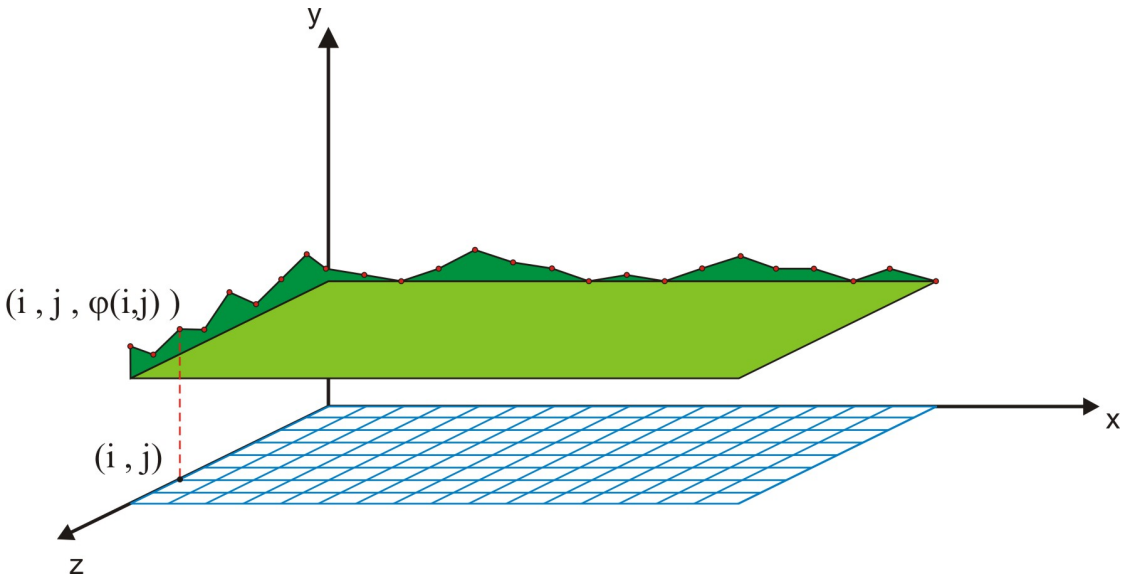


Figura 6.8: Representação do modelo de terreno.

Na implementação utilizada, ambas as *lattices* possuem a mesma dimensão. A Figura 6.9 ilustra um nó (i, j) da *lattice* e seus quatro vizinhos, denotados por $V1$, $V2$, $V3$ e $V4$. O vetor $\vec{s}$ denota o declive do terreno em um dado ponto $(i, j, \varphi(i, j))$, definido como a projeção da normal à superfície sobre a *lattice* bidimensional.

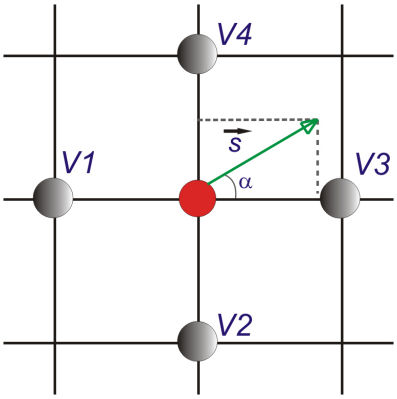


Figura 6.9: Vizinhança de um nó da *lattice*.

Para cada nó (i, j) da *lattice* está associado um contador de partículas, o qual representa o acúmulo de água no ponto $(i, j, \varphi(i, j))$ correspondente do terreno. A elevação $(\varphi(i, j))$ e o declive $(\vec{s})$ definem os campos estacionários do modelo. A evolução do sistema de partículas atualiza os contadores, que serão os campos não-estacionários. Na seção 6.4.2 veremos de que forma esses contadores são atualizados.

O movimento das partículas é discreto, o que significa que cada partícula se move de um determinado nó da *lattice* para outro nó presente em sua vizinhança no espaço de tempo Δt. Portanto, dada uma partícula na posição (i, j) e o declive $\vec{s}(i, j)$, é possível determinar a direção de movimento desta partícula (vide trabalho de Barcellos et al (2007)).

Basicamente, a partícula se move de um nó (i, j) para um vizinho V_k (onde $k \in \{1, 2, 3, 4\}$) se a direção da *lattice* correspondente $V_k - (i, j)$ for a mais próxima à direção do declive $\vec{s}(i, j)$. Outro fator a se levar em conta é que a altura da superfície livre em (i, j) precisa ser maior do que em V_k. O cálculo da altura h da superfície livre em um dado nó (i, j) num dado instante t de tempo é dado por:

$$h(i, j, t) = \varphi(i, j) + \beta \cdot contador(i, j, t), \tag{6.5}$$

onde β é um parâmetro de escala e *contador* retorna a quantidade de partículas acumulada no ponto (i, j) no instante t.

O método proposto por Barcellos et al (2007) é livre de instabilidade numérica uma vez que os campos que apresentam valores em ponto flutuante são estacionários e a atualização dos contadores de partículas é baseada em comparações em ponto flutuante e aritmética inteira. Além disso, a conservação de massa é garantida, uma vez que as operações são conservativas com relação ao número de partículas (Barcellos et al, 2007, 2008). Este método pode ser implementado em GPU, como descrito no Apêndice B.

6.4.2 Interação entre os Modelos

Os modelos de simulação estão implementados em CPU, enquanto o modelo de fluxo sobre o terreno está implementado em GPU. Para tanto, é necessário a implementação de uma interface de comunicação entre estes modelos. Uma vez definidas todas as condições iniciais tem início o processo de evolução do sistema. A Figura 6.10 representa o modelo de comunicação entre a CPU e GPU, implementado neste trabalho. Tal modelo é executado a cada passo de tempo.

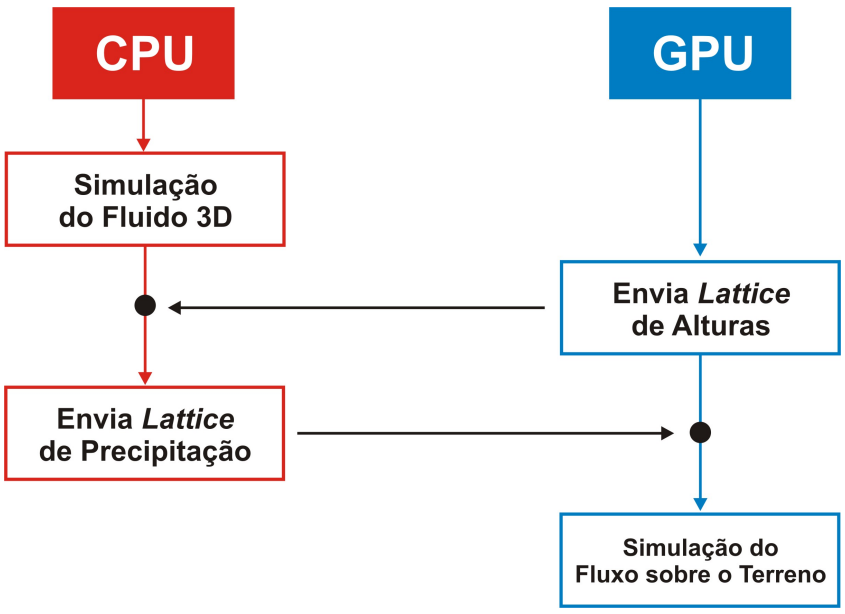


Figura 6.10: Comunicação entre CPU e GPU (a cada passo de tempo).

A tarefa chamada **Simulação do Fluido 3D**, na Figura 6.10, é responsável pela simulação da chuva através do modelo $3D$ explicado na seção 6.1. É nesta etapa que ocorrem os cálculos dos campos macroscópicos no domínio tridimensional, regidos pela expressão (6.1) para o campo de densidade e pela expressão (6.2) para o campo de velocidade, bem como a evolução do sistema de partículas através das expressões (6.3) e (6.4).

Entretanto, é preciso verificar se uma dada partícula macroscópica alcançou os limites do terreno. Para tal, o processo na GPU (descrito no Apêndice B) envia para o processo em execução na CPU uma matriz com informações sobre a altura do terreno em cada nó (i, j), obtida através da expressão (6.5). Com esta informação, o sistema verifica todas as novas posições de cada partícula macroscópica: se encontrou o terreno ela é excluída do sistema de partículas e uma matriz de precipitação é atualizada exatamente no nó (i, j) correspondente ao ponto onde a partícula foi excluída.

Uma vez verificadas todas as partículas macroscópicas, o processo em CPU envia para o processo em GPU a matriz com as informações de precipitação sobre o terreno. Tais informações serão utilizadas para atualizar os contadores de partículas, utilizados na expressão (6.5). Desta forma, o processo em GPU é capaz de simular o comportamento do fluido sobre o terreno, ilustrado na Figura 6.10. Por fim, o processo em GPU precisa atualizar os valores da elevação da superfície livre em cada ponto do terreno, através da expressão (6.5).

6.4.3 Condição Inicial

O aplicativo implementado para execução em tempo real utiliza o modelo $3D$ proposto na seção 6.1 para simular a dinâmica da chuva, o modelo baseado em GPU (Apêndice B) para simular a superfície livre e um DTM (seção 6.4.1) para representar o terreno. O modelo de fluido $3D$ é composto pelos modelos bidimensionais FHP ou D2Q9.

A condição inicial do modelo FHP é dada através dos números de ocupação n_i que determinam se há ou não partícula na direção i de movimento. Para esta aplicação, delimitamos um pequeno volume no topo do domínio $3D$ onde as partículas do FHP foram inicializadas aleatoriamente, como mostra a Figura 6.11(a).

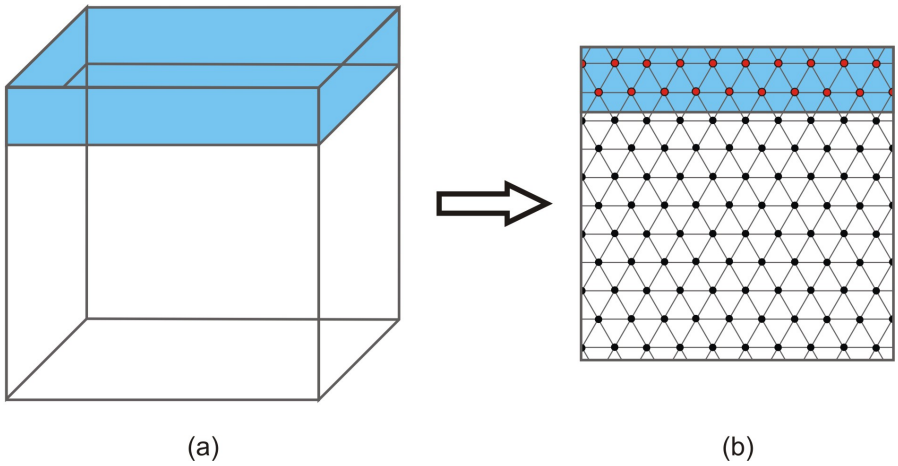


Figura 6.11: Condição Inicial para o FHP na aplicação em tempo real. (a) Delimitação de um pequeno volume no topo do domínio $3D$. (b) Região da *lattice* do FHP onde serão inicializadas as partículas microscópicas.

Para cada nó da *lattice* dentro deste volume (Figura 6.11(b)) foi gerado um número aleatório entre 0 e 6, representando a quantidade de partículas que serão inicializadas naquele nó, onde 0 significa que nenhuma partícula é definida para o nó em questão e 6 significa que todas as direções serão ocupadas por partículas. Além disso, para cada partícula é escolhida aleatoriamente a sua velocidade. Uma vez inicializadas as partículas do FHP, à medida que o sistema evolui, novas partículas são geradas nos nós da *lattice* dentro desse volume, com o intuito de manter o fluxo da chuva.

No caso do método *Lattice Boltzmann* D2Q9, a condição inicial é especificada em termos das variáveis macroscópicas de densidade e velocidade. Uma vez definidas tais quantidades macroscópicas, a distribuição f_i inicial em cada direção i de movimento é dada da seguinte forma: para cada uma das 9 direções de cada nó da *lattice* do modelo D2Q9 é calculada a função de distribuição de equilíbrio (expressão (4.32)).

A visualização da chuva é implementada através de um conjunto de partículas macroscópicas, cuja condição inicial é especificada em termos das posições de cada partícula (velocidade inicial é nula). Tais partículas residem no domínio tridimensional e suas posições iniciais são definidas aleatoriamente dentro do volume definido na Figura 6.11(a). A dinâmica dessas partículas segue a metodologia explicada na seção 6.2, a qual depende do campo de velocidade macroscópico gerado através dos modelos de simulação. Se uma dada partícula, ao se mover, ultrapassar os limites que contém as *lattices*, essa partícula é excluída do sistema.

6.5 Aplicação *Off-Line*

A proposta da aplicação para execução *off-line* consiste em simular a dinâmica de um volume de gás e visualizar o resultado desta simulação através da técnica *Volume Scattering*, descrita na seção 5.6.1. Tal técnica está disponível na biblioteca PBRT (Pharr e Humphreys, 2004), um sistema de *rendering* baseado em física utilizado neste trabalho. A Figura 6.12 mostra o aplicativo desenvolvido para a animação *off-line*.

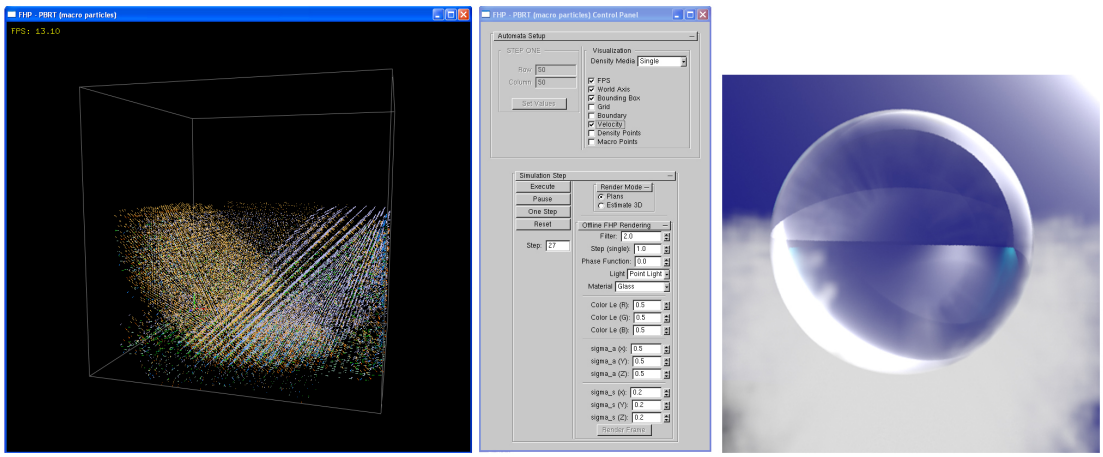


Figura 6.12: Aplicativo desenvolvido para animação *off-line*: (a) Janela de Visualização; (b) Tela de interação com o usuário; (c) Resultado da cena após processo de renderização.

Na Figura 6.12(a) podemos ver a janela de visualização onde é apresentado um frame da simulação gerado através do modelo $3D$ (seção 6.1) para simular a dinâmica do volume de gás. Uma vez realizada a simulação, é preciso passar às classes do PBRT todas as informações contidas no frame para renderizar a cena. Na Figura 6.12(b) temos a interface gráfica, onde o usuário pode definir os parâmetros necessários à simulação bem como pode manipular a evolução do sistema. Na Figura 6.12(c) temos uma cena resultante após renderização via *volume scattering*.

6.5.1 Condição Inicial

Analogamente à aplicação em tempo real, a aplicação *off-line* utiliza o modelo $3D$ proposto na seção 6.1 para simular a dinâmica do volume de gás. E tal modelo é composto pelos modelos bidimensionais FHP ou D2Q9. Como visto na seção 6.4.3, a condição inicial do modelo FHP é dada através dos números de ocupação n_i que determinam se há ou não partícula na direção i de movimento.

Para a aplicação *off-line* delimitamos um pequeno volume cilíndrico na base do domínio $3D$ onde as partículas do FHP foram inicializadas aleatoriamente, seguindo a metodologia descrita na seção 6.4.3, como mostra a Figura 6.13(a).

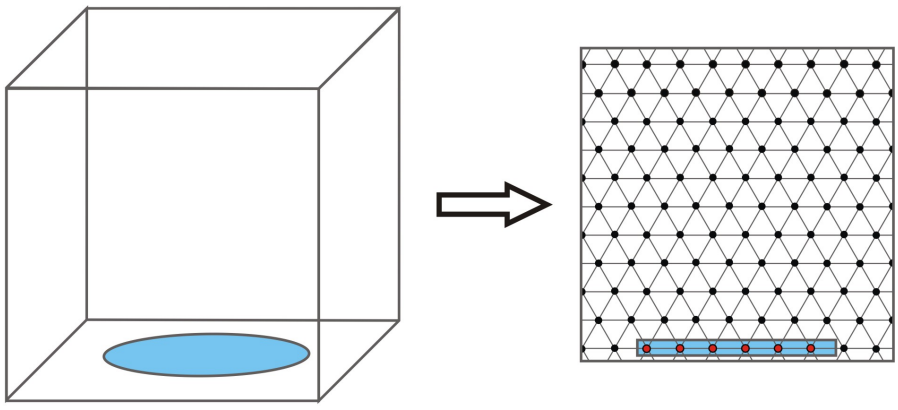


Figura 6.13: Condição Inicial para o FHP na aplicação *off-line*. (a) Delimitação de uma pequena área cilíndrica na base do domínio $3D$. (b) Nós da *lattice* do FHP dentro do cilindro serão inicializados aleatoriamente.

Uma vez inicializadas as partículas do FHP, à medida que o sistema evolui, novas partículas são geradas nos nós da *lattice* dentro dessa pequena área circular, com o intuito de manter o fluxo do volume de gás.

Para o modelo $3D$ baseado no método D2Q9, os nós da *lattice* correspondente, dentro deste volume, serão inicializados seguindo a metodologia descrita na seção 6.4.3. A escolha por uma região cilíndrica está relacionada com resultados de evolução do volume de gás mais realistas, como veremos na seção 7.2.

Partículas macroscópicas podem ser utilizadas para visualizar a dinâmica do volume de gás, juntamente com o *volume scattering*. Estas partículas residem no domínio tridimensional e suas posições iniciais são definidas aleatoriamente dentro do volume cilíndrico da Figura 6.13. A dinâmica dessas partículas segue a metodologia explicada na seção 6.2. Analogamente à aplicação em tempo real, se uma dada partícula, ao se mover, ultrapassar os limites que contém as *lattices*, essa partícula é excluída do sistema.

Capítulo 7

Resultados e Discussões

Na seção 4.2 apresentamos o FHP, um autômato celular do tipo *Lattice-Gas*, utilizado para simular fluidos bidimensionais, e na seção 4.3.2 apresentamos o D2Q9, um modelo bidimensional do Método *Lattice Boltzmann*. Com base nesses modelos bidimensionais, propusemos um modelo tridimensional na seção 6.1, para animação de fluidos. Desta forma, com o intuito de mostrar na prática o modelo $3D$ proposto, desenvolvemos uma aplicação em tempo real e uma aplicação *off-line*, descritas nas seções 6.4 e 6.5 respectivamente.

Como visto na seção 6.1, a malha tridimensional é construída através da distribuição de planos bidimensionais. Para simplificar a implementação optamos por construir uma malha tridimensional regular. Para tanto, uma vez definida a resolução ($linha \times coluna$) de cada plano, o aplicativo se encarrega de construir uma malha retangular $3D$ de resolução ($linha \times coluna \times coluna$).

Nas próximas seções discutiremos os resultados obtidos com os aplicativos descritos nas seções 6.4 e 6.5. Os experimentos foram executados no Sistema Operacional Windows XP, em uma máquina com processador Intel Core 2 Quad 3.0 GHz, com 4GB de memória RAM e uma placa de vídeo NVidia GeForce 9800 GTX. Todas as figuras foram obtidas diretamente dos aplicativos desenvolvidos. Os resultados apresentados neste capítulo estão baseados nas referências (Judice e Giraldi, 2008a,b; Judice et al, 2008, 2009) publicadas utilizando-se as técnicas desenvolvidas. As animações geradas estão disponíveis em www.siciliajudice.com.

7.1 Experimentos Computacionais da Aplicação em Tempo Real

Primeiramente, usamos o modelo de fluido $3D$ baseado no FHP para a simulação da chuva sobre um modelo digital de terreno (Judice et al, 2008). Posteriormente, repetimos o mesmo experimento usando o modelo D2Q9 (Judice et al, 2009), com o intuito de comparar os resultados.

O aplicativo desenvolvido para execução em tempo real é composto por: (a) Técnica de animação de fluido $3D$ descrita na seção 6.1; (b) Método de animação de fluido sobre modelos de terreno (vide seção 6.4.1), implementado em GPU e proposto por Barcellos et al (2007); (c) Interação entre os modelos, explicada na seção 6.4.2; (d) Técnicas de *rendering* descritas nas seções 5.5.1-5.5.3.

Descreveremos a seguir os experimentos realizados, cujas condições iniciais são as descritas na seção 6.4.3. Para o modelo D2Q9 os valores iniciais de densidade e velocidade foram $\rho = 1.0$ e $\vec{u} = (u_x, u_y) = (0.0, 0.0)$ para os nós internos da malha, e $\rho = 1.0$ e $\vec{u} = (0.0, -1.0)$ para os nós da fronteira. Tais valores são substituídos na equação (4.32), e os valores resultantes geram a distribuição inicial.

Além disso, para o sistema de partículas macroscópicas que representa a chuva, utilizamos um raio de influência $r = 5$ na expressão (6.3). Entretanto, a constante σ na expressão (6.4) recebe valor $\sigma = 0.0015$ na simulação com o modelo FHP e $\sigma = 0.125$ para o modelo D2Q9. Todos os experimentos evoluíram até 3000 (três mil) passos de tempo. A resolução da malha $3D$ utilizada foi $30 \times 30 \times 30$ nós, e a dimensão da *lattice*, tanto para a simulação do fluxo sobre o terreno quanto para a resolução do próprio terreno foi de 256×256.

Os experimentos a seguir mostram o comportamento do fluido em diferentes modelos de terrenos, bem como enfatiza certos aspectos que podem ser úteis para computação gráfica e aplicações em tempo real, tais como a simplicidade de se inicializar a animação, a simulação de fluidos sobre topografias complexas e a eficiência computacional. Uma comparação entre a dinâmica bem como a eficiência computacional dos modelos FHP e D2Q9 também é descrita.

7.1.1 Comportamento do Fluido sobre o Terreno

Através da interface gráfica o usuário pode definir certos aspectos da animação que se pretende gerar. O primeiro passo é escolher o modelo digital de terreno, que deve ser especificado via um mapa de alturas. Além disso, é possível escolher também em que tipo de cenário o usuário quer simular a chuva. A Figura 7.1 mostra um *screenshot* do aplicativo e quatro combinações de cenários e terrenos geradas através da interface gráfica. A escolha do modelo digital de terreno influencia diretamente na simulação, uma vez que o comportamento do fluido depende das características topográficas de cada modelo.

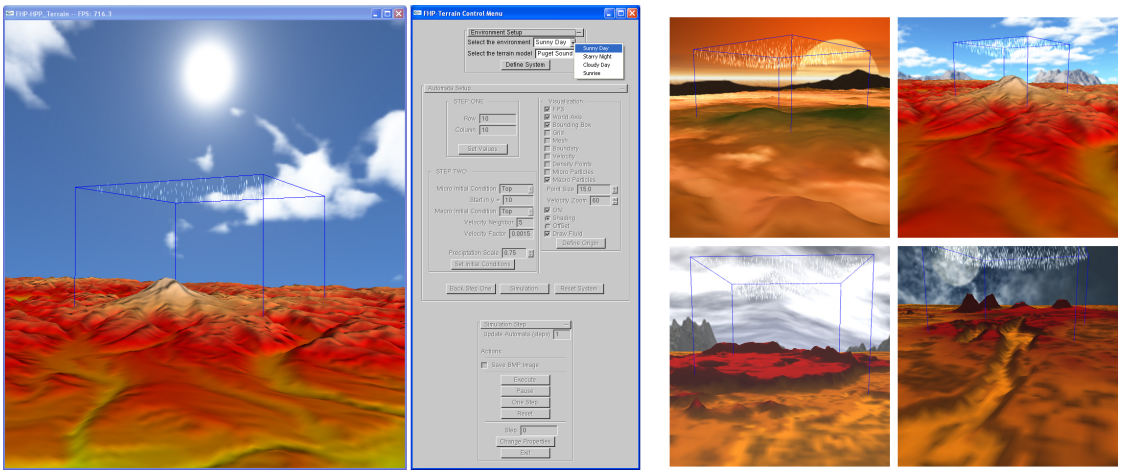


Figura 7.1: Inicializando a animação: escolha de modelo de terrenos e cenários.

A animação terá início com base nas condições iniciais descritas na seção 6.4.3. Para analisar o comportamento do fluido, executamos testes onde a chuva foi simulada tanto com o modelo FHP quanto o D2Q9, utilizando modelos digitais de terreno com topografias diversas.

Na Figura 7.2 podemos ver os primeiros experimentos realizados sobre uma superfície quase plana, com pequenas irregularidades no terreno. Desta forma, podemos ver o comportamento da precipitação da chuva sobre o terreno, cuja tendência é cobrir uma área cada vez maior à medida que a animação evolui. A dinâmica da chuva na Figura 7.2(a) foi obtida através da simulação com o modelo FHP, enquanto na Figura 7.2(b) com o modelo D2Q9.

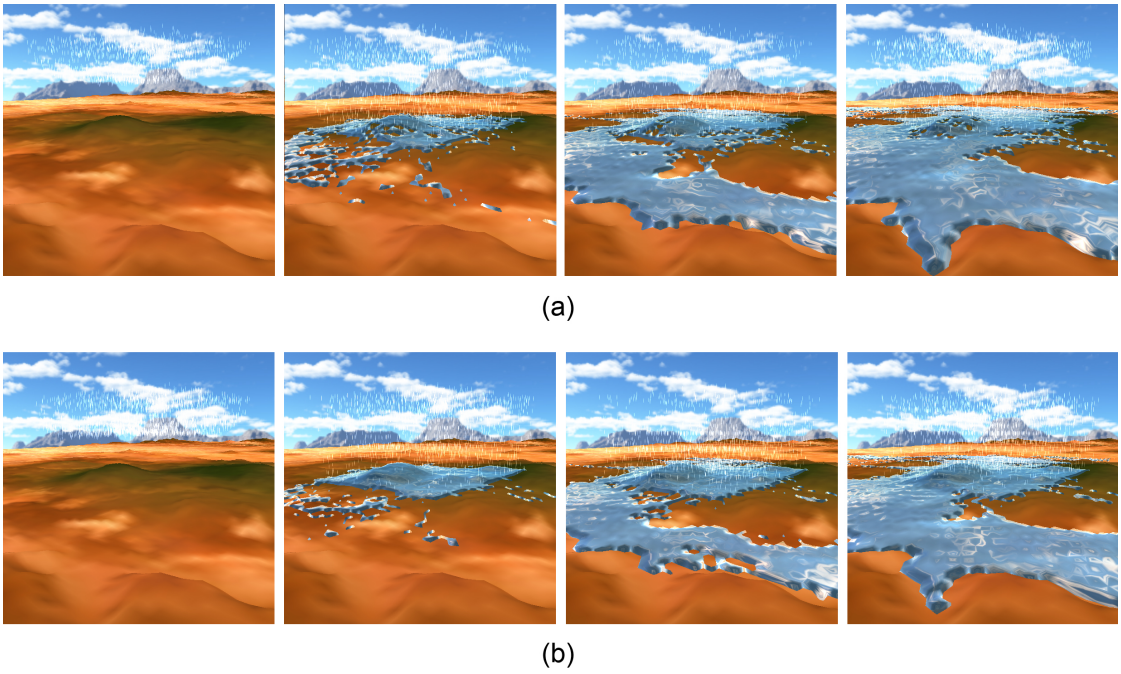


Figura 7.2: Exemplo de precipitação de chuva sobre terreno: (a) FHP; (b) D2Q9. Ambos nos passos de tempo $t = 0$, $t = 150$, $t = 800$ e $t = 3000$.

O exemplo ilustrado na Na Figura 7.2 utilizou uma topografia simples. Veremos agora experimentos realizados com modelos digitais de terreno com topografias mais complexas. A Figura 7.3, por exemplo, mostra a evolução do fluido em uma região do terreno que apresenta elevações e vales. Como esperado, o fluido acompanha a topografia do terreno, sempre escolhendo as trajetórias mais acessíveis de acordo com o declive. O exemplo na Figura 7.3(a) foi simulado através do modelo FHP, enquanto na Figura 7.3(b) com o modelo D2Q9.

Uma análise visual da Figura 7.3 indica que o comportamento do fluido em ambos os casos é similar. Entretanto, se atentarmos para os detalhes, veremos que no caso do FHP há o surgimento de certos padrões (alguns buracos) na superfície livre presente no topo da montanha (região mais elevada no terreno), padrões esses que não aparecem no exemplo do D2Q9. Isto acontece devido às regras do FHP e à geometria de sua *lattice* que tende a criar fluxos principais de acordo com suas direções. A Figura 7.4 apresenta esses detalhes, onde podemos ver à esquerda os padrões formados pelo modelo FHP, o que não ocorre com o D2Q9 à direita.

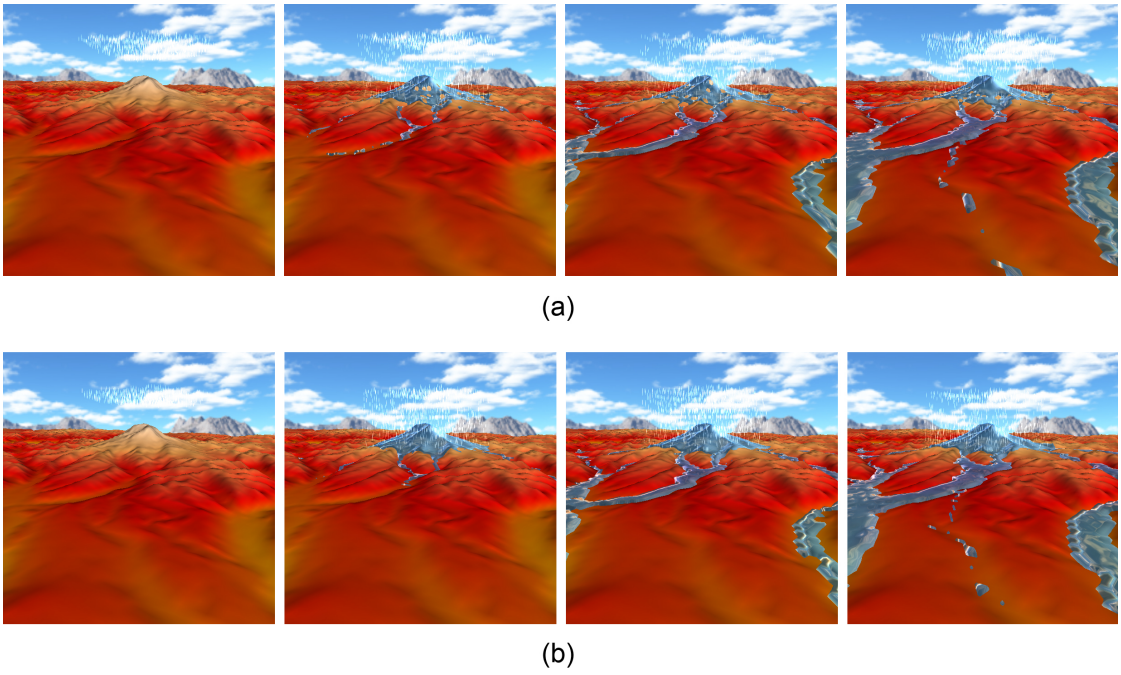


Figura 7.3: Exemplo de busca por caminhos: (a) FHP; (b) D2Q9. Ambos nos passos de tempo $t = 0$, $t = 150$, $t = 1200$ e $t = 3000$.

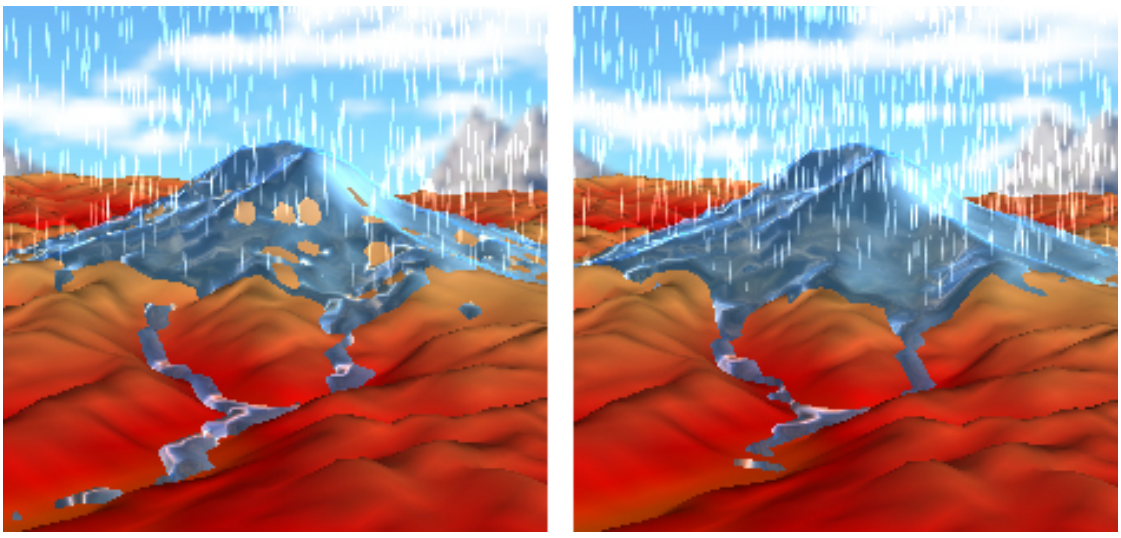


Figura 7.4: Destaque dos detalhes do surgimento de certos padrões (alguns buracos no topo da montanha) ao utilizar o modelo de simulação FHP (à esquerda); o que não ocorre com o modelo D2Q9 (à direita).

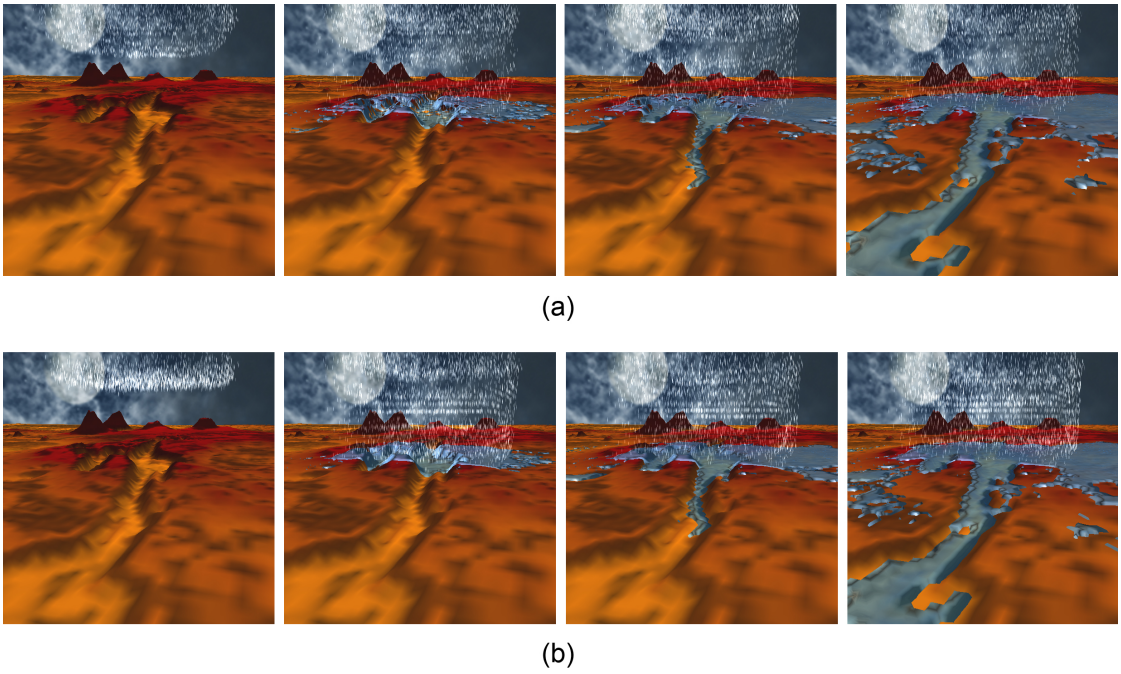


Figura 7.5: Exemplo de formação de lagos: (a) FHP; (b) D2Q9. Ambos nos passos de tempo $t = 0$, $t = 150$, $t = 1100$ e $t = 3000$.

Em contrapartida, na Figura 7.5 a animação ocorre sobre um modelo de terreno que apresenta vales mais profundos em sua superfície. Devido a essa característica, a formação de lagos e rios se tornou mais evidente. Podemos notar, ao longo da evolução temporal, o alargamento da área inundada.

Novamente é perceptível uma similaridade no comportamento dos fluidos em ambos os casos. Conseqüentemente, apesar do FHP ser um modelo mais simplificado do ponto de vista teórico, este resultado nos mostra que ele é capaz de reproduzir efeitos fisicamente plausíveis.

Para o caso das aplicações em tempo real este fato é de grande importância, pois possibilita alcançar resultados visuais tão bons quanto modelos mais complexos, sem prejudicar a eficiência computacional. Na seção 7.1.2 veremos com mais detalhes as análise da eficiência computacional dos exemplos gerados.

Para finalizar, a Figura 7.6 mostra um modelo de terreno que apresenta uma *bacia* em sua superfície. Ao posicionarmos o domínio $3D$ da chuva sobre essa região, notamos o acúmulo de água e o comportamento do fluido ao transbordar.

Assim como os demais exemplos mostrados nessa seção, na Figura 7.6(a) utilizou-se o modelo FHP para a simulação da dinâmica da chuva, enquanto na Figura 7.6(b) o modelo utilizado foi o D2Q9.

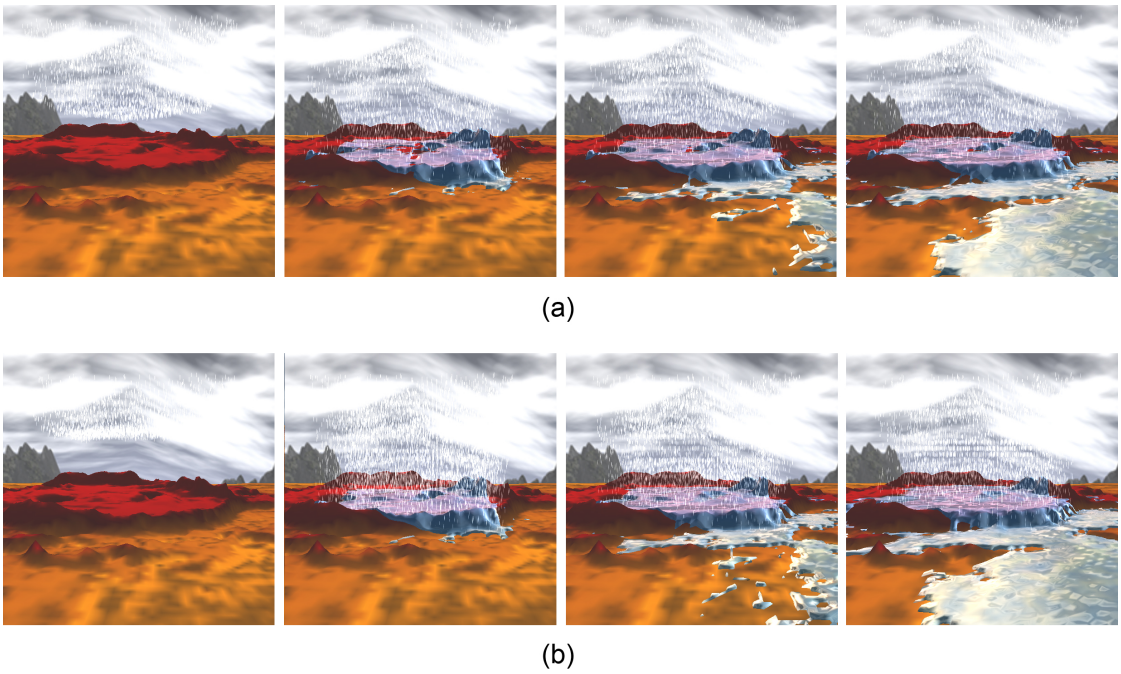


Figura 7.6: Exemplo de acúmulo de água: (a) FHP; (b) D2Q9. Ambos nos passos de tempo $t = 0$, $t = 150$, $t = 1200$ e $t = 3000$.

Para renderizar a superfície livre usamos a técnica de *Fresnel*, descrita na seção 5.5.2. A equação responsável por calcular a cor em cada pixel da superfície livre (expressão (5.11)) depende, dentre outros parâmetros, da cor da textura do *Cube Mapping*, ou seja, da textura que representa os diversos cenários utilizados. Desta forma, cenários diferentes geram uma visualização da superfície livre diferente, como podemos notar ao comparar os exemplos desta seção. As Figuras 7.2 e 7.3, que utilizaram uma textura de um dia ensolarado, apresentam uma superfície livre mais azulada, em contrapartida a Figura 7.5, com uma cena noturna, mostra uma superfície livre mais escura e a Figura 7.6 uma superfície esbranquiçada devido ao cenário de um dia nublado.

7.1.2 Eficiência Computacional

Para analisarmos a eficiência computacional, geramos diversos exemplos com resoluções de malha $3D$ distintas. As condições iniciais de todos os exemplos são as mesmas utilizadas nos exemplos da seção 7.1.1. Definimos também uma resolução de terreno fixa (256×256), sendo o modelo de terreno aquele utilizado no exemplo da Figura 7.3. Todos os exemplos evoluíram até 3000 (três mil) passos de tempo, onde medimos o FPS (*frames* por segundo) de cada animação. As mesmas resoluções de malha $3D$ foram usadas tanto no modelo FHP quanto no D2Q9. Em primeiro momento vamos analisar os resultados obtidos utilizando o modelo FHP.

Na Tabela 7.1 a coluna **Malha 3D FHP** lista as resoluções utilizadas, cujas medidas dos valores representam (altura $\times$ largura $\times$ profundidade) do domínio $3D$; a coluna **Partículas d'água** traz o número de partículas acumuladas no fluido sobre o terreno ao longo dos 3000 passos de tempo; a coluna **Partículas Macro** representa o número médio (por *frame*) de partículas macroscópicas presentes na simulação da chuva; e a última coluna lista a quantidade de FPS obtida.

A $11^{\underline{a}}$ linha da Tabela 7.1 representa exatamente a resolução de malha $3D$ usada para gerar os exemplos ilustrados na seção 7.1.1. Podemos notar que para essa dada resolução temos como resultado 32,61 FPS, que vai além da taxa de *frame* necessária para uma aplicação em tempo real (aplicações essas que necessitam de uma taxa igual ou superior a 30 *frames* por segundo).

Analisando os demais resultados de FPS, vemos que a maioria das configurações de malha $3D$ listadas na Tabela 7.1 retornou um valor superior a 30 FPS. As resoluções que retornaram FPS abaixo dos 30 são as resoluções de maior dimensão na base do domínio (largura $\times$ profundidade). O decrescimento do FPS não depende somente do número de nós da malha $3D$. Por isto, a resolução da $8^{\underline{a}}$ linha da Tabela 7.1, cuja malha $3D$ contém 32000 (trinta e dois mil) nós retorna um FPS menor (26,55) se comparado com a $15^{\underline{a}}$ linha (30,93 FPS), cuja malha $3D$ contém 36000 (trinta e seis mil) nós, ou seja, 4 mil nós a mais.

Tabela 7.1: Variação do FPS de acordo com a resolução da malha $3D$ usando o modelo de simulação FHP.

Malha 3D FHP	**Partículas d'água**	**Partículas Macro**	**FPS**
01) 10×10×10	795	180	93,75
02) 10×20×20	2778	496	75,00
03) 10×30×30	5644	1030	54,55
04) 10×40×40	7872	1751	38,46
05) 20×10×10	274	109	90,91
06) 20×20×20	1443	762	61,23
07) 20×30×30	5111	1537	40,54
08) 20×40×40	10518	2678	26,55
09) 30×10×10	182	126	85,71
10) 30×20×20	981	489	56,60
11) 30×30×30	2434	2147	32,61
12) 30×40×40	8332	3548	20,83
13) 40×10×10	111	139	81,08
14) 40×20×20	683	567	50,85
15) 40×30×30	1078	1314	30,93
16) 40×40×40	3142	4501	17,05

Podemos observar que o FPS obtido é dependente do número de nós da malha $3D$ bem como do número de partículas macroscópicas que compõe a chuva. Entretanto, uma vez que tais partículas são excluídas do sistema ao ultrapassarem os limites do domínio, é possível que duas resoluções de malha similares retornem um FPS diferente. Vamos analisar, por exemplo, os experimentos da $2^{\underline{a}}$ e $13^{\underline{a}}$ linhas da Tabela 7.1. Ambos os casos utilizaram a mesma quantidade de nós (4000), entretanto podemos notar que o FPS obtido no primeiro caso é menor que o FPS obtido no segundo.

O experimento da 13ª linha utiliza uma malha $3D$ com 40 nós de altura enquanto a malha da 2ª linha tem apenas 10 nós de altura. Desta forma, o número médio de partículas macroscópicas (por *frame*) no exemplo da 13ª linha é menor que o da 2ª linha (139×496), pois a probabilidade de uma macro partícula ultrapassar os limites do domínio $3D$ antes de atingir o terreno é grande. Conseqüentemente, a malha $3D$ mais alta contribui bem menos para a precipitação da chuva sobre o terreno (111 contra 2778 partículas acumuladas).

Podemos concluir então que o custo computacional envolvido no movimento de partículas macroscópicas bem como o custo de verificação de precipitação no terreno é maior em malhas $3D$ cuja dimensão de altura é menor do que as dimensões da base.

Analisaremos agora os resultados obtidos ao utilizar o modelo de simulação D2Q9. A Tabela 7.2 lista os resultados obtidos nas animações geradas com o modelo D2Q9 para a simulação da chuva. Para fins de comparação, mantivemos as mesmas configurações utilizadas para a Tabela 7.1.

Na Tabela 7.2 a coluna **Malha 3D D2Q9** lista as resoluções utilizadas, cujas medidas dos valores representam (altura $\times$ largura $\times$ profundidade) do domínio $3D$; a coluna **Partículas d'água** traz o número de partículas acumuladas no fluido sobre o terreno; a coluna **Partículas Macro** representa o número médio (por *frame*) de partículas macroscópicas presentes na simulação da chuva; e a última coluna lista a quantidade de FPS obtida.

Podemos observar que na 11ª linha da Tabela 7.2, cuja resolução da malha $3D$ foi a mesma usada para gerar os exemplos ilustrados na seção 7.1.1, obtivemos um resultado de 22,56 FPS. Assim, para a mesma resolução de malha, a utilização do modelo FHP para simular a dinâmica da chuva retornou um FPS maior do que o modelo D2Q9. Analisando os demais resultados vemos que o comportamento é análogo à Tabela 7.1, onde a maioria das configurações de malha $3D$ retornou valor superior a 30 FPS, e as resoluções que retornaram FPS inferior são as que apresentam maior dimensão na base do domínio (largura $\times$ profundidade).

Tabela 7.2: Variação do FPS de acordo com a resolução da malha 3D usando o modelo de simulação D2Q9.

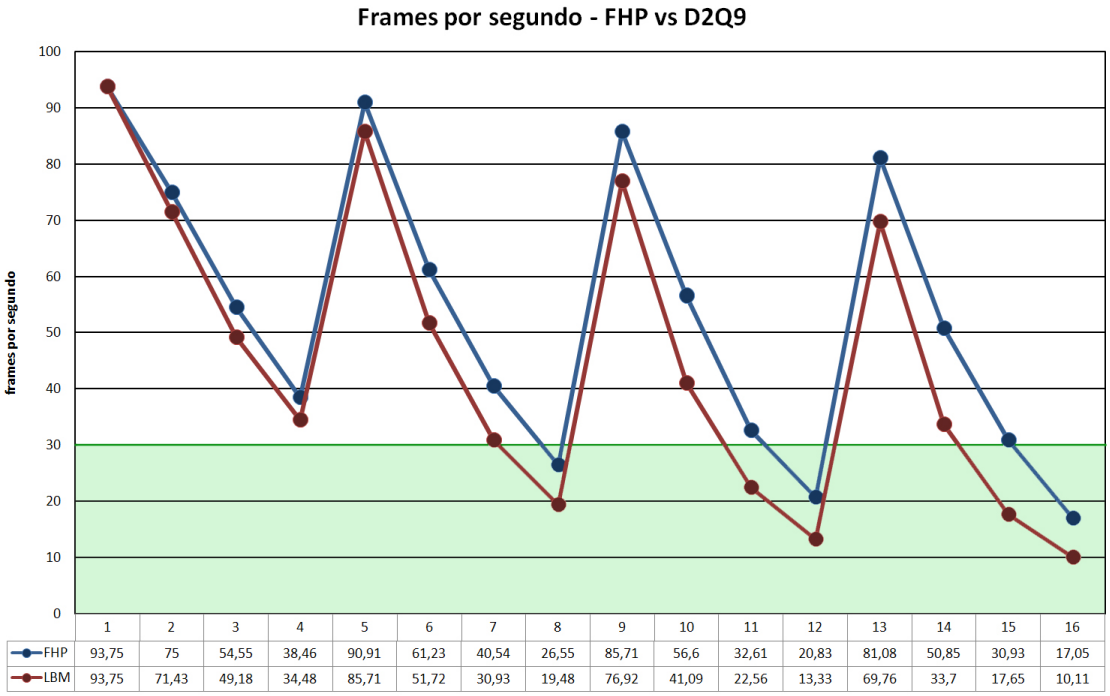
Malha 3D D2Q9	**Partículas d'água**	**Partículas Macro**	**FPS**
01) 10×10×10	922	106	93,75
02) 10×20×20	5884	373	71,43
03) 10×30×30	10628	801	49,18
04) 10×40×40	17864	1335	34,48
05) 20×10×10	977	216	85,71
06) 20×20×20	6252	739	51,72
07) 20×30×30	11562	1542	30,93
08) 20×40×40	19364	2640	19,48
09) 30×10×10	1032	306	76,92
10) 30×20×20	6267	1074	41,09
11) 30×30×30	11650	2309	22,56
12) 30×40×40	19652	3981	13,33
13) 40×10×10	1048	419	69,76
14) 40×20×20	6302	1435	33,70
15) 40×30×30	11770	3040	17,65
16) 40×40×40	19652	5326	10,11

Além disso, nos Métodos *Lattice Boltzmann* não há partículas microscópicas como no modelo FHP. Desta forma, a cada interação da simulação, a função (4.27) atualiza a distribuição f_i para cada direção i de movimento em cada nó da *lattice* do D2Q9. Isto explica porque duas malhas similares na Tabela 7.2 (2ª e 13ª linhas) obtiveram taxas de FPS similares (71,43 e 69,76 FPS).

Similar ao FHP, o FPS obtido com o modelo D2Q9 é dependente do número de nós da malha bem como do número de partículas macroscópicas que compõe a chuva. Entretanto, o custo computacional para Métodos *Lattice Boltzmann* é

maior, se comparado ao FHP, uma vez que no LBM a função de distribuição de equilíbrio (4.32), presente na regra de evolução do D2Q9, depende das quantidades macroscópicas de densidade e velocidade. Isto implica que para cada passo de interação é necessário calcular estas quantidades, aumentando o custo computacional do modelo. Conseqüentemente, para uma dada malha $3D$, a simulação através do modelo FHP tende a ser mais rápida comparada à simulação com o modelo D2Q9.

A Figura 7.7 apresenta um gráfico comparativo entre as resoluções das malhas $3D$ e suas respectivas taxas de FPS obtidas nas Tabelas 7.1 e 7.2. A região em verde ressalta as faixas que representam valores abaixo dos 30 FPS, ou seja, valores insatisfatórios para uma aplicação em tempo real.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
FHP	93,75	75	54,55	38,46	90,91	61,23	40,54	26,55	85,71	56,6	32,61	20,83	81,08	50,85	30,93	17,05
LBM	93,75	71,43	49,18	34,48	85,71	51,72	30,93	19,48	76,92	41,09	22,56	13,33	69,76	33,7	17,65	10,11

Figura 7.7: Gráfico comparativo das taxas de FPS: relação das malhas $3D$ com suas respectivas taxas de FPS obtidas. A região em verde ressalta as faixas que representam valores abaixo dos 30 FPS.

Naturalmente, à medida que aumentamos a resolução da malha $3D$ perdemos em eficiência computacional. Entretanto, não podemos esquecer que esses experimentos não sofreram nenhuma otimização em termos de paralelização dos códigos.

Tanto o FHP quanto o D2Q9 são modelos cujas regras de evolução, além de simples, ocorrem localmente, o que torna viável a paralelização do código. Desta forma, é possível alcançar maiores resoluções de malha $3D$, uma vez que a paralelização aumentará o desempenho computacional dos modelos. Falaremos mais sobre a paralelização na seção de trabalhos futuros.

O compilador GCC armazena em um arquivo informações sobre o consumo do tempo de processamento. Neste arquivo ele lista quais chamadas de funções do aplicativo consomem mais tempo, e retorna a porcentagem do tempo total consumido por cada uma dessas funções.

Com base nesse arquivo, a Figura 7.8 apresenta um gráfico comparativo das principais funções e sua porcentagem de consumo do tempo total. As medidas foram tiradas tanto com o uso do FHP quanto do D2Q9, em uma mesma condição inicial e de malha $3D$, e a evolução ocorreu até 3000 passos de tempo.

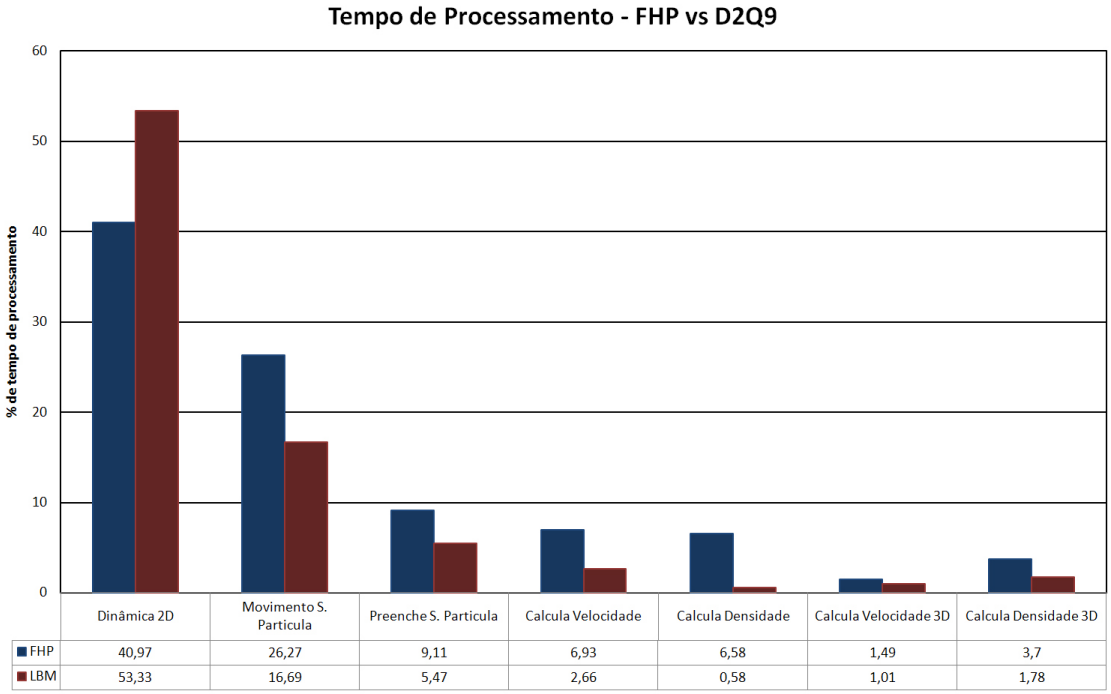


Figura 7.8: Gráfico comparativo do consumo do tempo de processamento para principais funções do aplicativo em tempo real.

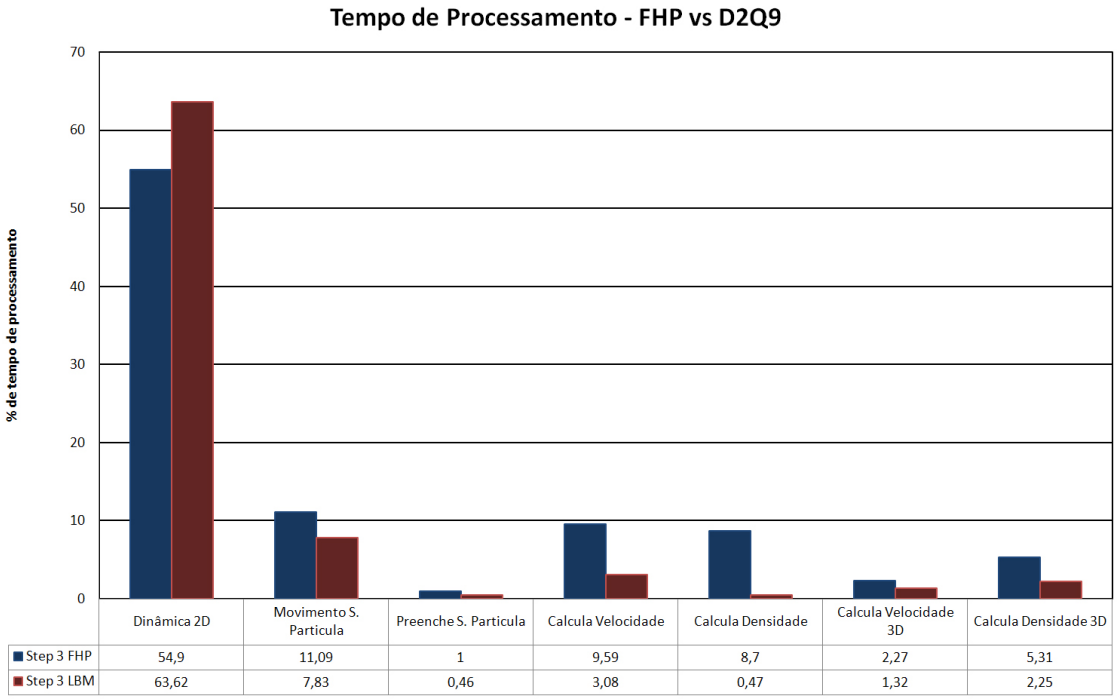
As funções em destaque na Figura 7.8 são:

- **Dinâmica 2D:** responsável pela evolução dos modelos FHP e D2Q9, dada pelos Algoritmos 1 e 2 respectivamente (vide seção 6.3.1);
- **Movimento S. Partícula:** responsável por atualizar os movimentos das partículas macroscópicas contidas no sistema de partículas, dada pela equação (6.4) (vide seção 6.2);
- **Preenche S. Partícula:** procedimento que se encarrega de realimentar o sistema de partículas com novas partículas, para manter o fluxo da chuva, como explicado na seção 6.4.3;
- **Calcula Velocidade:** responsável por calcular o campo de velocidade macroscópica bidimensional dos modelos FHP e D2Q9, dado pelas equações (4.24) e (4.31) respectivamente;
- **Calcula Densidade:** responsável por calcular o campo de densidade macroscópica bidimensional dos modelos FHP e D2Q9, dado pelas equações (4.23) e (4.30) respectivamente;
- **Calcula Velocidade 3D:** procedimento que calcula o campo de velocidade macroscópica tridimensional, dado pela equação (6.2);
- **Calcula Densidade 3D:** procedimento que calcula o campo de densidade macroscópica tridimensional, dado pela equação (6.1).

O resultado obtido para a função **Dinâmica 2D**, representado pelas duas primeiras barras na Figura 7.8, já era esperado, uma vez que o D2Q9 demanda mais custo computacional por trabalhar com dados numéricos ponto flutuante e por necessitar dos campos macroscópicos a cada interação de tempo.

Entretanto, o resultado obtido para a função **Movimento S. Partícula** mostrou que tal função demanda um consumo do tempo total considerável. Para reduzir o custo computacional, reduzimos a quantidade de chamadas dessa função, diminuindo assim o número de partículas macroscópicas presentes no sistema.

Desta forma, ao invés de chamar a função a cada instante de tempo, optamos por chamá-la a cada 3 instantes de tempo. A Figura 7.9 apresenta o gráfico comparativo das principais funções após a aplicação dessa otimização. Visualmente, não observamos diferenças significativas nas animações geradas.



	Dinâmica 2D	Movimento S. Particula	Preenche S. Particula	Calcula Velocidade	Calcula Densidade	Calcula Velocidade 3D	Calcula Densidade 3D
Step 3 FHP	54,9	11,09	1	9,59	8,7	2,27	5,31
Step 3 LBM	63,62	7,83	0,46	3,08	0,47	1,32	2,25

Figura 7.9: Gráfico comparativo do consumo do tempo de processamento para principais funções do aplicativo em tempo real, após otimização.

A Figura 7.10 mostra um comparativo das taxas de FPS obtidas após aplicação desta otimização. As simulações feitas com o modelo FHP estão representadas pelas linhas verde (sem otimização) e azul (com otimização), e as simulações feitas com o modelo D2Q9 estão representadas pelas linhas amarela (sem otimização) e vermelha (com otimização). Comparando com a Figura 7.7 podemos observar que há uma pequena melhora na eficiência computacional.

A função que mais consome tempo de processamento é a função **Dinâmica 2D**, responsável pela evolução dos modelos de simulação. Um caminho a seguir, em termos de otimização, é a paralelização dessa função, como veremos na seção de Trabalhos Futuros (seção 8.1).

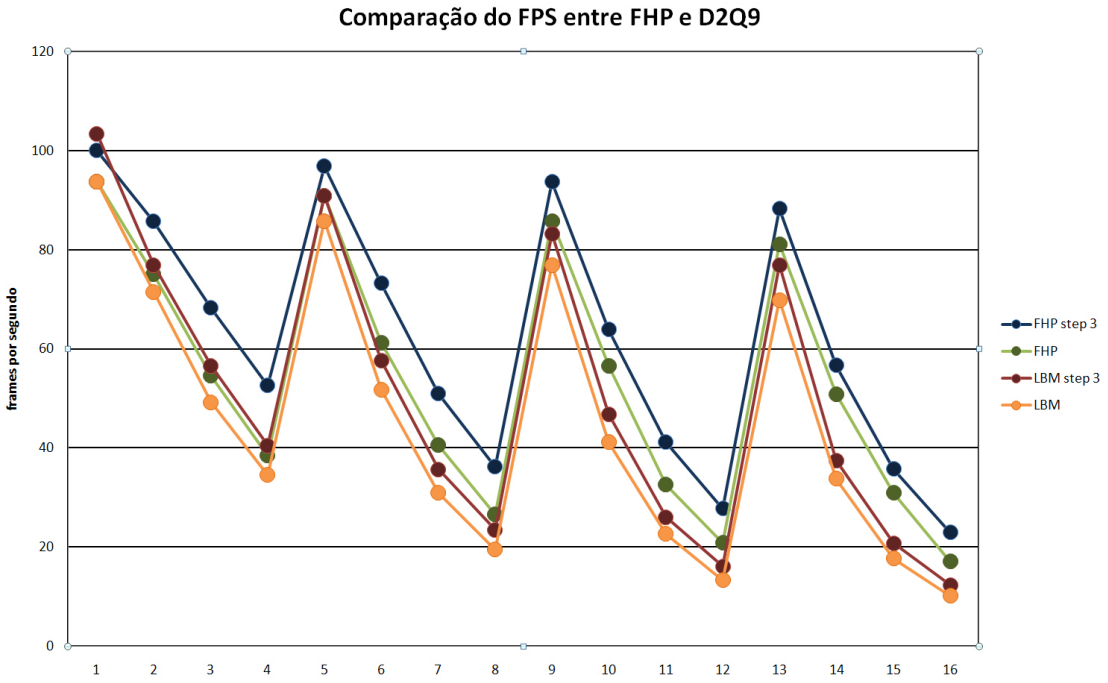


Figura 7.10: Gráfico comparativo das taxas de FPS após otimização.

7.2 Experimentos Computacionais da Aplicação *Off-Line*

A aplicação desenvolvida para execução *off-line* é composta pelo modelo de animação de fluido $3D$ explicado na seção 6.1 e pela técnica de *rendering* volumétrico *Volume Scattering* descrita na seção 5.6.1. Como explicado na seção 6.3, o aplicativo foi desenvolvido utilizando a linguagem de programação C/C++ em conjunto com a biblioteca GLUI para a interface gráfica.

Descreveremos a seguir os experimentos realizados com a aplicação *off-line*, com o intuito de mostrar o comportamento de um volume de gás. Na seção 6.4.3 explicamos as condições iniciais utilizadas na aplicação *off-line*, onde delimitamos um volume cilíndrico na base do domínio $3D$.

Para o modelo D2Q9, os valores iniciais de densidade dos nós pertencentes a essa região circular foram gerados aleatoriamente dentro de um intervalo previamente estabelecido (ρ_{min} - ρ_{max}). Para os resultados dessa seção utilizamos $\rho_{min} = 1.0$ e $\rho_{max} = 10.0$. Nos demais nós utilizamos o valor de densidade $\rho = 1.0$. A velocidade inicial de todos os nós do modelo D2Q9 assumiu valor nulo.

Os valores de densidade e velocidade são substituídos na equação (4.32) e os valores resultantes são as distribuições iniciais para cada direção de movimento em cada nó da *lattice*. À medida que o sistema evolui re-alimentamos as densidades dos nós pertencentes ao volume cilíndrico com valores aleatórios no intervalo $[1, 10]$.

Utilizamos também um sistema de partículas macroscópicas para enriquecer a visualização do comportamento do fluido. Para tanto, utilizamos um raio de influência $r = 5$ na expressão (6.3). A constante σ na expressão (6.4) recebe valor $\sigma = 0.0015$ para a simulação com o modelo FHP e $\sigma = 0.125$ para o modelo D2Q9.

A resolução da malha $3D$ utilizada foi $(50 \times 50 \times 50)$ nós, tanto para o modelo FHP quanto para o D2Q9. As imagens a seguir foram geradas através da técnica de *rendering Volume Scattering* (seção 5.6.1) onde, na equação (5.13) utilizamos $L_{ve} = 0.5$, $\sigma_s = 0.5$ e $f = 1.0$. Na equação (5.15) utilizamos $\sigma_a = 0.2$, que somado ao σ_s nos dá $\sigma_t = 0.7$ utilizado na equação (5.16). Com estes parâmetros, a equação (5.12) de transporte pode ser computada.

7.2.1 Comportamento do Volume de Gás

A visualização do volume de gás se dá através do campo de densidade macroscópica tridimensional, dado pela equação 6.1. Para o caso do modelo de simulação D2Q9, o valor de densidade em cada nó da *lattice* tem que ser positivo diferente de zero. Positivo, pois não faz sentido físico um valor de densidade negativo; e diferente de zero porque o cálculo de velocidade macroscópica, dado pela equação 4.31, é inversamente proporcional ao valor de densidade. Assim, uma densidade de valor zero implica em uma indeterminação no cálculo da velocidade.

Desta forma, uma vez que o campo de densidade macroscópica $3D$ tem valores positivos diferentes de zero em todos os nós, o resultado visual é uma frente de gás que se desloca pelo domínio $3D$, como mostra a Figura 7.11. O resultado fica visualmente pouco interessante. Para melhorar a aparência do volume de gás ilustrada na Figura 7.11, definimos uma região cilíndrica na base do domínio (seção 6.4.3) por onde o volume de gás será alimentado à medida que o sistema evolui.

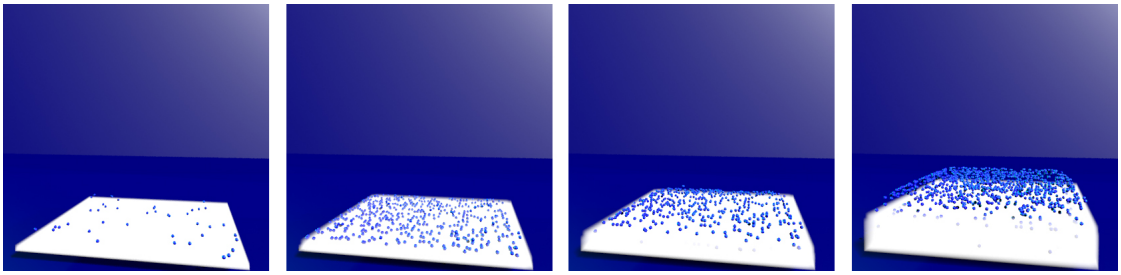


Figura 7.11: Volume de Gás preenchendo todo domínio $3D$.

Além disso, foi criada uma matriz auxiliar para armazenar as informações de densidade que serão passadas para a técnica de *rendering*. Esta matriz auxiliar (M') tem mesma dimensão que a matriz de densidade (M), e seus valores são dados como se segue:

$$M'(i,j,k) = \begin{cases} M(i,j,k), & \text{se } M(i,j,k) > \alpha; \\ 0, & \text{caso contrário,} \end{cases} \tag{7.1}$$

onde α é um limiar, utilizado para criar o efeito visual de uma frente de gás que se desloca no espaço. Utilizamos $\alpha = 2.5$ para os resultados dessa seção.

Para o modelo de simulação FHP não foi preciso aplicar essa filtragem de densidade, uma vez que sua densidade macroscópica é definida em função dos números de ocupação n_i que determinam se há ou não partícula na direção i de movimento. Desta forma, é possível que um determinado nó da malha apresente densidade nula. A Figura 7.12 mostra a evolução do volume de gás usando o modelo FHP para simulação.

Podemos perceber que, por mais que delimitemos a pequena área circular por onde o gás é alimentado, o modelo de simulação FHP tende a rapidamente preencher o domínio tridimensional devido às características de sua regra de evolução. Além disso, é perceptível na Figura 7.12 a formação de padrões na evolução do gás. Esses padrões acontecem devido à geometria da *lattice* hexagonal do FHP, que tende a criar fluxos principais de acordo com suas direções.

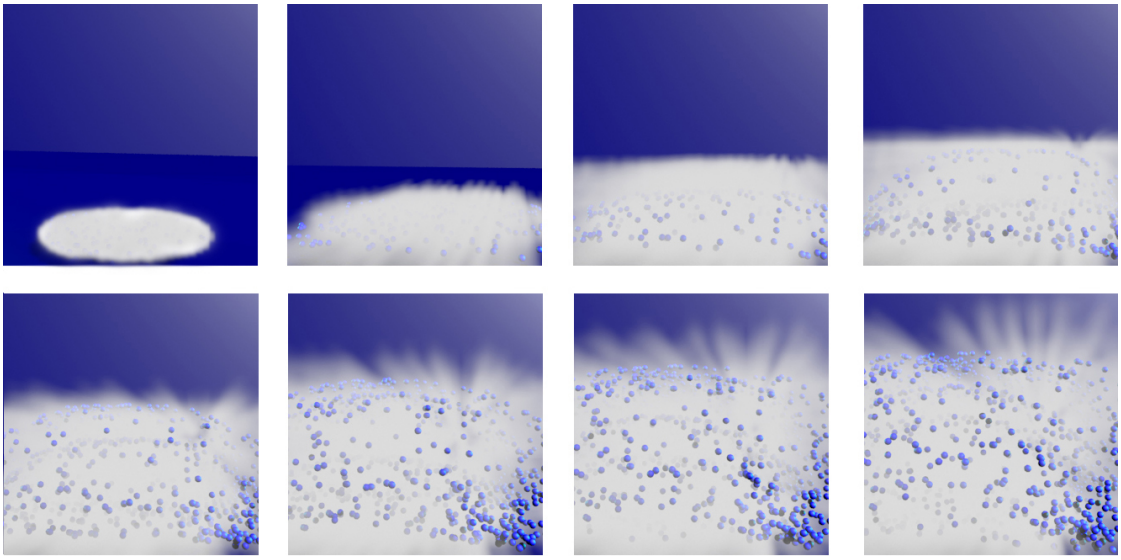


Figura 7.12: Evolução do volume de gás através do modelo FHP.

A Figura 7.13 mostra a evolução do volume de gás através do modelo D2Q9 para simulação. Como dito na seção 7.2, para gerar o fluxo do gás nós re-alimentamos valores de densidade nos nós pertencentes à região cilíndrica na base. Desta forma, a partir de uma região mais densa de gás, a função (4.32) atualiza a função de distribuição nos passos seguintes de evolução, resultando na expansão do gás em direção às regiões menos densa.

Dependendo da escolha do limiar α na função 7.1, o resultado da animação através do modelo D2Q9 se torna mais realista comparado aos resultados com o FHP, pois não há a formação de padrões na expansão do gás.

A Figura 7.14 mostra outro exemplo, onde temos a evolução do volume de gás simulada através do modelo FHP, em torno de uma esfera de vidro. Podemos ver a interação do gás com o objeto sólido, bem como efeitos de reflexão das partículas gasosas na superfície da esfera. Tais efeitos foram possíveis de se obter devido ao potencial da técnica de *rendering volume scattering* utilizada.

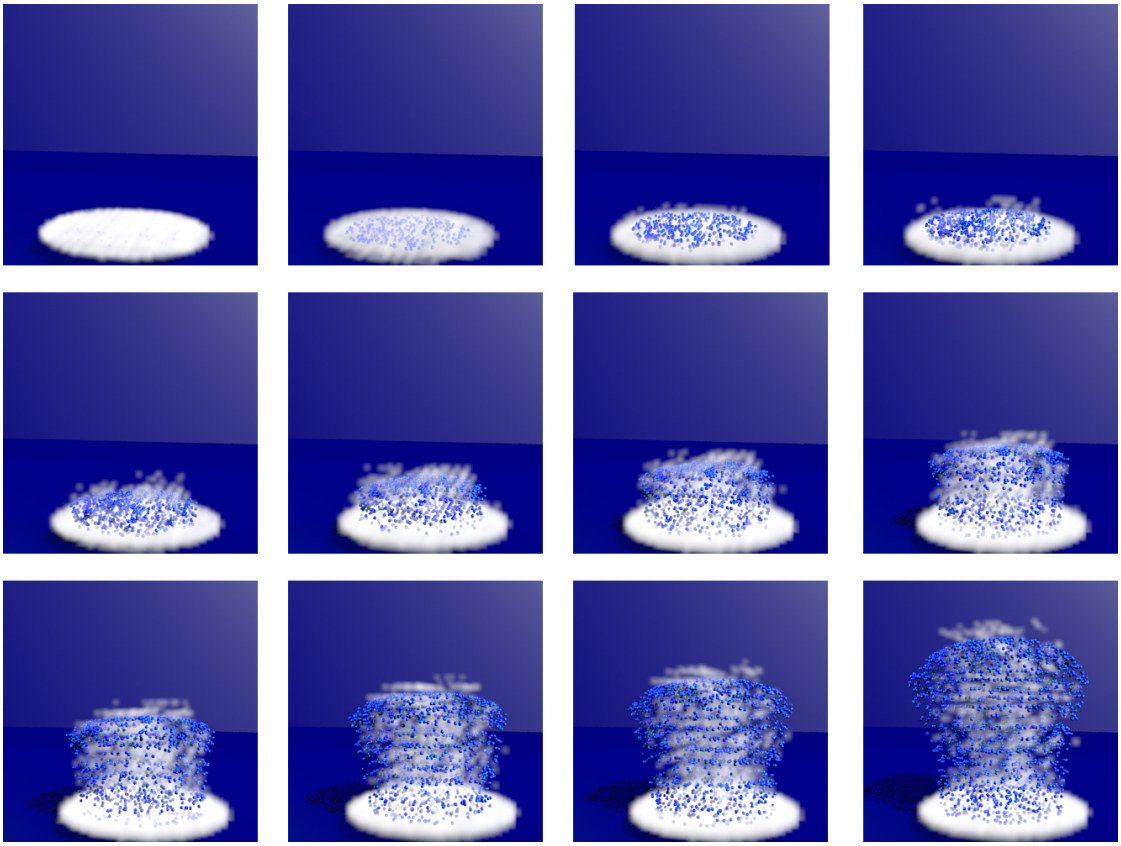


Figura 7.13: Evolução do volume de gás através do modelo D2Q9.

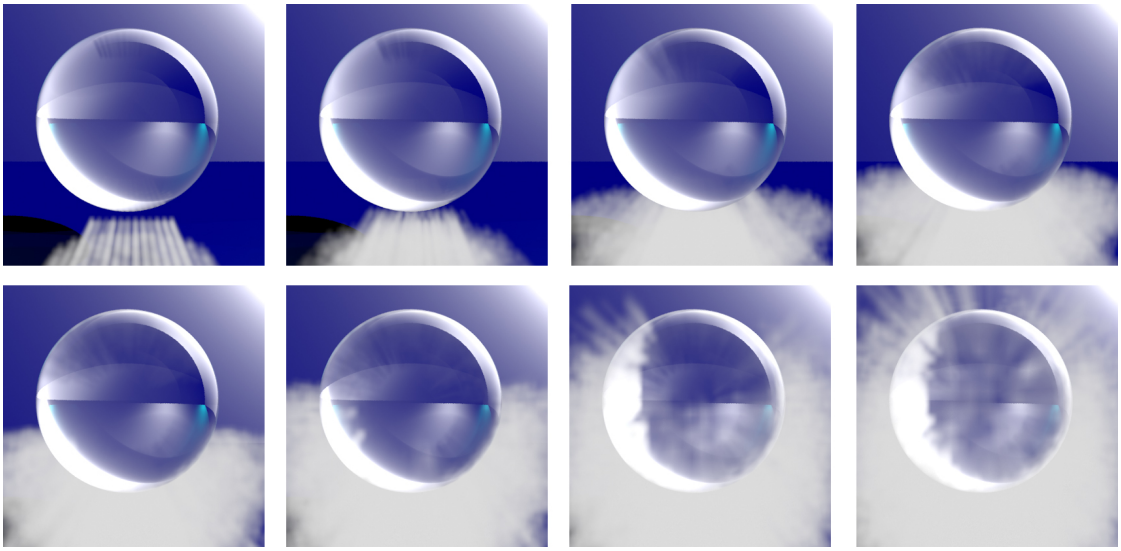


Figura 7.14: Evolução do volume de gás em torno de uma esfera.

Capítulo 8

Conclusão

O presente trabalho utilizou o método FHP, um Autômato Celular do tipo *Lattice Gas*, e o D2Q9, um Método *Lattice Boltzmann*, para o uso em animação de fluidos em computação gráfica.

A motivação para o nosso trabalho está no crescente interesse por técnicas de simulação de fluidos para a geração de animações realistas. As aplicações destas técnicas fazem parte da chamada Animação Computacional de Fluidos, uma área multidisciplinar que envolve métodos para simulação de fluidos, bem como técnicas em computação gráfica e visualização científica.

A metodologia conhecida como *top-down*, tradicional em animação computacional de fluidos, faz uso de métodos numéricos para resolver as equações que descrevem a dinâmica do fluido, na tentativa de capturar o comportamento global do sistema. Contudo, este trabalho faz uso de metodologias *bottom-up*, as quais buscam obter as relações globais entre variáveis macroscópicas através da descrição de interações locais.

É importante ressaltar que o uso de técnicas menos precisas do ponto de vista físico, mas que geram resultados visuais semelhantes àqueles gerados por modelos mais elaborados, é de grande importância em animação de fluidos para computação gráfica, uma vez que tais técnicas podem ser mais eficientes do ponto de vista computacional. Desta forma, é nesse contexto que propomos o uso de LGCAs e métodos *Lattice Boltzmann* para animação de fluidos.

Xavier et al (2005) propõe o uso do método FHP, um modelo bidimensional, para animação de fluidos em computação gráfica. Em continuidade, o presente trabalho propôs um modelo de fluido $3D$ baseado na metodologia FHP e em técnicas de interpolação. Optamos por trabalhar com o FHP por sua simplicidade, eficiência computacional e possibilidade de simular Navier-Stokes com um certo nível de precisão.

Além disso, o FHP utiliza apenas aritmética inteira para a evolução do fluido, não sofrendo portanto problemas envolvendo estabilidade numérica. Contudo, o método possui limitações. Alguns exemplos apresentados na seção 7.2.1 mostram que o FHP apresenta problemas em situações onde a malha hexagonal induz a comportamentos não realistas.

Desta forma, trabalhamos com o método de *Lattice Boltzmann*, o qual é mais eficiente que a dinâmica *booleana* dos autômatos e provê um modelo computacional mais apropriado para simular fluidos com número de Reynolds alto. Apesar do LBM prover modelos tridimensionais, optamos por trabalhar com o modelo bidimensional D2Q9 com o intuito de comparar com o FHP. Para tanto, desenvolvemos um modelo $3D$ análogo àquele baseado no FHP, mas agora utilizando o D2Q9.

O presente trabalho propôs duas aplicações para geração de animação de fluidos através das técnicas de fluidos $3D$ proposta, a saber: a aplicação em tempo real, descrita na seção 6.4; e a aplicação *off-line*, descrita na seção 6.5.

O objetivo da aplicação em tempo real é a animação de chuva sobre modelos digitais de terreno. Para tanto, foi proposto um modelo de interação entre a técnica de fluido $3D$ proposta com o método de animação de fluido sobre modelos digitais de terreno proposto por Barcellos et al (2007).

Na seção 7.1.1 mostramos os resultados visuais obtidos tanto com o modelo de simulação FHP quanto com o D2Q9. Vimos que apesar do FHP ser um modelo mais simplificado do ponto de vista teórico, é capaz de reproduzir efeitos fisicamente plausíveis. Para a computação gráfica, em especial para aplicações em tempo real, este resultado é importante, uma vez que podemos obter animações visualmente

realistas através de um método mais simplificado, implicando em maior eficiência computacional. Os resultados obtidos com o modelo baseado no FHP foram, em geral, computacionalmente mais eficientes, atingindo taxas de FPS apropriadas para aplicações em tempo real.

O objetivo da aplicação *off-line* é a animação de um volume de gás e visualização do resultado através da técnica de *rendering* volumétrico denominada *Volume Scattering* (seção 5.6.1). A técnica utilizada está disponível na biblioteca PBRT (Pharr e Humphreys, 2004), um sistema de *rendering* baseado em física. Desta forma, buscamos obter resultados mais realistas, utilizando o esquema de simulação proposto, para *rendering off-line*. Neste caso, o método baseado no D2Q9 mostrou resultados visualmente mais realistas, indicando que para estas aplicações este modelo pode ser mais adequado.

Os aplicativos desenvolvidos, descritos nas seções 6.4 e 6.5, tornaram a geração das animações mais eficiente, uma vez que os parâmetros intrínsecos às simulações puderam ser definidos em tempo de execução, agilizando todo o processo.

8.1 Trabalhos Futuros

Na seção 7.1.2 discutimos sobre a eficiência computacional da técnica de fluido $3D$ proposta na aplicação em tempo real. Apesar de termos obtidos valores de FPS satisfatórios, vimos que quanto maior a resolução da malha $3D$, maior é o custo computacional. Desta forma, um trabalho futuro para ganhar eficiência é aproveitar o paralelismo natural intrínseco aos modelos FHP e D2Q9 e implementar suas regras de evolução direto em *hardware* gráfico (GPU), o qual pode proporcionar melhor desempenho computacional.

Pretendemos também explorar as técnicas de *rendering* disponíveis na biblioteca PBRT (Pharr e Humphreys, 2004) em busca de resultados visuais ainda mais realistas que os apresentados na seção 7.2.1. Além disto, pretendemos explorar a interação líquido-gás e a interação entre fluidos e superfícies deformáveis, seguindo a linha descrita por Judice e Giraldi (2008b).

Por exemplo, seguindo a linha desenvolvida por Xavier (2006), podemos simular o líquido usando o método de *Smoothed Particle Hydrodynamics* (SPH) (Morris, 2000; Liu e Liu, 2003; Neto et al, 2005) e o gás utilizando o modelo $3D$ proposto nesta tese.

No caso de aplicações *off-line*, podemos explorar o método de *rendering* denominado *Photon Map* (Jensen e Christensen, 1998), também disponível no PBRT, e comparar os resultados com o *volume scattering*.

Além disso, será implementado o modelo D3Q15, um método do tipo *Lattice Boltzmann* tridimensional, para comparação com o modelo $3D$ proposto, tanto com relação ao custo computacional quanto realismo físico.

Referências Bibliográficas

Adabala N, Manohar S (2002) Techniques for realistic visualization of fluids: A survey. Comput Graph Forum 21(1):65–81

Appel A (1968) Some techniques for shading machine renderings of solids. Em: Proceedings of the Spring Joint Computer Conference, ACM, New York, NY, USA, págs 37–45, DOI http://doi.acm.org/10.1145/1468075.1468082

Azevedo E, Conci A (2003) Computação Gráfica: Teoria e Prática. Elsevier Editora

Baker S (2006) The Freeglut Project. Último acesso em 19/04/2009, URL http://freeglut.sourceforge.net/index.php

Barcellos B, Giraldi GA, Silva RL, Apolinario AL, Rodrigues PSS (2007) Surface flow animation in digital terrain models. Em: SVR 2007 - IX Symposium on Virtual an Augmented Reality, págs 123–132

Barcellos B, Giraldi GA, Apolinario AL, Rodrigues PSS (2008) Gpu surface flow simulation and multiresolution animation in digital terrain models. Relatório técnico, National Laboratory of Scientific Computing

Bhate N (1993) Application of rapid hierarchical radiosity to participating media. Em: Proceedings of Advanced Techniques in Animation, Rendering, and Visualization, Bilkent University, Ankara, Turkey, págs 43–53

Bhatnagar PL, Gross EP, Krook M (1954) A model for collision processes in gases. Physical Review 94(3):511–525

Blinn JF (1982) Light reflection functions for simulation of clouds and dusty surfaces. SIGGRAPH Comput Graph 16(3):21–29, DOI http://doi.acm.org/10.1145/965145.801255

Blinn JF, Newell ME (1976) Texture and reflection in computer generated images. Commun ACM 19(10):542–547, DOI http://doi.acm.org/10.1145/360349.360353

Boghosian B, Coveney P, Emerton A (1996) A lattice-gas model of microemulsions. Em: Proceedings of the Royal Society of London, págs 1221–1250

Buck I, Purcell T (2004) A toolkit for computation on gpus. Em: Pharr M (ed) GPU Gems, Addison-Wesley, págs 621–636

Burks AW (1970) Essays on Cellular Automata. University of Illinois Press, a collection of several papers on cellular automata.

Catmull EE (1974) A subdivision algorithm for computer display of curved surfaces. Tese de Doutorado, The University of Utah

Chang CL, Zhang YJ, Gdong YY (2004) Cellular automata for edge detection of images. Em: Third International Conference on Machine Learning and Cybernetics, Shanghai, págs 26–29

Chapra C, Canale RP (1988) Numerical Methods For Engineers. MacGraw-Hill International Editions

Chattopadhyay S, Adhikari S, Sengupta S, Pal M (2000) Highly regular, modular, and cascadable design of cellular automata-based pattern classifier. IEEE Transactions on Very Large Scale Integration Systems 8(6):724–735, DOI http://dx.doi.org/10.1109/92.902267

Chen S, Doolen GD (1998) Lattice boltzmann method for fluid flows. Annual Review of Fluid Mechanics 30:329–364

Chen S, Diemer K, Doolen GD, Eggert K, Fu C, Gutman S, Travis BJ (1991) Lattice gas automata for flow through porous media. Em: Proceedings of the NATO advanced research workshop on Lattice gas methods for PDEs : theory, applications and hardware, North-Holland Publishing Co., Amsterdam, The Netherlands, págs 72–84, DOI http://dx.doi.org/10.1016/0167-2789(91)90281-D

Chopard B, Droz M (1998) Cellular Automata Modeling of Physical Systems. Cambridge University Press

Chopard B, Luthi P, Masselot A (1998) Cellular automata and lattice boltzmann techniques: An approach to model and simulate complex systems. Em: Advances in Physics

Codd E (1968) Cellular Automata. Academic Press

Das AK, Sanyal A, Palchaudhuri P (1992) On characterization of cellular automata with matrix algebra. Information Sciences 61(3):251–277

Deusen O, Ebert DS, Fedkiw R, Musgrave FK, Prusinkiewicz P, Roble D, Stam J, Tessendorf J (2004) The elements of nature: interactive and realistic techniques. Em: SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes, ACM, New York, NY, USA, pág 32, DOI http://doi.acm.org/10.1145/1103900.1103932

Eberly D (2003) Derivative appoximation by finite differences. Magic Software, Inc

Enright D, Marschner S, Fedkiw R (2002) Animation and rendering of complex water surfaces. ACM Trans Graph 21(3):736–744, DOI http://doi.acm.org/10.1145/566654.566645

Fernando R, Kilgard MJ (2003) The CG tutorial: the definitive guide to programmable real-time graphics. Addison-Wesley

Ferwerda JA (2003) Three varieties of realism in computer graphics. Em: In Proceedings SPIE Human Vision and Electronic Imaging, págs 290–297

Filho JK, Almeida RC (2003) Introdução à modelagem matemática. Notas de aula do curso de Nivelamento da Pós-Graduação do LNCC

Frisch U, Hasslacher B, Pomeau Y (1986) Lattice-gas automata for the navier-stokes equation. Phys Rev págs 1505–1508

Frisch U, D'Humières D, Hasslacher B, Lallemand P, Pomeau Y, Rivet JP (1987) Lattice gas hydrodynamics in two and three dimension. Complex Systems 1:649–707

Gardner M (1970) The fantastic combinations of john conway's new solitaire game life. Scientific American 223:120–123

Giraldi GA (2005) Concepts in theory of computation and applications for automata-based modeling. Lecture Notes from National Laboratory for Scientific Computing - Brazil

Giraldi GA, JR ALA, Oliveira AAF, Feijóo RA (2005) Animação de fluidos via técnicas de visualização científica e mecânica computacional. Relatório técnico, Laboratório Nacional de Computação Científica, Petrópolis, RJ, Brasil

Goral CM, Torrance KE, Greenberg DP, Battaile B (1984) Modeling the interaction of light between diffuse surfaces. SIGGRAPH Comput Graph 18(3):213–222, DOI http://doi.acm.org/10.1145/964965.808601

Greene N (1986) Environment mapping and other applications of world projections. IEEE Comput Graph Appl 6(11):21–29, DOI http://dx.doi.org/10.1109/MCG.1986.276658

Guendelman E, Selle A, Losasso F, Fedkiw R (2005) Coupling water and smoke to thin deformable and rigid shells. Em: SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, ACM, New York, NY, USA, págs 973–981, DOI http://doi.acm.org/10.1145/1186822.1073299

He X, Luo LS (1997) A priori derivation of the lattice boltzmann equation. Phys Rev E 55(6):R6333–R6336, DOI 10.1103/PhysRevE.55.R6333

Higuera FJ, Jimenez J, Succi S (1989a) Boltzmann approach to lattice gas simulations. Europhys Lett 9

Higuera FJ, Succi S, Benzi R (1989b) Lattice gas dynamics with enhanced collisions. Europhys Lett 9:345–349, DOI 10.1209/0295-5075/9/4/008

Hirsch C (1988) Numerical Computation Of Internal And External Flows: Fundamentals Of Numerical Discretization, vol 1. John Wiley & Sons

Hughes TJR (1987) The Finite Element Method: Linear Static And Dynamic Finite Element Analysis. Prentice-Hall, Inc., New Jersey

Jensen HW, Christensen PH (1998) Efficient simulation of light transport in scences with participating media using photon maps. Em: SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, págs 311–320, DOI http://doi.acm.org/10.1145/280814.280925

Judice SF, Giraldi G (2008a) Animação de fluidos 3d via autômatos celulares do tipo fhp. Em: IV Congresso Sul Catarinense de Computação

Judice SF, Giraldi G (2008b) Discrete models for animating gas-liquid and fluid-surface interactions. Em: SIBGRAPI 08: Anais do WTDCGPI 2008 - Workshop de Teses e Dissertações em Computação Gráfica, Processamento de Imagens e Visão Computacional, págs 7–16

Judice SF, Coutinho BB, Giraldi G (2008) A cellular automata framework for real time fluid animation. Em: Proceedings of SBGames 08: Computing Track. Selecionado entre os 10 melhores da Área de Computação., págs 169–176

Judice SF, Coutinho BB, Giraldi G (2009) Lattice methods for fluid animation

in games. ACM Computers in Entertainment To appear in ACM Computers in Entertainment, Volume 7, Issue 4, 2009

Kajiya JT, Von Herzen BP (1984) Ray tracing volume densities. SIGGRAPH Comput Graph 18(3):165–174, DOI http://doi.acm.org/10.1145/964965.808594

Kari J (2005) Theory of cellular automata: a survey. Theor Comput Sci 334(1-3):3–33, DOI http://dx.doi.org/10.1016/j.tcs.2004.11.021

Kilgard MJ (1996) The OpenGL Utility Toolkit (GLUT): Programming Interface. Último acesso em 19/04/2009, URL http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf

Károlyi A, , Kertész J (1994) Hydrodynamics cellular automata for granular media. Em: Proceeding of the 6th Joint EPS-APS International Conference on Physics Computing, págs 675–681

Lafortune E, Lafortune EP, Lafortune EP, Willems YD, Willems YD (1996) Rendering participating media with bidirectional path tracing. Em: In Eurographics Rendering Workshop, Springer-Verlag/Wien, págs 91–100

Lengyel E (2004) Mathematics for 3D game programming and computer graphics, 2nd edição. Charles River Media, INC., Hingham, Massachusetts

Liu GR, Liu MB (2003) Smoothed particle hydrodynamics : a meshfree particle method. World Scientific, New Jersey

Martin O, Odlyzko A, Wolfram S (1984) Algebraic properties of cellular automata. Commun Math Phys 93:219–258

Max NL (1994) Efficient light propagation for multiple anisotropic volume scattering. Em: Fifth Eurographics Workshop on Rendering, Darmstadt, Germany, págs 87–104

McNamara GR, Zanetti G (1988) Use of the boltzmann equation to simulate lattice-gas automata. Phys Rev Lett 61(20):2332–2335, DOI 10.1103/PhysRevLett.61.2332

Monitzer A (2008) Fluid simulation on the gpu with complex obstacles using the lattice boltzmann method. Dissertação de Mestrado, Institute of Computer Graphics and Algorithms, Vienna University of Technology

Morris JP (2000) Simulating surface tension with smoothed particles hydrodynamics. International Journal for Numerical Methods in Fluids 33(3):333–353

Müller M, Charypar D, Gross M (2003) Particle-based fluid simulation for interactive applications. Em: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland

Müller M, Keiser R, Nealen A, Pauly M, Gross M, Alexa M (2004a) Point based animation of elastic, plastic and melting objects. Em: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, págs 141–151, DOI http://doi.acm.org/10.1145/1028523.1028542

Müller M, Schirm S, Teschner M (2004b) Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics. Technol Health Care 12(1):25–31

Müller M, Stam J, James D, Thürey N (2008) Real time physics: class notes. Em: SIGGRAPH 08: ACM SIGGRAPH 2008 classes, ACM, New York, NY, USA, págs 1–90, DOI http://doi.acm.org/10.1145/1401132.1401245

Nealen A, Müller M, Keiser R, Boxerman E, Carlson M (2006) Physically based deformable models in computer graphics. Comput Graph Forum 25(4):809–836

Neto AAS, Giraldi GA, Jr ALA, Rodrigues PS (2005) Animação computacional

de fluidos via smoothed particle hydrodynamics. Relatório técnico, LNCC - Laboratório Nacional de Computação Científica

von Neumann J (1966) Theory of Self-Reproducing Automata. University of Illinois Press, Urbana, Illinois

Nixon D, Lobb R (2002) A fluid-based soft-object model. IEEE Comput Graph Appl 22(4):68–75, DOI http://dx.doi.org/10.1109/MCG.2002.1016700

Owens J (2005) Streaming architectures and technology trends. Em: SIGGRAPH 05: ACM SIGGRAPH 2005 Courses, ACM Press, New York, NY, USA, pág 9, DOI http://doi.acm.org/10.1145/1198555.1198766

Pharr M, Humphreys G (2004) Physically Based Rendering: From Theory to Implementation. Elsevier, http://www.pbrt.org/

Phong BT (1975) Illumination for computer generated pictures. Commun ACM 18(6):311–317, DOI http://doi.acm.org/10.1145/360825.360839

Pozrikidis C (2001) Fluid Dynamics: theory, computation and numerical simulation. Kluwer Academic

Rademacher P (1999) GLUI: A GLUT-Based User Interface Library

Reeves WT (1983) Particle systems - a technique for modeling a class of fuzzy objects. ACM Transactions on Graphics 2:359–376

Rogers DF (1998) Procedural Elements for Computer Graphics, 2nd edição. Tom Casson

Rothman D, Zaleski S (1997) Lattice-Gas Cellular Automata: Simple Models of Complex Hydrodynamics. Cambridge University Press

Rothman DH, Zaleski S (1994) Lattice-gas models of phase separation: Interface, phase transition and multiphase flows. Rev Mod Phys 66:1417–1479

Rushmeier HE, Torrance KE (1987) The zonal method for calculating light intensities in the presence of a participating medium. SIGGRAPH Comput Graph 21(4):293–302, DOI http://doi.acm.org/10.1145/37402.37436

Sarkar P (2000) A brief history of cellular automata. Relatório técnico, Indian Statistical Institute

Schlick C (1994) An inexpensive brdf model for physically-based rendering. Computer Graphics Forum 13:233–246

Shaughnessy EJ, Katz IM, Schaffer JP (2005) Introduction to Fluid Mechanics. Oxford University Press

Shreiner D (2004) OpenGL Rerefence Manual: The Official Reference Document to OpenGL, 4th edição. Addison-Wesley Professional, http://www.opengl.org/

Sims K (1990) Particle animation and rendering using data parallel computation. Em: SIGGRAPH 90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, págs 405–413, DOI http://doi.acm.org/10.1145/97879.97923

Stam J (1999) Stable fluids. Em: SIGGRAPH 99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, págs 121–128, DOI http://doi.acm.org/10.1145/311535.311548

Stewart N (2006) GLUI User Interface Library. http://glui.sourceforge.net, Último acesso em 09/02/2009

Sucontphunt T, Mo Z, Neumann U, Deng Z (2008) Interactive 3d facial expression posing through 2d portrait manipulation. Em: GI '08: Proceedings of graphics interface 2008, Toronto, Ont., Canada, Canada, págs 177–184

Toffoli T, Margolus N (1987) Cellular Automata Machines: A New Environment for Modeling. MIT Press Series

Veach E, Guibas LJ (1997) Metropolis light transport. Em: SIGGRAPH 97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, págs 65–76, DOI http://doi.acm.org/10.1145/258734.258775

Wagner A (2008) A practical introduction to the lattice boltzmann method. Dept. of Physics, North Dakota State University

Wei X, Member S, Li W, Mueller K, Kaufman AE (2004) The lattice-boltzmann method for simulating gaseous phenomena. IEEE Transactions on Visualization and Computer Graphics 10:164–176

Weimar JR, Boon JP (1994) Class of cellular automata for reaction-diffusion systems. Physical Review E 49(2):1749–1752

Weimar JR, Watson LT, Tyson JJ (1991) Cellular automaton models for reaction diffusion equations. Em: Sixth Distributed Memory Computing Conference, IEEE Computer Soc., págs 431–434

Whitted T (1980) An improved illumination model for shaded display. Commun ACM 23(6):343–349, DOI http://doi.acm.org/10.1145/358876.358882

Witting P (1999) Computational fluid dynamics in a traditional animation enviroment. Em: SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, págs 129–136, DOI http://doi.acm.org/10.1145/311535.311549

Wolfram S (1994) Cellular Automata and Complexity: Collected Papers, 1st edição. AddisonWesley Pub. Co., http://www.stephenwolfram.com/publications/books/ca-reprint/

Wright RS, Lipchak B, Haemel N (2007) OpenGL SuperBible (4rd Edition). Addison-Wesley Professional

Xavier AV (2006) Animação de fluidos via autômatos celulares e sistemas de partículas. Dissertação de Mestrado, LNCC - Laboratório Nacional de Computação Científica

Xavier AV, Giraldi GA, Rodrigues PS, Jr ALA, Neto AAS (2005) Lattice gas cellular automata for computational fluid animation. Em: SIBGRAPI 2005: XVIII Brazilian Symposium On Computer Graphics And Image Processing, Natal, RN, Brasil

Zou Q, He X (1997) On pressure and velocity boundary conditions for the lattice boltzmann bgk model. Physics of Fluids 9(6):1591–1598

Apêndice A

Equação de *Boltzmann* e Modelo BGK

Nos anos 80, métodos de Autômatos Celulares do tipo *Lattice Gas* são propostos como uma técnica alternativa para simulação da dinâmica computacional dos fluidos (vide seção 3.4.3). Entretanto, tais métodos apresentam certas limitações, principalmente relacionadas à existência de ruídos devido a sua natureza discreta e ao surgimento de termos espúrios nas equações *Navier-Stokes* que limitam seu sucesso (Wagner, 2008).

Foi proposta então uma nova metodologia que substitui a variável de ocupação n_i (seção 4.2) por uma função de distribuição. Desta forma, eliminou-se o ruído do método e permitiu-se o uso de operadores de colisão mais gerais. Este é o Método de *Lattice Boltzmann*, uma técnica eficiente para resolver uma versão simplificada da Equação de *Boltzmann*, que em sua forma completa é dada por:

$$\partial_t f + \vec{v}\partial_{\vec{x}} f + \vec{F}\partial_{\vec{v}} f = \int dv_1' dv_2' dv_2 (f_1' f_2' - f_1 f_2) P_{12\to 1'2'}, \tag{A.1}$$

onde $\vec{v} = (v_1, v_2, v_3)$ é a velocidade microscópica, $\vec{x} = (x_1, x_2, x_3)$ indica posição no espaço, $\vec{F}$ é a distribuição de forças externas e f é distribuição de densidade de partículas. Wagner (2008) demonstra que a dinâmica da Equação de *Boltzmann* evolui no sentido de minimizar o Funcional H, dado por:

$$H(t) = \int f(\vec{x}, \vec{v}, t) \log(f(\vec{x}, \vec{v}, t)) dx_1 dx_2 dx_3 dv_1 dv_2 dv_3. \tag{A.2}$$

Desta forma, a função de distribuição de equilíbrio f^{eq}, para um sistema confinado em um volume V, com densidade de partícula n, com momento $n\vec{u}$ e energia $n\epsilon = \frac{1}{2}n\vec{u}^2 + \frac{3}{2}n\theta$, é aquela que minimiza o Funcional H. Podemos usar multiplicadores de Lagrange para minimizar este funcional:

$$\begin{aligned} H(t) = &\int f(\vec{x},\vec{v},t)\log(f(\vec{x},\vec{v},t))dx_1dx_2dx_3dv_1dv_2dv_3 \\ &- \lambda_1(nV - \int f(\vec{x},\vec{v},t)dx_1dx_2dx_3dv_1dv_2dv_3) \\ &- \lambda_{2\alpha}(nu_\alpha V - \int f(\vec{x},\vec{v},t)v_\alpha dx_1dx_2dx_3dv_1dv_2dv_3) \\ &- \lambda_3(n\epsilon V - \int f(\vec{x},\vec{v},t)\frac{\vec{v}^2}{2}dx_1dx_2dx_3dv_1dv_2dv_3). \end{aligned} \tag{A.3}$$

A solução de (A.3) fornece a expressão para distribuição de equilíbrio (Wagner, 2008):

$$f^{eq} = \frac{n}{(2\pi\theta)^{\frac{3}{2}}}\exp\left[-\frac{(\vec{v}-\vec{u})^2}{2\theta}\right]. \tag{A.4}$$

Bhatnagar, Gross and Krook notaram, entretanto, que o principal efeito do termo de colisão na Equação de *Boltzmann* é fazer com que a função de distribuição se aproxime da distribuição de equilíbrio ao longo do tempo. Assim, Bhatnagar et al (1954) propõe o modelo a seguir para o termo de colisão, baseado em um termo de relaxação τ:

$$\int dv_1'dv_2'dv_2(f_1'f_2' - f_1f_2)P_{12\to 1'2'} \approx \frac{1}{\tau}(f^{eq} - f). \tag{A.5}$$

Com esta aproximação, a Equação de *Boltzmann* se torna:

$$\partial_t f + \vec{v}\partial_{\vec{x}}f + \vec{F}\partial_{\vec{v}}f = \frac{1}{\tau}(f^{eq} - f). \tag{A.6}$$

Podemos usar essa expressão para aproximar a função de distribuição pela distribuição de equilíbrio e suas derivadas:

$$f = f^{eq} - \tau(\partial_t f + \vec{v}\partial_{\vec{x}}f + \vec{F}\partial_{\vec{v}}f). \tag{A.7}$$

Apêndice B

Implementação em GPU

GPU (*Graphics Processing Unit*) é uma unidade de processamento com propósito específico, baseada na arquitetura SIMD (*Single Instruction, Multiple Data*), uma arquitetura com unidades de processamento paralelas, onde vários conjuntos de dados podem ser processados simultaneamente, pelo mesmo conjunto de instrução ou *kernel*, executados em unidades distintas (Buck e Purcell, 2004). Outra característica das GPUs modernas é sua capacidade de processamento vetorial.

A GPU é projetada com o foco em aplicações gráficas, e sua arquitetura interna reflete a estrutura em pipeline que tais aplicações possuem. Nesse pipeline gráfico os vértices de um modelo $3D$ são processados em uma série de estágios e, ao final, um conjunto de *pixels* correspondente à imagem do modelo é gerado. Com a evolução, alguns desses estágios passam a ser programáveis, permitindo que um processo a ser executado em um dado estágio possa ser redefinido. Tais programas são denominados *shaders* (Owens, 2005). Atualmente GPUs permitem a criação de *shaders* para processamento de vértices, primitivas e fragmentos.

O método de animação de fluido sobre terrenos, descrito na seção 6.4.1, é baseado na metodologia de autômatos celulares, cujas regras de evolução são simples e ocorrem localmente, o que facilita uma implementação em GPU. A idéia básica consiste em codificar a informação necessária na simulação em forma de texturas, utilizando um mapeamento ponto-a-ponto entre os nós da *lattice* e os elementos de textura (*texels*).

Estas texturas serão atualizadas através dos *shaders* de fragmentos e então o resultado será enviado para um *array* de vértices. A Figura B.1 ilustra a implementação em GPU. A cada iteração uma textura bidimensional que codifica a configuração do autômato, chamada **Property**, é gerada. Para cada *texel* (i, j), cada canal de cor armazena uma informação diferente, a saber: a direção de movimento, a elevação do terreno ($\varphi(i, j)$) e um contador de partículas, para os canais R, G e B, respectivamente.

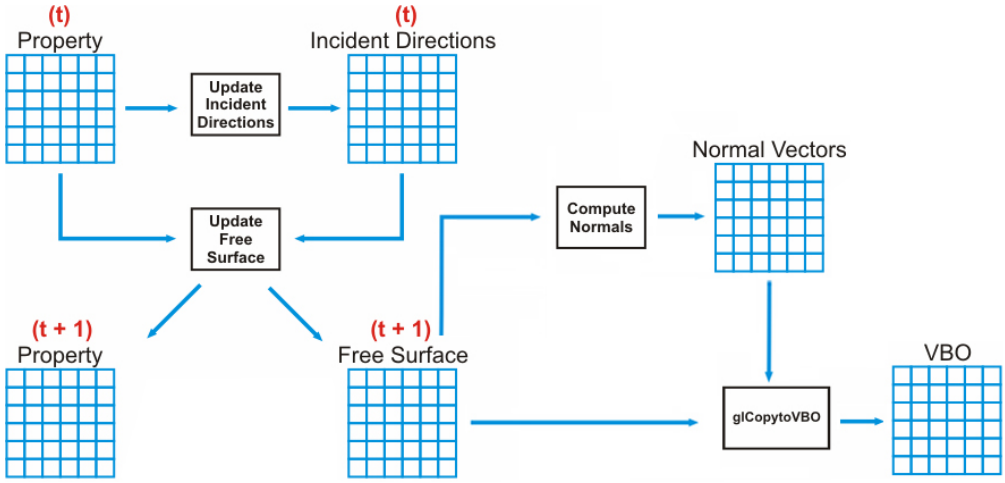


Figura B.1: Implementação em GPU para o método de animação de fluido sobre terrenos.

Com base nas informações contidas em Property o *shader* de fragmento **Update Incident Directions** gera uma textura RGBA, chamada **Incident Directions**. Nesta textura, para cada *texel*, cada canal R, G, B e A está associado ao vizinho V_k ($k \in \{1, 2, 3, 4\}$), respectivamente. Cada canal armazena uma variável *booleana*, cujo valor será 1 caso vizinho correspondente ao canal possuir uma partícula para enviar para (i, j), ou 0 caso contrário. Tal processo ocorre via comparação entre os valores de elevação da superfície livre de cada vizinho com o *texel* (i, j):

$$id_k(i,j) = \begin{cases} 1, & \text{se } \varphi_k(i,j) > \varphi(i,j),\ k \in \{1,2,3,4\}; \\ 0, & \text{caso contrário,} \end{cases} \tag{B.1}$$

onde id_k corresponde ao canal k do *texel* (i, j) da textura Incident Directions.

Com base nas informações contidas nas texturas Incident Directions e Property geradas no passo de tempo t, o *shader* denominado **Update Free Surface** atualiza, para cada *texel* (i, j), os contadores de partículas e calcula os pontos da superfície livre. O campo de contadores é armazenado na textura Property no passo de tempo $t + 1$, e o cálculo da superfície livre segue a expressão (6.5). Seus valores são armazenados em uma nova textura, chamada **Free Surface**.

A textura Free Surface é utilizada como dado de entrada para o *shader* de fragmento denominado **Compute Normals**, cuja função é calcular o vetor normal da superfície livre para cada *texel* (i, j) e armazenar o resultado na textura chamada **Normal Vectors**. O *shader* **glCopytoVBO** utiliza as informações das texturas Free Surface e Normal Vectors para gerar uma nova textura, chamada **VBO** (*Vertex Buffer Object*), que é um *array* de vértices utilizado para visualizar a superfície livre do fluido.

Desta forma, tanto a simulação quanto a visualização do método estão implementados em GPU, o que permite minimizar a troca de informação com a CPU, aumentando o desempenho da aplicação. Os *shaders* correspondentes estão detalhados no trabalho de Barcellos et al (2008).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo