COPPE/UFRJ

ANIMAÇÃO COMPUTACIONAL DA CHUVA COM FLUXO SUPERFICIAL SIMULADO VIA SPH

Bruno Barcellos de Souza Coutinho

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Antonio Alberto Fernandes de Oliveira

Rio de Janeiro
Março de 2011

ANIMAÇÃO COMPUTACIONAL DA CHUVA COM FLUXO SUPERFICIAL SIMULADO VIA SPH

Bruno Barcellos de Souza Coutinho

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

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

Prof. Claudio Esperança, Ph.D.

Prof. Gilson Antonio Giraldi, D.Sc.

Prof. Marcelo Walter, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2011

Barcellos de Souza Coutinho, Bruno

Animação Computacional da Chuva com Fluxo Superficial Simulado via SPH/Bruno Barcellos de Souza Coutinho. – Rio de Janeiro: UFRJ/COPPE, 2011.

XII, 71 p.: il.; 29, 7cm.

Orientador: Antonio Alberto Fernandes de Oliveira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 67 – 71.

1. animação da chuva. 2. SPH. 3. sistema de partículas. 4. PRT. 5. modelo de terreno. I. Alberto Fernandes de Oliveira, Antonio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A minha mãe, sem a sua ajuda
nada disso seria possível.
Obrigado por estar sempre ao
meu lado.

Agradecimentos

Gostaria de agradecer a todas as pessoas que contribuíram direta ou indiretamente para a realização desse trabalho. Gostaria de agradecer em especial ao meu orientador, professor Antonio Oliveira, por todo o suporte e árduas horas de dedicação ao trabalho. Ao professor Gilson Giraldi, que vem orientando meu trabalho desde a iniciação cientifica, e cuja contribuição foi crucial para a realização desse trabalho, além de meu amadurecimento acadêmico. Ressalto que os professores Antonio Oliveira e Gilson Giraldi foram muito mais do que orientadores, eles foram grandes amigos, sempre por perto em todos os momentos. A Sicilia Judice por sua contribuição nos diversos aspectos do trabalho: texto, discussões, vídeos, figuras, etc. A contribuição de Sicilia Judice se estendeu a áreas que vão bem além do contexto desse trabalho, por isso não posso deixar de agradecê-la por estar comigo em todos os momentos (alegres ou tristes, complicados ou simples) ao longo desses dois anos. Também sou muito grato aos professores Ricardo Marroquim e Claudio Esperança, além dos alunos Yalmar Ponce e Renan Galvão, por todas as dicas e discusões.

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

ANIMAÇÃO COMPUTACIONAL DA CHUVA COM FLUXO SUPERFICIAL SIMULADO VIA SPH

Bruno Barcellos de Souza Coutinho

Março/2011

Orientador: Antonio Alberto Fernandes de Oliveira

Programa: Engenharia de Sistemas e Computação

A animação da chuva é uma tarefa complexa onde são necessários esforços não só para tratar a dinâmica da água, mas também para se obter um *rendering* realístico. Efeitos secundários, como os respingos (*splashes*) das gotas, a formação de fluxo superficial e acúmulo são dificuldades conhecidas em computação gráfica. Nesse trabalho propomos um *framework* para a integração desses elementos em uma cena contendo um modelo digital de terreno (MDT). O aplicativo também oferece a possibilidade de inserção de chuva (sem fluxo superficial e respingos) em um vídeo real. A chuva é modelada através de um sistema de partículas, implementado em CUDA. Cada partícula representa uma gota e então a *Precomputed Radiance Transfer* (PRT), expressa em uma base de harmônicos esféricos, é utilizada para incorporar a luz do ambiente (*environment lightning*) na etapa de *rendering*. Para a animação de fluxo superficial e acúmulo da água na superfície do MDT utilizamos o *Smoothed Particle Hydrodynamics* (SPH). O *rendering* da superfície livre do fluido é realizado aplicando *environment mapping*, dosado pelo termo de Fresnel, sobre uma malha regular conhecida como *carpet*, utilizada para extrair a superfície livre do fluido. Os resultados experimentais ressaltam o potencial do *framework* proposto para aplicações de computação gráfica e destaca que o modelo também pode ser utilizado para aplicações de tempo real.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COMPUTATIONAL ANIMATION OF RAIN WITH SURFACE FLOW SIMULATION BY SPH

Bruno Barcellos de Souza Coutinho

March/2011

Advisor: Antonio Alberto Fernandes de Oliveira

Department: Systems Engineering and Computer Science

Realistic rain scenes animation is a complex task due to both the rendering and fluid dynamics simulation issues. The associated phenomena, like rain strokes rendering, splashing of raindrops and the simulation of the generated surface flow are known difficulties in this area. This paper describes a computational framework to incorporate these elements in a scene containing a digital terrain model (DTM). The proposed framework also allows the rain insertion (without surface flow and splashes) in a real video. In the rain model, the raindrops are modeled by a particle system implemented in GPU. Each particle represents a drop and the Precomputed Radiance Transfer (PRT), expressed in a spherical harmonics bases, is used to incorporate environment lightning to the rendering engine. The Smoothed Particle Hydrodynamics (SPH) method is employed for simulating the superficial flow over the terrain and lake formations. The rendering of the fluid free surface is performed by applying an environment mapping technique plus Fresnel effects to a regular geometric representation known as "carpet". The experimental results show the potential of the proposed pipeline for computer graphics and highlight the fact that it is a promising framework for real time applications.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A chuva é um fenômeno natural bastante complexo, onde para se conseguir uma animação realística uma série de efeitos precisam ser modelados. Técnicas para a animação e *rendering* da chuva, bem como efeitos relacionados a esse fenômeno, têm recebido a atenção da comunidade de computação gráfica nos últimos anos [1] [2] [3]. Muitos esforços têm sido feitos para se obter uma animação realística e de baixo custo computacional [4] [5]. A indústria cinematográfica e de jogos eletrônicos, além de aplicações de realidade virtual e simuladores, têm uma enorme carência por sistemas capazes de realizar esse tipo de animação.

Nesse trabalho propomos um *framework* capaz de realizar uma animação rápida e realística da chuva. Dentro desse contexto, propomos a junção de vários modelos descritos a seguir: (i) um modelo baseado em partículas utilizado para a animação da chuva; (ii) um modelo digital de terreno (MDT); (iii) o modelo SPH (*Smoothed Particle Hydrodynamics*) acoplado a um modelo responsável pela extração da superfície do fluido, para a animação do fluxo superficial produzido pela chuva; (iv) um modelo baseado em textura utilizado para a animação dos respingos das gotas sobre as superfícies.

O modelo para animação da chuva é inspirado no trabalho de [6]. Nesse sistema cada gota de chuva é representada por uma partícula. A atualização do sistema de partículas é realizada em paralelo pela GPU. A ideia é capturar detalhes das gotas da chuva de um vídeo real, e então inserir esses detalhes na renderização do sistema de partículas. Após a extração do vídeo, as imagens ainda passam por uma etapa de processamento, com o intuito de eliminar ruídos e obter assim um maior realismo. Nesse trabalho, as etapas de extração e processamento são substituídas por algumas imagens extraídas do banco de dados de gotas de chuva proposto no trabalho de [7]. Os autores desse trabalho produziram um banco de imagens de gotas sob diversas condições de iluminação e direções de visão, com ótimos

resultados visuais. Essas imagens são utilizadas como mapa de transparência (*alpha map*) para a cor computada, em tempo real, através da PRT (*Precomputed Radiance Transfer*). A PRT é utilizada para incorporar a iluminação ambiente (*environment lightning*) na renderização do sistema de partículas da chuva. A PRT é um método de iluminação global utilizado como alternativa aos custosos métodos baseados em *ray tracing* [8].

Além da animação da chuva, nosso *framework* também conta com a animação de efeitos secundários, como fluxo superficial e respingos nas superfícies. A superfície de um MDT (Modelo Digital de Terreno) é utilizada para a simulação do escoamento da água proveniente da chuva. Um modelo de terreno é definido pelas características físicas de uma região limitada da superfície terrestre. Sua topografia associa estas características em um conjunto de relações estruturadas. Um Modelo Digital de Terreno é uma representação do terreno e sua topografia, armazenada em um computador como uma estrutura de dados, e um conjunto de procedimentos associados [9]. Utilizamos um modelo tradicional representado por um mapa de alturas, denominado Modelo Digital de Elevação (*Digital Elevation Model* - DEM). Esse tipo de modelo pode ser obtido a partir de imagens aéreas, ou então gerado de forma procedural por algum algoritmo.

Ao passo que a água da chuva atinge o solo, é necessário dispor de um modelo apropriado para a simulação de fluxo superficial e acúmulo. Desde a sua introdução em computação gráfica por Desbrun [10], o SPH vem sendo empregado com sucesso na simulação de fluidos diversos. O SPH é um método livre de malha, baseado em partículas, utilizado para a solução numérica de equações diferenciais parciais e largamente utilizado em mecânica dos fluidos [11]. Devido à uma série de atrativos, os quais serão descritos mais adiante, esse foi o método escolhido para a simulação de fluidos.

O SPH utilizado nesse trabalho foi o proposto por Muller [12] para a simulação de líquidos, ou seja, fluidos com superfície livre e tensão superficial. Como o método é lagrangiano (não existe malha conectando as partículas), utilizamos o método proposto em [13] para a reconstrução e *rendering* da superfície livre do fluido. Esse método, chamado pelo autor de *carpet*, é um método alternativo aos encontrados na literatura, que visa a eficiência na animação de fluidos representados por funções de (x, y). Ele é composto basicamente de duas estruturas, a saber: uma malha regular, que cobre todo o domínio do terreno; e uma *quadtree* que possibilita a varredura eficiente dessa malha.

A animação dos respingos nas superfícies do fluido e do terreno é baseada na proposta [2]. Nesse modelo, a evolução temporal do respingo é representada por uma sequência de texturas, onde cada textura armazena a imagem de um instante da evolução.

Enfatizamos que a atualização do sistema de partículas da chuva é realizada em GPU (*Graphical Processor Unit*), as quais são projetadas com o foco em aplicações gráficas. As GPUs hoje são uma ótima opção de plataforma de processamento de alto desempenho com baixo custo. Com essa perspectiva, uma série de trabalhos vem sendo desenvolvidos no intuito de aproveitar esse potencial de processamento para aplicações não gráficas, o que se tem denominado GPGPU (*General-Purpose computation on GPUs*) [14]. O modelo de programação utilizado nesse tipo de arquitetura é denominado programação em fluxo ou *streaming programming*. Nesse modelo, diz-se que o programa executado pelas diversas unidades de processamento são núcleos (ou *kernels*) que atuam sobre o fluxo de dados (*data stream*). A distribuição dos dados pelas unidades de processamento é feita em uma ordem aleatória e, portanto, não há controle sobre a ordem na qual os dados se encontram dentro do fluxo.

Linguagens de programação, tais como o CUDA [15] da NVidia e o OpenCL [16] da ATI, surgiram com o objetivo de oferecer aos programadores uma camada de abstração sobre o *pipeline* gráfico. Nessa abstração, a GPU passa a ser vista como um conjunto de *streaming processors* idênticos e independentes de *pipeline* gráfico. Desta forma, o programador não precisa de conhecimentos sobre programação gráfica, se preocupando apenas com os aspectos da paralelização do algoritmo. A ideia é distribuir a execução do algoritmo entre um conjunto de *threads*, onde cada *thread* é executado simultaneamente pelos *streaming processors* presentes na GPU. Nesse trabalho utilizamos a linguagem CUDA para a implementação do sistema de partículas da chuva.

A principal contribuição deste trabalho é o desenvolvimento de um *framework* baseado em partículas para a animação da chuva e fenômenos relacionados, tais como fluxo superficial, formação de acúmulo e respingos nas superfícies. Esse tipo de sistema pode ser utilizado por aplicações em tempo real (jogos eletrônicos, aplicações em realidade virtual e simuladores), e ainda na produção de efeitos especiais. Na seção de resultados também apresentamos alguns testes onde o sistema foi utilizado, com sucesso, para realizar a inserção da chuva em um vídeo real.

O texto está organizado da seguinte forma. O Capítulo 2 traz uma revisão dos métodos utilizados em animação de chuva. O Capítulo 3 fala de técnicas para animação computacional de fluidos. O Capítulo 4 descreve o método de iluminação global denominado PRT (*Precomputed Radiance Transfer*). O Capítulo 5 apresenta os detalhes dos métodos que compõem o *framework* e a interação entre eles. O Capítulo 6 discute os resultados obtidos, finalizando com o Capítulo 7, que traz as conclusões e trabalhos futuros.

Capítulo 2

Animação da Chuva

Figura 2.1: Método interativo para a animação da chuva e diversos efeitos relacionados (Imagem retirada de [4]).

Ao longo da última década inúmeros métodos para a animação da chuva foram propostos em computação gráfica. Nesse período o problema foi investigado por pesquisadores das áreas de visão computacional, renderização fotorealística e aplicações em tempo real, como jogos eletrônicos e simuladores. Dessa maneira, as soluções encontradas na literatura são diversas, encontrando-se métodos baseados em imagens, baseados em textura, em sistemas de partículas ou ainda métodos híbridos.

2.1 Métodos Baseados em Imagem

A ideia básica desse tipo de método é reproduzir, em uma imagem ou vídeo, a geometria e o complexo padrão de brilho observado nas gotas. Esses padrões dependem de diversos fatores como, tipo de luz, posição do observador, refração/reflexão da luz, entre outros parâmetros de difícil controle durante a simulação [7]. Os resultados alcançados por esse tipo de método possuem elevado grau de realismo, obtido através da modelagem fisicamente correta do problema. No entanto, mesmo com o aumento de poder computacional, alavancado pelo uso de GPU, esse tipo de método ainda está um pouco distante das aplicações de tempo real.

O trabalho proposto por [1] foi pioneiro no processamento de imagens para a inserção/remoção da chuva. Após remover as gotas da chuva de um vídeo real, usando o filtro da mediana no tempo, as mesmas são inseridas em uma imagem qualquer, como ilustra a Figura 2.2. O método também possibilita a restauração do vídeo sem chuva. Em uma etapa de pós-processamento são inseridos efeitos de *blur* e a luminância da imagem do fundo é equalizada com a luminância das gotas.

Figura 2.2: Método pioneiro em realizar a inserção da chuva em imagens. (Imagem retirada de [1]).

O método proposto em [7], que inclui a geração de um banco de dados com gotas da chuva, é uma referência relevante na área, sendo citada em vários trabalhos posteriores. Nesse trabalho o autor realiza um estudo onde são medidos alguns parâmetros, sob diferentes condições de iluminação e pontos de vista, responsáveis pela aparência das gotas da chuva. Então esses parâmetros são utilizados pelo renderizador para a geração da banco de dados das gotas. Os autores também utilizam esse banco de dados para inserção de chuva em imagens ou vídeos com qualidade fotorealística, como ilustra a Figura 2.3.

Figura 2.3: Método capaz de realizar a inserção da chuva em imagens e vídeos. (Imagem retirada de [7]).

2.2 Métodos Baseados em Textura

Técnicas baseadas em textura utilizam uma textura que é continuamente "rolada" (*scrolling texture*) seguindo a direção da chuva [17]. Um dos principais atrativos dessa classe de métodos é a eficiência quando comparado com métodos baseados em imagem ou partículas. Vale ressaltar que para a geração de uma imagem final realística, sem a aparência planar, é interessante simular a chuva em diversas camadas de textura. Cada camada armazena uma determinada profundidade, dando a noção de um movimento *parallax*.

A animação da chuva apresentada em [4] é um bom exemplo desse tipo de método. Na Figura 2.4(a) observamos várias camadas de texturas, com profundidades e velocidades diferentes, codificadas em uma única textura. O sistema proposto realiza uma simulação de chuva sobre uma cidade durante a noite, como ilustra a Figura 2.4(b).

Apesar da animação da chuva em si ser baseada em textura, os autores apresentam diversos métodos baseados em partículas e GPU para a animação de efeitos secundários, tais como: chuva pesada (*strong rainfall*), gotas escorrendo sobre superfícies, respingos e goteiras, reflexões na superfície dos objetos e nas poças d'água, água jorrando de objetos, atenuação atmosférica da luz, poças d'água com ondas, dentre outros. Unidos a um sistema de iluminação com imagens no formato HDR (*High Dynamic Range*), o resultado final apresenta elevado grau de realismo, além de ótimas taxas de FPS (*frames per second* ou *frames* por segundo).

(a) (b)

Figura 2.4: (a): Mapa 8 bits de textura utilizado para animar a chuva. (b): Inserção do mapa na cena, unido a uma série de outros efeitos. (Imagem retirada de [4]).

As desvantagens do método são a impossibilidade de variação no ângulo da câmera, devido à chuva ser simulada via *scrolling texture*, a necessidade de se usar cerca de 500 *shaders* para as etapas de simulação e *rendering*, e uma placa gráfica ATI X1800 como requisito de *hardware*.

A metodologia proposta no trabalho de [18] permite variações no ângulo da câmera, porém ainda não permite a interação da chuva com o resto da cena. A metodologia ilustrada na Figura 2.5 é utilizada para a animação de chuva e neve. Basicamente o algoritmo mapeia quatro texturas em um cone duplo centrado na câmera. Assim, os cones são inclinados para se ajustarem ao movimento da câmera, as texturas são alongadas para simular *motion-blur*, e por fim roladas para modelar a gravidade. Cada uma das quatro texturas é rolada com velocidade diferente e possui gotas com tamanhos diferentes para a obtenção de efeitos de *parallax*.

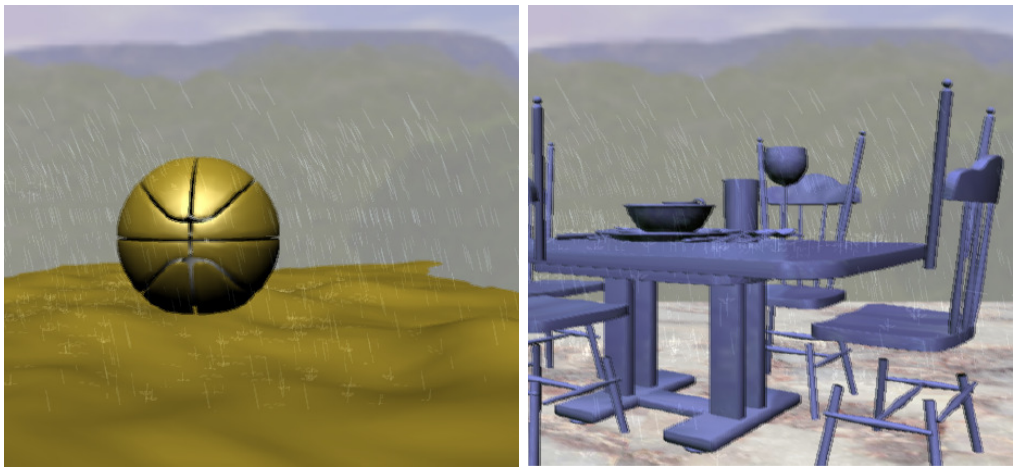
2.3 Métodos Baseados em Sistemas de Partículas

Desde a sua introdução por Reeves [19], no início dos anos 80, os sistemas de partículas estão presentes na modelagem de inúmeros fenômenos em computação gráfica. Esses sistemas possuem características bastante interessantes para a realização de uma animação rápida e realística da chuva, como a simplicidade de detecção de colisões com objetos da cena, facilidade para a aplicação de física (gravidade, vento e *rendering*), além da possibilidade de implementação quase direta em GPU. A animação da chuva usando partículas em geral combina sistemas

Figura 2.5: Animação de neve utilizando método de mapeamento de textura em cones. (Imagem retirada de [18]).

de partículas com modelos baseados em física para a interação da luz com as gotas.

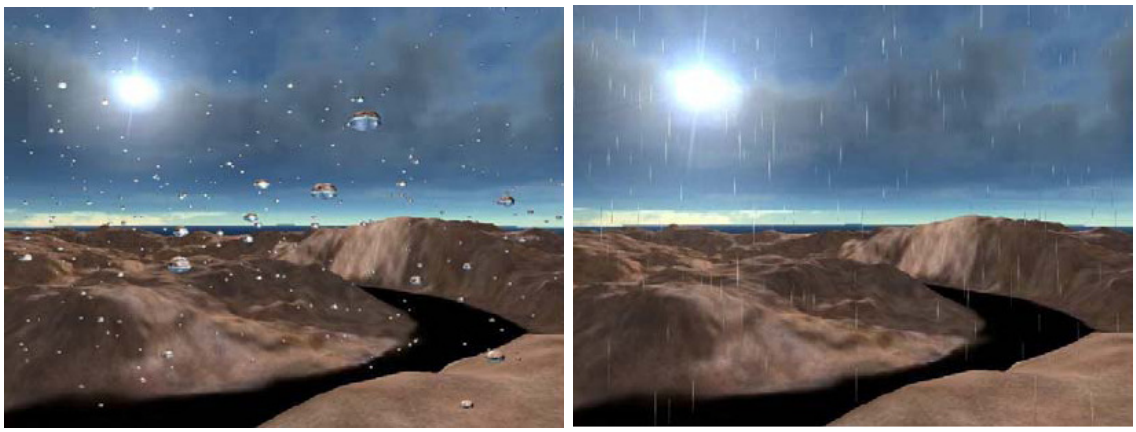
A proposta [2] apresenta um dos primeiros sistemas capazes de realizar a animação da chuva e alguns efeitos relacionados à taxas interativas em GPU. O sistema de partículas conta com a modelagem do efeito do vento, além da detecção de colisão com objetos da cena. O trabalho também apresenta uma interessante proposta baseada na rotação de *point sprites* para a animação dos respingos produzidos pelas gotas (Figura 2.6). A etapa de *rendering* também conta com efeitos de iluminação e *depth of field* (DOF).



(a) (b)

Figura 2.6: Método *real time*, baseado em GPU, para a animação da chuva, além de efeitos como vento, respingos e DOF. (Imagem retirada de [2].)

No trabalho de [20] foi proposto um método baseado em partículas, e acelerado em GPU, capaz de produzir uma animação da chuva em tempo interativo e com grau de realismo satisfatório. A etapa de *rendering* conta com a modelagem da refração de todo o ambiente dentro de cada gota. O objetivo desta metodologia é fornecer algumas simplificações na modelagem do processo de refração/reflexão da luz que permitam alcançar resultados semelhantes aos obtidos com *ray tracing*, com a vantagem de ser em tempo interativo. A Figura 2.7 ilustra resultados obtidos com essa metodologia.



(a) (b)

Figura 2.7: Método para a animação da chuva, acelerado em GPU, com o *rendering* baseado em física. (Imagem retirada de [20].)

O trabalho de [3] apresenta um sistema de partículas acelerado em GPU e renderizado utilizando o banco de imagens de gotas produzido no trabalho de [7]. O aplicativo desenvolvido oferece ao usuário o controle de uma série de parâmetros em tempo de execução, como direção do vento, *fog*, direção da luz, dentre outros. Esse trabalho faz uso de recentes funcionalidades do modelo de programação *Shader 4.0*, tais como *TextureArrays* e *StreamOut*, obtendo resultados visuais bastante convincentes (Figura 2.8), além de ótimo desempenho, porém não apresenta contribuições teóricas significativas.

Em [5] os autores apresentam uma proposta que visa adaptar o sistema de partículas à situações onde o usuário se move constantemente, como o caso de simuladores e jogos eletrônicos. O trabalho apresenta um sistema multiresolução para a representação do sistema de partículas bastante interessante. Basicamente o método adapta a distribuição e o raio das partículas de acordo com a posição do observador: quanto mais próximo, maior o número de partículas e menor o raio de cada partícula. A

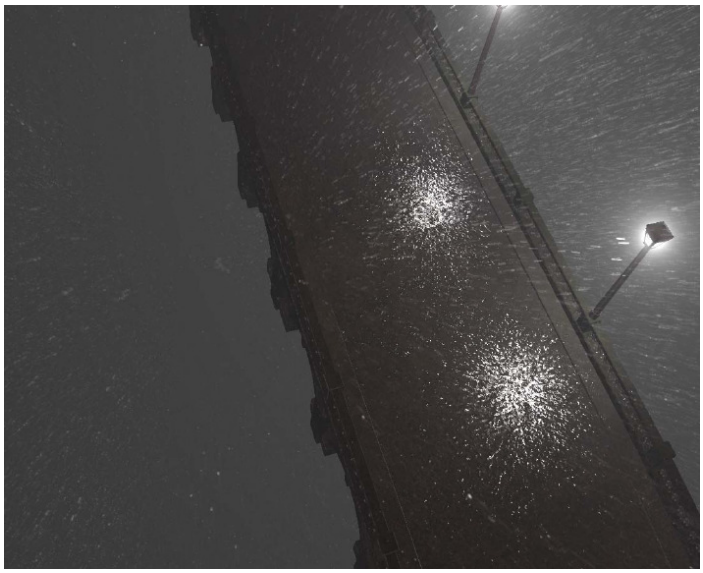


Figura 2.8: Aplicativo para animação tempo real da chuva (com ótima qualidade visual) que possibilita ao usuário o controle de diversos parâmetros de simulação e *rendering*. (Imagem retirada de [3].)

metodologia proposta também explora o conceito de *rain zones*, com propostas para uma transição suave entre regiões com diferentes tipos de chuva. A proposta apresenta taxas de FPS bastante satisfatórias, no entanto, utiliza apenas mapeamento de textura tradicional para o *rendering*, sem nenhum modelo avançado de iluminação. A Figura 2.9 ilustra a animação da chuva através desse método.

Figura 2.9: Animação da chuva através de método multiresolução. (Imagem retirada de [5].)

Capítulo 3

Animação Computacional de Fluidos

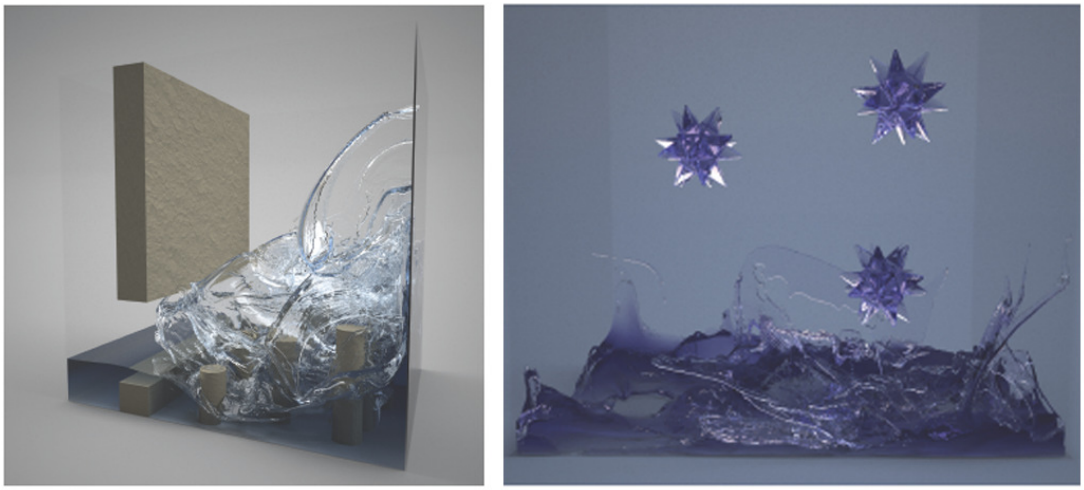


Figura 3.1: Animação e *rendering* de fluidos baseada em física. (Imagem retirada de [21].)

O aumento do poder computacional, do nível de paralelismo nas instruções, além dos avanços ocorridos na área de algoritmos, têm permitido que simulações cada vez mais complexas sejam realizas à taxas aceitáveis. No caso particular da simulação de fluidos, já é possível contar com diversas soluções interativas para as equações de Navier-Stokes em três dimensões [22] [23] [24]. Essas equações formam o modelo que descreve a evolução temporal de um volume de fluido quando sujeito à forças internas, derivadas da viscosidade e pressão, além de forças externas, como a gravidade por exemplo).

Inicialmente essas equações foram resolvidas utilizando metodologias (discretizações) baseadas em malhas [25] [26], ou seja, metodologias eulerianas. Esses métodos fixam o olhar em um ponto fixo do espaço e observam como propriedades do fluido medidas nesse ponto variam no tempo. Em seguida métodos (discretizações) baseados em partículas [27] [28], ou lagrangianos, foram empregados na solução de Navier-Stokes com uma série de vantagens sobre os métodos de malha. Métodos lagrangianos acompanham uma partícula que "carrega" propriedades do fluido, e observam como essas propriedades variam no tempo e no espaço.

3.1 Equações de Navier-Stokes para Animação de Fluidos

Os primeiros modelos de fluidos utilizados em computação gráfica seguem a filosofia Euleriana da mecânica dos fluidos. Essa filosofia é baseada em uma visão *top down* do fluido: assim o fluido é considerado um sistema contínuo sujeito à leis de conservação e equações de estado conectando as variáveis macroscópicas da pressão (p), densidade (ρ), e velocidade ($\vec{u}$). A conservação de massa deve garantir que não ocorram variações na massa total de um sistema isolado. Nos modelos Eulerianos a conservarão de massa é modelada pela equação, também chamada equação de continuidade [29]:

$$\frac{\partial \rho}{\partial t} + \bigtriangledown \cdot (\rho \vec{u}) = 0. \tag{3.1}$$

Essa equação modela a variação da massa no tempo como função do fluxo de densidade devido à velocidade ($\vec{u}$), ou seja, a densidade variou porque entrou ao saiu fluido do sistema devido ao movimento do fluido. Para fluidos incompressíveis a densidade não varia e a expressão (3.1) pode ser reescrita como:

$$\bigtriangledown \cdot \vec{u} = 0. \tag{3.2}$$

A equação de conservação do momento linear do fluido, conhecida como equação de Navier-Stokes, é obtida aplicando a terceira lei de Newton à uma porção infinitesimal (dV) de fluido [29]:

$$\rho\left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot (\bigtriangledown \vec{u})\right) = - \bigtriangledown p + \rho \vec{g} + \mu \bigtriangledown^2 \vec{u} + F. \tag{3.3}$$

Enquanto no lado direito da equação (3.3) temos as forças de pressão ($\bigtriangledown p$), gravidade ($\rho\vec{g}$), viscosidade ($\mu \bigtriangledown^2 \vec{u}$) e forças externas ($F$), no lado esquerdo temos a massa específica multiplicando a aceleração, ou seja, a terceira lei de Newton.

Essa equação também precisa de condições iniciais $\vec{u}\,(t = 0, x, y, z)$, além de condições de fronteira. A ideia das condições de fronteira é modelar um comportamento diferenciado próximo à borda do fluido. Esse comportamento pode ser periódico ou reflexivo.

3.2 SPH em Animação de Fluidos

O SPH (*Smooth Particle Hydrodynamics*) é um método lagrangiano, livre de malha (*meshfree*), utilizado para a solução numérica de equações diferenciais parciais. Desenvolvido inicialmente para tratar problemas em astrofísica, o SPH foi introduzido na comunidade de computação gráfica por Desbrun em [10], onde foi utilizado para simular corpos deformáveis. Em seguida Muller [27] adaptou o SPH para animar fluidos com superfície livre e tensão superficial, como é o caso da água em um copo. O SPH utiliza um sistema de partículas e a teoria da interpolação para obter a solução numérica das equações de Navier-Stokes.

O método traz uma série de vantagens para a simulação de fluidos em computação gráfica, tais como eficiência computacional, simplicidade no tratamento das colisões [30], além da trivial conservação de massa. Uma característica comum dos métodos livre de malha é a necessidade de métodos auxiliares para a extração da superfície livre do fluido. Assim, o SPH também possui essa desvantagem e necessita de um método auxiliar, como o *marching cubes*, para a reconstrução da superfície do fluido. O *marching cubes* é um método, proposto em [31], para a reconstrução de uma iso-superfície a partir de um campo escalar.

3.2.1 Equações do SPH

O SPH é fundamentado em dois elementos básicos, a saber: uma função *kernel* (W) para interpolação e um sistema de partículas que representa uma versão discreta (amostra) do fluido [32]. No SPH os termos das equações de Navier-Stokes são modelados por um sistema de forças que atuam em cada partícula.

Uma quantidade escalar A, representando alguma grandeza macroscópica de interesse, pode ser computada utilizando a função *kernel* W, através da equação a seguir:

$$A = \int_{Space} \mathbf{A}(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}', \tag{3.4}$$

onde h é o raio de suavização, $\mathbf{r} - \mathbf{r}'$ é a distância da partícula corrente para a partícula vizinha, e a função $W(\mathbf{r} - \mathbf{r}', h)$ é um *kernel* de interpolação que satisfaz algumas propriedades, tais como normalização, função Dirac e suporte compacto.

A propriedade de normalização é dada por:

$$\int_r W(\mathbf{r}, h)dr = 1. \tag{3.5}$$

A propriedade de função Dirac, como limite quando h tende a zero, é dada por:

$$\lim_{h\to 0} W(\mathbf{r}, h) = \delta(\mathbf{r}). \tag{3.6}$$

A propriedade de suporte compacto garante que se tomarmos uma amostra de $\mathbf{A}$, então $\mathbf{A}(\mathbf{r}')$ na equação (3.4) será considerada apenas onde $|\mathbf{r} - \mathbf{r}'| \leq h$. A Figura 3.2 ilustra este processo, onde dado o raio de suavização h, as partículas que serão utilizadas na computação são as partículas internas ao raio. As partículas externas serão desconsideradas da computação.

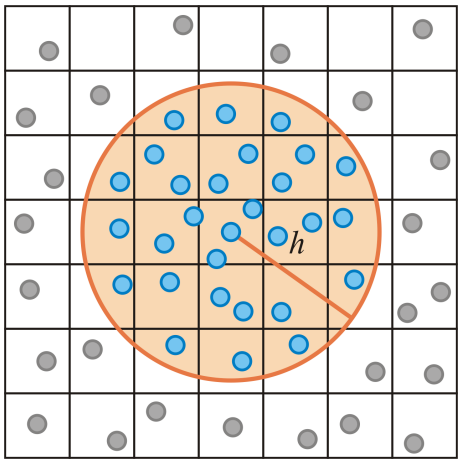


Figura 3.2: A propriedade de suporte compacto.

A versão discreta da equação (3.4) é dada por:

$$A_i = \sum_j A_j V_j W(\mathbf{r}_{ij}, h). \tag{3.7}$$

onde j é cada uma das partículas vizinhas, $V_j = \frac{m_j}{\rho_j}$ representa o volume de fluido atribuído à partícula j e $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ é a distância entre a partícula corrente i e sua vizinha j.

Em consequência das propriedades mencionadas acima, é possível mostrar que para a computação dos operadores diferenciais gradiente (∇) e laplaciano (∇^2) de um campo escalar A, basta aplicar o operador diretamente na função *kernel* [33]:

$$\nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \ \nabla W(\mathbf{r}_{ij}, h), \tag{3.8}$$

$$\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \ \nabla^{\mathbf{2}} W(\mathbf{r}_{ij}, h). \tag{3.9}$$

Resolvendo Navier-Stokes com SPH

O processo de solução de Navier-Stokes através de partículas possibilita algumas simplificações bastante interessantes do ponto de vista computacional. A primeira delas é na equação (3.1), que garante a conservação de massa para os fluidos eulerianos. Em modelos lagrangianos, como o SPH, o número de partículas é constante, assim a conservação de massa é garantida automaticamente, e essa equação não precisa ser computada.

A segunda simplificação é na equação de conservação de momento. Uma vez que as partículas movem-se junto com o fluido, não é necessário o termo advectivo $\vec{u} \cdot (\bigtriangledown \vec{u})$ no lado esquerdo da equação (3.3). Esse termo já está implicitamente modelado pela derivada total, assim a variação do campo de velocidade é simplesmente a taxa de variação da velocidade das partículas no tempo e o lado esquerdo da equação pode ser substituído pela derivada temporal da velocidade $\frac{d\vec{u}}{dt}$.

Então, a equação (3.3) é reescrita da seguinte maneira:

$$\frac{d\vec{u}}{dt} = \frac{1}{\rho}\bigg(- \bigtriangledown p + \rho \vec{g} + \mu \bigtriangledown^2 \vec{u} \bigg). \tag{3.10}$$

A modelagem por partículas também oferece várias possibilidades para o tratamento das condições de fronteira [34]. O caso mais simples e menos custoso é a reflexão perfeita. Essa reflexão é aplicada como uma força externa totalmente independente do SPH. A utilização de forças de repulsão nas fronteiras do fluido pode ser um pouco mais custosa, porém produz resultados mais precisos. Essa força tem a função de empurrar as partículas para longe da borda. Outras soluções são baseadas na inserção de partículas virtuais, ou fantasmas, na fronteira do fluido. Essa solução pode produzir resultados bem precisos, pois a solução é influenciada pelo kernel do SPH. Porém os resultados são altamente dependentes do local de inserção das partículas fantasmas.

3.2.2 *Kernels* de Suavização

Um aspecto crucial dos métodos livres de malha é como a aproximação de funções é realizada com base em uma coleção de pontos [33]. Uma vez que não existe uma malha para fornecer as relações topológicas, essa aproximação não é uma tarefa trivial. No método SPH, os *kernels* de suavização são utilizados para esse fim. A escolha correta desses *kernels* será determinante para vários aspectos do método, como estabilidade, precisão e desempenho.

Nesse trabalho serão utilizados os *kernels* para propósitos específicos desenvolvidos nos trabalhos [10] e [27]. O objetivo desses autores foi propor *kernels* mais realistas para cada quantidade física que está sendo interpolada. Como ilustrado na Figura 3.3, todos os *kernels* de suavização são normalizados, além de serem funções pares: $W(\mathbf{r}, h) = W(-\mathbf{r}, h)$. Essas duas propriedades garantem que a aproximação seja de segunda ordem, isto é possua grau de continuidade 2 (C^2).

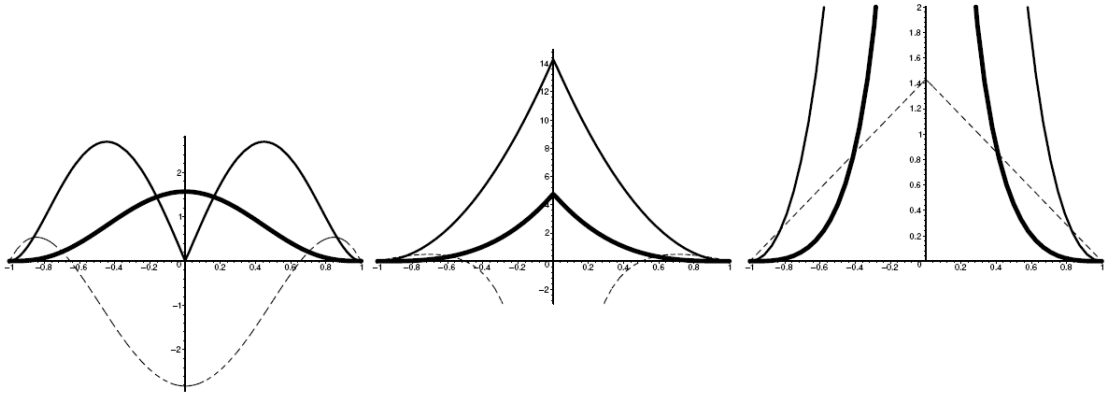


Figura 3.3: Gráficos das funções *kernels* utilizadas: W_{poly6}, W_{spiky} e W_{visc} respectivamente. Linhas grossas representam a função, linhas mais finas seu gradiente e linhas tracejadas seu laplaciano. (Imagem retirada de [27].)

O *kernel* abaixo é utilizado para a computação da densidade. Sua principal vantagem é a eficiência, uma vez que não é necessário computar raízes, pois a distância (r) sempre aparece elevada ao quadrado.

$$W_{poly6}(r,h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & \text{para} \quad 0 \leq |r| \leq h \\ 0 & \text{caso contrário} \end{cases} \tag{3.11}$$

No entanto, o valor do gradiente desse *kernel* decai até zero quando a distância entre as partículas tende a zero, como ilustra o gráfico à esquerda na Figura 3.3. Se esse *kernel* for utilizado na computação da força de pressão, as partículas irão formar aglomerados (*clusters*) quando estiverem próximas. Esse efeito não corresponde à realidade, mas é contornado com a utilização do *kernel* abaixo para computar a pressão:

$$W_{spiky}(r,h) = \begin{cases} \frac{15}{\pi h^6}(h - r)^3 & \text{para} \quad 0 \leq |r| \leq h \\ 0 & \text{caso contrário} \end{cases} \tag{3.12}$$

A Figura 3.3 ilustra esse *kernel*: quanto menor a distância, maior é o valor do seu gradiente. O efeito obtido é uma força de repulsão proporcional à distância, quanto mais próximas as partículas estão, maior é a força de repulsão entre elas. Esse resultado é coerente com o que ocorre na realidade.

A viscosidade é um fenômeno que transforma a energia cinética em calor, reduzindo a velocidade das partículas. Observando a Figura 3.3, é possível notar que o valor do laplaciano do *kernel* W_{poly6} é negativo para quase todo o seu domínio. Quando esse *kernel* é utilizado para computar a viscosidade, produz resultados imprecisos, pois as forças geradas podem fazer o valor da velocidade aumentar.

$$W_{visc}(r,h) = \begin{cases} \frac{15}{2\pi h^3}(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) & \text{para} \quad 0 \leq |r| \leq h \\ 0 & \text{caso contrário} \end{cases} \tag{3.13}$$

A solução é utilizar o *kernel* W_{visc}. Como pode ser observado no gráfico da Figura 3.3, esse *kernel* possui o laplaciano positivo para todo o seu domínio, eliminando o surgimento de forças que aumentam o valor da velocidade.

Capítulo 4

Iluminação Global com a PRT

Figura 4.1: Iluminação global utilizando a PRT. (Imagem retirada de [35].)

A *Precomputed Radiance Transfer* (PRT) é um método de iluminação global utilizado como alternativa aos custosos métodos baseados em *ray tracing* [8]. Além da vantagem de alcançar taxas requeridas por processos interativos, a PRT possui outras características bastante interessantes, como a flexibilidade com relação ao número de fontes de luz, além da possibilidade do uso de *environment lighting*, onde a luz pode vir de todas as direções. Outra característica interessante é a capacidade de modelar todos os efeitos de transporte, como sombras, cáusticas, interreflexões e espalhamento de sub-superfície (Figura 4.2).

A ideia do método é pré-computar a luz incidente, bem como a função de transferência do objeto, para em seguida expressá-las em uma base de harmônicos esféricos. Dispondo dessas expressões de todas as fontes de iluminação e funções

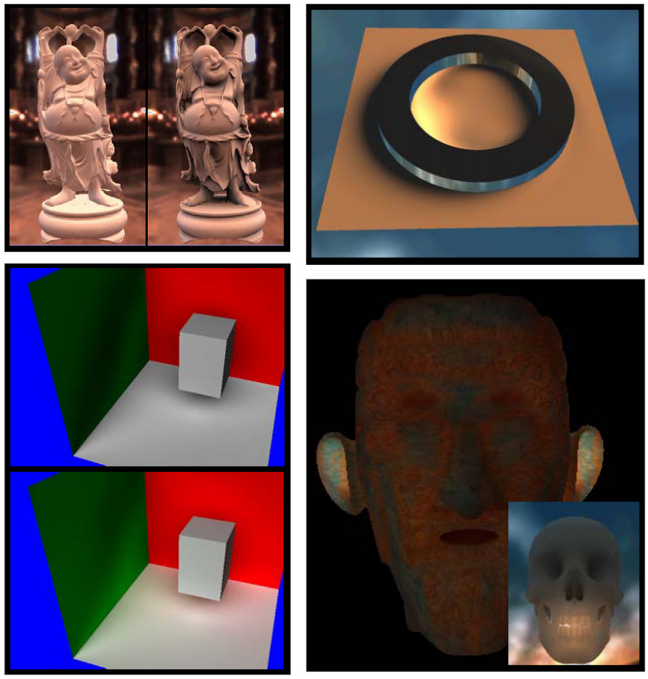


Figura 4.2: Efeitos produzidos através da PRT: sombras, cáusticas, interreflexões e espalhamento de sub-superfície. (Imagem retirada de [36].)

de transferência, em tempo de execução, a equação de *rendering* é reconstruída aplicando operações de adição e multiplicação sobre os coeficientes gerados. Dessa forma, a integral dos raios de luz incidente é reduzida a um simples produto entre os vetores de coeficientes da função de iluminação e da função de transferência. A radiância em cada vértice é então computada usando *Gouraud shading* [37]. Nesse trabalho a PRT é utilizada para computar a cor da chuva, com base em um *environment map*, em tempo real [6].

4.1 Equação de *Rendering*

A equação de *rendering* [38] combina essencialmente a teoria da ótica geométrica com a lei da conservação de energia (termodinâmica) com o objetivo de equacionar o problema de iluminação como um transporte de energia radiante.

$$L_o(x, \vec{\omega_o}) \;=\; L_e(x, \vec{\omega_o}) + \int_\Omega f_r(x, \vec{\omega_i} \rightarrow \vec{\omega_o}) L(x', \vec{\omega_i}) V(x, x') G(x, x') d\omega_i. \qquad (4.1)$$

Na equação (4.1) temos:

Ω = domínio de todas as possíveis direções.

$L_o(x, \vec{\omega_o})$ = intensidade de luz refletida na posição x e direção $\vec{\omega_o}$.

$L_e(x, \vec{\omega_o})$ = luz emitida pelo próprio objeto a partir de x, na direção $\vec{\omega_o}$.

$f_r(x, \vec{\omega_i} \rightarrow \vec{\omega_o})$ = é a BRDF da superfície no ponto x. Ela descreve como a luz reflete em uma superfície composta de um determinado material. Sua função é expressar o quanto da luz que chega no ponto x, ao longo da direção $\vec{\omega_i}$, é refletida na direção de saída $\vec{\omega_o}$.

$L(x', \vec{\omega_i})$ = intensidade de luz emitida por um ponto x', na direção $\vec{\omega_i}$ definida por $x - x'$.

$V(x, x')$ = função binária de visibilidade entre x e x'. Se o ponto x for visível a partir de x' seu valor é 1, caso contrário é 0.

$G(x, x')$ = relação geométrica entre x e x'.

A equação de *rendering* computa a intensidade de luz refletida na posição x e direção $\vec{\omega_o}$ como a soma das contribuições da luz emitida $L_e(x, \vec{\omega_o})$ e da luz incidente, como ilustra a Figura 4.3. A contribuição dessa última é computada integrando as contribuições de todas as direções que incidem naquele ponto, dada pelo produto $L(x', \vec{\omega_i})V(x, x')G(x, x')$ na equação (4.1). Infelizmente essa equação não possui solução analítica e sua computação direta é muito custosa. No lado direito temos uma integral cujo domínio consta de infinitas direções, tornando sua computação impraticável para aplicações de tempo real.

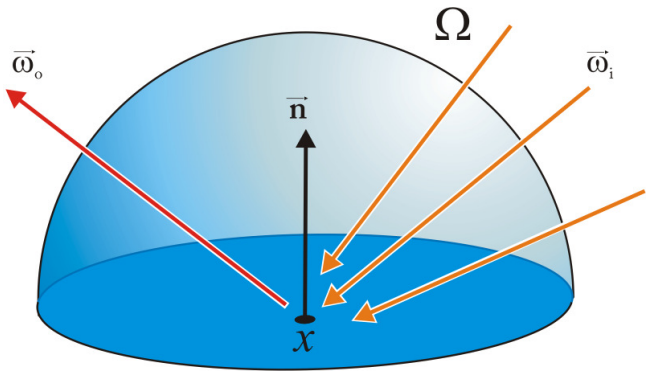


Figura 4.3: Equação de *rendering*: computa a intensidade de luz refletida na posição x e direção $\vec{\omega_o}$ integrando as contribuições de todas as direções $\vec{\omega_i}$ que incidem naquele ponto.

No entanto, a equação de *rendering* sofre algumas simplificações quando utilizamos a PRT, a saber: (i) os objetos da cena são tidos como não emissores; e (ii) as direções de chegada são independentes da posição do ponto x, ou seja, é assumido que as fontes estão no infinito. Dessa forma o termo $L_e(x, \vec{\omega_o})$ é omitido e o termo

$L(x', \vec{\omega_i})$ é simplificado para $L(\vec{\omega_i})$. As funções $f_r(x, \vec{\omega_i} \rightarrow \vec{\omega_o})$, $V(x, x')$ e $G(x, x')$ também são agrupadas em uma única função, chamada função de transferência. A função de transferência tem como objetivo expressar como o ponto x vai responder a iluminação incidente. Então a equação (4.1) pode ser reescrita como [39]:

$$L_o(x, \vec{\omega_o}) \ = \ \int_\Omega L(\vec{\omega_i}) \ T(x, \vec{\omega_o}, \vec{\omega_i}, x') d\omega_i. \tag{4.2}$$

Vale ressaltar que essas funções são computadas independentemente e combinadas para produzir o resultado final.

4.2 Projeção e Reconstrução de Funções

Seja uma função $f : C \rightarrow D$, tal que $\{C \subseteq R^n, D \subseteq R^m\}$, e seja B uma base do espaço funcional $F(C, D)$ definido por esses dois conjuntos, suposta ortonormal em relação ao produto interno dado por $< g, h >= \int_c < g(x)h(x) > dx$. Isto é: se β_1 e β_2 são funções de B então:

$$\int_c ||\beta_1(x)||^2 dx = 1, \tag{4.3}$$

e

$$\int_c < \beta_1(x)\beta_2(x) > dx = 0. \tag{4.4}$$

Ou seja, a integral do produto de duas funções de base será 0 quando estas funções forem diferentes e 1 quando elas forem iguais.

Como B pode ter dimensão não finita tomemos um subconjunto finito de B, $B' = \{\beta_i, i = 0, ..., p\}$. A melhor aproximação de f no subespaço gerado por B'_i, ou seja, sua projeção sobre esse subespaço, é então obtida por:

$$\tilde{f}(x) = \sum_{i=0}^{p} C_i\beta_i, \quad onde \quad C_i = \int_c < f(x)\beta_i(x) > dx. \tag{4.5}$$

Nesse contexto, seria desejável dispor de uma base para a qual, por meio de uma formulação simples se pudessem escolher subconjuntos B' que propiciassem obter representações de f no formato dado em (4.5), as mais compactas possíveis dado o grau de aproximação desejado. Quando $C = R^2$ a base constituída pelos harmônicos esféricos cumpre razoavelmente esse papel.

4.3 Monte Carlo

A equação que descreve a intensidade da luz de saída (seção 4.1) contém uma integral de todas as infinitas direções incidentes, que não possui solução analítica. Nessas condições o método de Monte Carlo pode ser empregado para obter a solução numérica dessa integral. As inúmeras direções são inicialmente reduzidas a um número N de direções, onde o valor de N define a qualidade da solução que será obtida.

Utilizando conceitos probabilísticos básicos é possível provar que se $f(x)$ é a função e $p(x)$ é a função de densidade de probabilidade (PDF) empregada para obter uma amostra x_i, então:

$$\int f(x)dx = \int \frac{f(x)}{p(x)}p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)w(x_i) \quad onde \quad w(x_i) = \frac{1}{p(x_i)}. \tag{4.6}$$

Ou seja, o método de Monte Carlo escala cada amostra pela PDF, acrescenta ao somatório, e por fim divide a soma pelo total de amostras. Essa é uma boa aproximação para a integral do lado direito da equação (4.1), quanto maior o número de amostras, melhor a aproximação. No contexto desse trabalho, aproximações obtidas com um número médio de 10000 pontos foram consideradas satisfatórias, uma vez que o tamanho da gota a ser renderizada é muito pequeno.

4.4 Harmônicos Esféricos

Os harmônicos esféricos definem uma base ortonormal para funções reais definidas na esfera unitária, de forma análoga a transformada de Fourier sobre o círculo unitário [40]. Dessa forma, as direções possíveis são definidas em um domino bidimensional, e então, para cada direção é computado um vetor de coeficientes.

4.4.1 Polinômios de Legendre

Os polinômios de Legendre ($P_l^m(x)$) são utilizados para computar as funções da base dos harmônicos esféricos. Seus argumentos são a banda $l \in N$ e o parâmetro $m \leq l$. Esses polinômios são definidos no intervalo $[-1, 1]$, retornando sempre um número real:

$$
\begin{aligned}
&P_0^0(x)\\
&P_1^0(x), P_1^1(x)\\
&P_2^0(x), P_2^1(x), P_2^2(x)\\
&P_3^0(x), P_3^1(x), P_3^2(x), P_3^3(x)\\
&...
\end{aligned}
$$

A computação desses polinômios segue uma formulação recursiva. Primeiramente, dado m o método computa o polinômio da banda mais baixa para a qual o parâmetro pode ser m (P_m^m). Isso é feito usando as expressões:

$$P_0^0(x) = 1, \tag{4.7}$$

$$P_m^m = (-1)^m(2m-1)!!(1-x^2)^{\frac{m}{2}}, \quad m > 0, \tag{4.8}$$

onde $(2m-1)!!$ representa o produto de todos os números ímpares $\leq 2m-1$. Em seguida obtemos o polinômio de parâmetro m da banda seguinte:

$$P_{m+1}^m = x(2m+1)P_m^m. \tag{4.9}$$

Tendo esses dois valores, a expressão recursiva é utilizada:

$$(l-m)P_l^m = x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m. \tag{4.10}$$

Fazendo $l = m+2$:

$$P_{m+2}^m = \frac{1}{2}\Big[x(2m+3)P_{m+1}^m - (2m+1)P_m^m\Big]. \tag{4.11}$$

E então para $l = m+3$:

$$P_{m+3}^m = \frac{1}{3}\Big[x(2m+5)P_{m+2}^m - (2m+2)P_{m+1}^m\Big]. \tag{4.12}$$

E assim por diante, até finalmente chegar a expressão de um polinômio de Legendre genérico (P_l^m) através de:

$$P_l^m = \frac{1}{(l-m)}\Big[x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m\Big]. \tag{4.13}$$

4.4.2 Projeção em Harmônicos Esféricos

As funções de base dos harmônicos esféricos são definidas sobre a esfera e parametrizadas através dos ângulos θ e φ, além da banda l e um parâmetro m. O parâmetro m varia de $-l$ a l. Para uma aproximação usando l bandas são gerados $(1 + 3 + ... + (2l - 1)) = l^2$ coeficientes. Valores baixos de l representam funções de base de baixa frequência, valores altos representam as frequências mais altas. Essas funções são expressas por:

$$Y_l^m(\theta, \varphi) = \begin{cases} \sqrt{2} K_l^m \cos(m\varphi) P_l^m(\cos\theta), & m > 0 \\ \sqrt{2} K_l^m \sin(-m\varphi) P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0 P_l^0(\cos\theta), & m = 0 \end{cases} \tag{4.14}$$

O parâmetro K, utilizado nas funções acima, é apenas um fator de escala destinado a garantir que:

$$\int \int (Y_l^m(\theta, \varphi))^2 d\theta \sin\theta d\varphi = 1. \tag{4.15}$$

O processo de projeção sobre o sub-espaço gerado pelos harmônicos esféricos, até uma determinada banda L, calcula uma coordenada para cada (Y_l^m) pertencente a essa faixa integrando o produto entre a função f e esse harmônico:

$$C_l^m = \int_s f(\vec{\omega}) Y_l^m(\vec{\omega}) ds, \tag{4.16}$$

onde $|m| < l$ e $l \leq L$.

A forma discreta da equação acima é aproximada, utilizando N amostras, através do método de Monte Carlo:

$$C_l^m \approx \frac{1}{N} \sum_{j=1}^{m} f(\vec{\omega_j}) Y_l^m(\vec{\omega_i}) w(\vec{\omega_j}) \quad onde \quad w(\vec{\omega_j}) = \frac{1}{p(\vec{\omega_j})}. \tag{4.17}$$

Se a distribuição empregada para gerar as direções $w(\vec{\omega_j})$ for uniforme, o valor da probabilidade associado a cada direção é $\frac{1}{4\pi}$, uma vez que 4π é o valor da área

da superfície da esfera unitária. Assim, a equação (4.17) fica reescrita como:

$$C_l^m \approx \frac{4\pi}{N} \sum_{j=1}^{m} f(\vec{\omega_j}) Y_l^m(\vec{\omega_i}). \tag{4.18}$$

O processo de reconstrução faz o inverso da projeção, ou seja, ele soma cópias da função de base Y_l escaladas pelo coeficiente correspondente C_l:

$$\tilde{f}(\vec{\omega}) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} C_l^m Y_l^m(\vec{\omega}) = \sum_{i=0}^{n^2} C_l Y_l(\vec{\omega}). \tag{4.19}$$

Para que as direções $\vec{\omega}$ possam ser assumidas como equiprováveis é necessário gerar uma amostragem uniforme sobre a esfera unitária. Para isso a esfera é planificada em um plano (quadrado) unitário, esse plano é dividido em $\sqrt{n} \times \sqrt{n}$ células. Então escolhe-se uma amostra aleatória em cada célula, como ilustra a Figura 4.4.

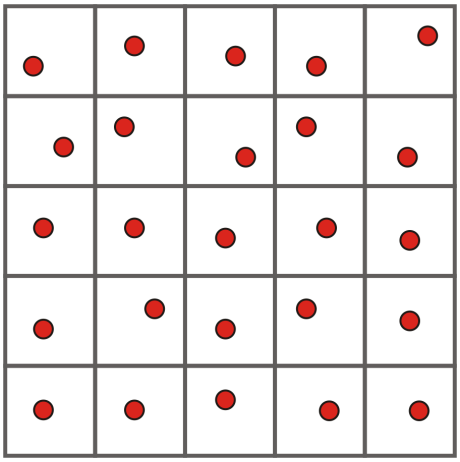


Figura 4.4: Amostragem estratificada da esfera.

Esse tipo de amostragem é chamado de amostragem estratificada e tem como característica baixa variância. As coordenadas de cada amostra nesse plano são mapeadas sobre a esfera utilizando a expressão:

$$(x, y) \rightarrow (2 \arccos(\sqrt{1-x}), 2\pi y) \rightarrow (\theta, \phi). \tag{4.20}$$

Assim, para cada uma dessas células será computado um vetor de coeficientes para as funções de base dos harmônicos esféricos. A partir desse processo é possível projetar qualquer função definida sobre a esfera na base de harmônicos esféricos.

A projeção é a parte final da etapa *off-line* do método, o resultado desse processo são dois vetores, um para a função de iluminação e outro para a função de transferência, com l^2 coeficientes cada um. A parte *real time* do algoritmo computa a cor final para cada vértice, ou *pixel*, através de um produto escalar, realizado em *shader*, desses vetores.

Capítulo 5

Método Proposto

A intenção desse trabalho é o desenvolvimento de um *framework* capaz de realizar uma animação da chuva com realismo físico à taxas interativas. A principal contribuição do trabalho é a integração de diversos modelos baseados em física em um único *framework*, para a realizar a animação da chuva e efeitos relacionados. O *framework* proposto é composto pelos seguintes elementos: (i) um modelo baseado em partículas e renderizado através da PRT para a animação da chuva; (ii) um modelo digital de terreno (MDT), utilizado como superfície para o fluxo produzido pela água da chuva; (iii) o modelo SPH, acoplado a um modelo alternativo para a extração da superfície livre do fluido, para a animação do fluxo superficial sobre o MDT; (iv) um modelo baseado em *sprite* para a animação dos respingos das gotas sobre as superfícies. Ressaltamos que a integração de um modelo de animação de chuva a um modelo de SPH, para simular o escoamento sobre um MDT, não foi encontrada na extensa pesquisa bibliográfica que realizamos. Cada seção desse capítulo irá descrever um desses elementos, finalizando com a seção que descreve a interação entre eles.

5.1 Animação Realística e Interativa da Chuva

A animação da chuva é baseada na proposta de [6], que utiliza um sistema de partículas para a animação e *rendering* da chuva. Então, utilizando a PRT, a aparência final da chuva é influenciada pela iluminação ambiente. Através desse sistema é possível realizar uma animação realística de chuva controlada por diversos parâmetros. Os resultados alcançados pelo aplicativo (Figura 5.1), em tempo interativo, possuem grau de realismo comparável aos obtidos com o uso de métodos baseados em *ray tracing*.

Figura 5.1: Animação da chuva com sistema de partículas e *rendering* utilizando *environment lighting*. (Imagem retirada de [6].)

5.1.1 Dinâmica das Partículas

As partículas da chuva são inicializadas ("nascem") nas células da área de precipitação e aceleradas pela ação da gravidade (g). A área de precipitação é uma grade retangular uniformemente espaçada. Essa grade pode ser vista como uma matriz, onde cada célula possui uma altura fixa. As velocidades horizontais de cada partícula eliminam a possibilidade do surgimento de padrões indesejados devido à uniformidade da grade inicial. Ainda com o intuito de eliminar o surgimento de padrões, as velocidades verticais e horizontais também são inicializadas aleatoriamente.

Como esse sistema de partículas é um sistema *stateless*, ou seja, não existe interação entre as partículas, apenas o instante inicial é armazenado. A posição para o tempo t é computada utilizando as equações do movimento uniforme e uniformemente variado:

Horizontal (M.U.)

$$x_i^t = x_i^0 + v_{ix}^0 \cdot t, \tag{5.1}$$

$$y_i^t = y_i^0 + v_{iy}^0 \cdot t. \tag{5.2}$$

Vertical (M.U.V.)

$$z_i^t = z_i^0 + v_{iz}^0 \cdot t - \frac{g}{2} \cdot t^2. \tag{5.3}$$

Quando uma partícula colide com a superfície do fluido ou com a superfície do terreno, essa partícula é desconsiderada do sistema, ou seja, a partícula "morre". Se uma partícula morre no tempo t, então uma nova partícula é inserida no sistema no tempo $t + 1$ na mesma posição onde a partícula morta foi inicializada. A partícula morta também será contabilizada para a inserção das partículas do fluxo superficial, descrita em detalhes na seção 5.5.

Esse processo de atualização das partículas, e detecção das colisões, é realizado em paralelo na GPU, por um sistema implementado em CUDA. Dessa forma, velocidades e posições são inicializadas em CPU e armazenadas em *arrays* unidimensionais de ponto flutuante com 4 posições *(float4)*. Então esses *arrays* são transferidos para a memória da GPU e atualizados através de um *kernel* CUDA. Um *kernel* CUDA é um processo que é executado simultaneamente nas centenas de processadores presentes na GPU. Como já foi dito, no nosso sistema não há interação entre as partículas, assim ele é mapeado diretamente para a GPU, simplesmente associando a atualização de cada partícula a um *thread*.

5.1.2 *Rendering*

O *rendering* do sistema de partículas, proposto em [6], tem como objetivo computar a cor da chuva através da PRT utilizando a iluminação vinda do ambiente. Assim, a aparência da chuva é afetada pela iluminação de um *environment map*, o que possibilita a obtenção de uma imagem final com elevado grau de realismo.

Como foi descrito no Capítulo 4, a PRT utiliza representações em um sistema de coordenadas dado por harmônicos esféricos, tanto da função de iluminação quanto da função de transferência (RTF), para a determinação da cor de cada *pixel* da imagem. Nesse caso o *environment map* faz o papel da função de iluminação, já para a RTF (*Radiance Transfer Function*) os autores derivam uma forma fechada utilizando óptica geométrica. Essa forma fechada modela as refrações e reflexões dos raios de luz na superfície e no interior da gota.

Em [6], inicialmente imagens de gotas de chuva, extraídas de um vídeo real na forma de texturas, são utilizadas como mapas de opacidade para a cor computada através da PRT. Esse mapa de opacidade tem a função de descrever a geometria da imagem da gota de chuva, além de descrever a interação com a luz. Lembramos que a imagem de uma gota de chuva depende da velocidade de queda, além da velocidade de captação da câmera.

Nesse trabalho, essas imagens foram substituídas por outras de um banco de imagens, produzidas com efeitos fotorealísticos, no trabalho de [7]. Apesar desse banco de imagens dispor de uma certa quantidade de imagens, apenas uma imagem é utilizada. Cada imagem é então escala de acordo com sua velocidade vertical e profundidade através do *vertex shader*.

Cada gota de chuva é representada por uma partícula que consiste de um modelo esférico, além de atributos simples, como posição, velocidade e índice de refração. Na Figura 5.2 temos um *overview* de todo o processo. Assim, para cada partícula do sistema que representa a chuva será associado um mapa de opacidade (*alpha matte*) obtido a partir do banco de imagens de [7]. O *alpha matte* é definido como uma região cuja largura corresponde ao diâmetro da gota e tamanho corresponde ao produto da velocidade vertical pelo tempo de exposição à câmera (Figura 5.2, esquerda-abaixo). Esclarecemos que nesse banco de imagens apenas gotas caindo verticalmente estão representadas.

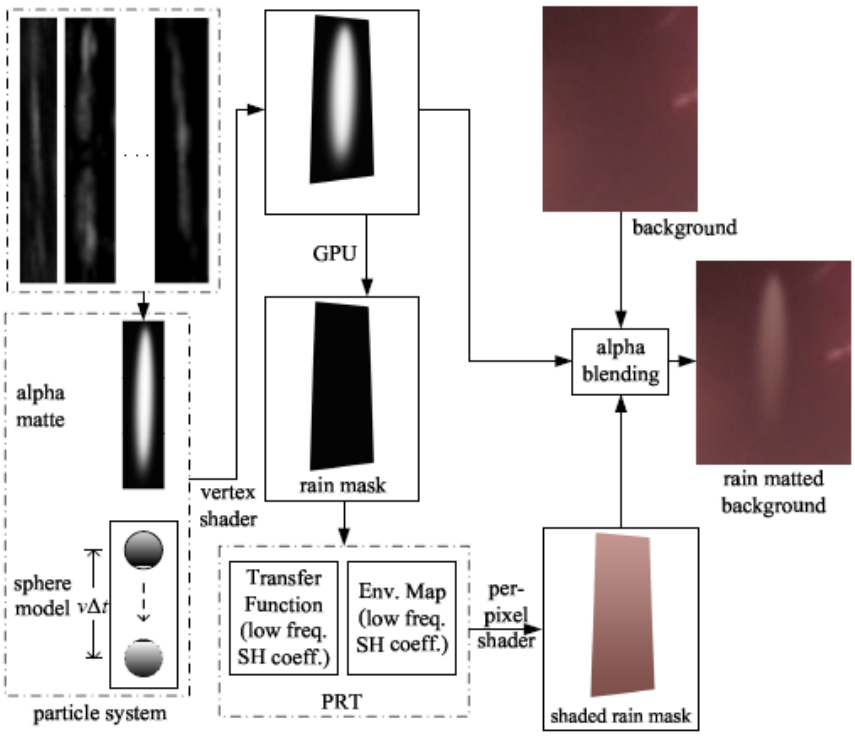


Figura 5.2: *Overview* do processo de *rendering* da chuva. (Imagem retirada de [6].)

A imagem final da gota é obtida fazendo o centro do *alpha matte* e da partícula coincidirem. Então o *alpha matte* é orientado de forma que dois de seus lados se tornem paralelos à velocidade e os outros dois ortogonais à direção de visão. Uma

vez posicionada, a partícula é projetada no plano da imagem. Essa projeção no plano da imagem é definida como *alpha mask* (Figura 5.2, acima-meio). Cada *pixel* do *alpha mask* irá compor a imagem final da gota.

Para computar a cor/brilho de cada partícula dois componentes serão considerados: (i) um vindo do *environment map* (EM), o qual é computado através da PRT; (ii) outro vindo do *background* e acrescentado via *blending*. O *environment map* pode ser gerado baseado em uma imagem ou em um *frame* de vídeo real, como em [6]. A função do EM nesse trabalho é servir com fonte de luz para o *environment lighting*, modificando a aparência da chuva em tempo real.

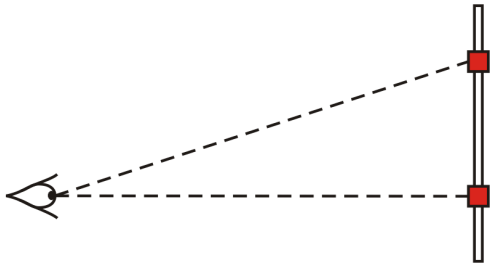


Figura 5.3: A RTF da gota para cada *pixel* irá depender da posição que esse *pixel* ocupa na tela.

Cada *pixel* do *alpha mask* define uma direção de visão (D_v), como ilustra a Figura 5.3. A forma esférica da gota permite derivar uma forma fechada da RTF da gota que será usada pela PRT. Utilizando essa forma fechada da RTF não é necessário o uso de nenhum tipo de *ray tracing* para computar as reflexões internas na gota. A computação do fluxo de luz deixando a gota (*out-going*) em uma certa direção, levando em consideração a contribuição de todas as direções de chegada é muito custosa computacionalmente e impraticável em aplicações em tempo real.

Nesse trabalho esse problema é resolvido através da PRT, que projeta tanto o EM quanto a RTF para cada D_v sobre o sub-espaço gerado pelos harmônicos esféricos, cuja banda é ≤ 4 . Esse processo de projeção é descrito em detalhes na seção 4.4.2. O conjunto de harmônicos gerado consiste de 16 funções de baixa frequência. Em computação gráfica é comum utilizar-se entre 9 e 25 funções de base de harmônicos esféricos. Testes foram feitos utilizando 25 funções, porém os resultados visuais mostraram-se inalterados, por isso optamos por empregar apenas 16 funções.

O produto das representações do EM e da RTF relativa à direção considerada fornece a contribuição do EM para o *pixel*. Vale ressaltar que essas representações são obtidas em uma etapa de pré-processamento. O fato da RTF depender da direção é contornado computando-se previamente sua representação para um conjunto denso de direções (C_d) e empregando-se para uma dada direção (D_v) a representação da direção mais próxima dele que está em C_d. Então, a GPU realiza o produto escalar (entre os vetores das representações obtidas) em *pixel shader*, obtendo a contribuição do EM para o *pixel* em *real time* (Figura 5.2, meio-abaixo).

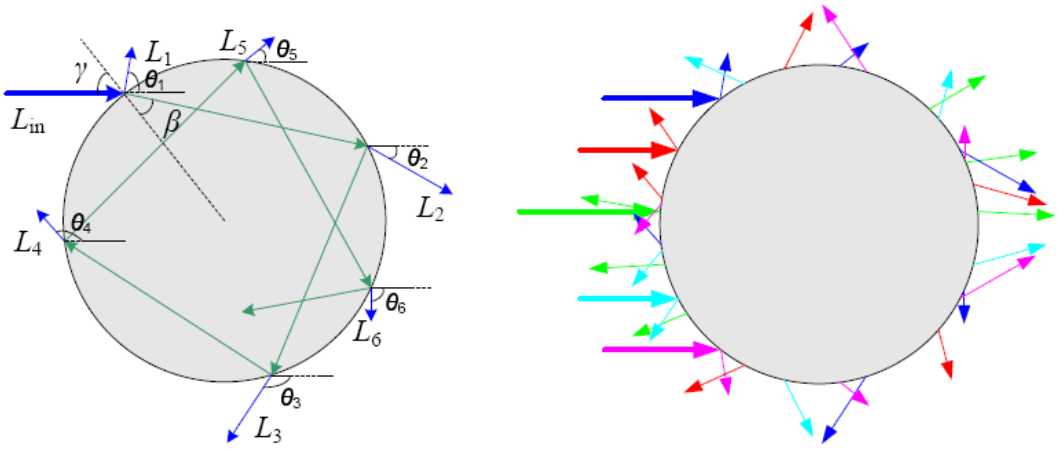
Na etapa final do processo a contribuição do *background* é acrescentada à contribuição do EM para produzir a cor final do *pixel*. Os valores *alpha* do *blending* de cada *pixel* são obtidos da projeção do *alpha matte* (Figura 5.2).

Função de Transferência da Gota

A proposta de Wang et al.'s [6] foi utilizada para a obtenção de uma forma fechada da função de transferência (RTF) da gota. No entanto, tratamos o problema em um contexto discreto e indicamos um processo de computação mais robusto. A reversibilidade da luz garante que a RTF para uma direção de visão (D_v) é idêntica à função de distribuição de saída de luz quando a direção de entrada é D_v. Dessa forma, o foco será obter uma forma fechada para a função de distribuição de saída de luz, assumindo que a entrada se dê por uma única direção D_v. Os raios incidentes (paralelos a D_v) podem atingir a gota em diferentes posições resultando em diferentes ângulos de incidência γ (Figura 5.4).

Sejam I_{in} e dF_{in} a intensidade da luz e do fluxo incidente, respectivamente. Sem perda de generalidade podemos assumir que a direção da luz incidente é a representada por $(0, 0)$ no sistema de coordenadas esféricas local da gota. Além disso, sejam L_n, I_n e dF_n, respectivamente, a direção, intensidade e o fluxo local da luz que sai da gota depois de ser refletida dentro dela $(n - 2)$ vezes.

Observe que o caminho da luz dentro da gota está contido no plano (P) determinado pelo seu centro, o ponto de incidência e a direção da luz incidente. Isso é decorrente do fato de todas as retas suporte das normais de uma superfície esférica se encontrarem no centro dela. Embora os raios incidentes tenham todos a direção $(0, 0)$ eles podem atingir a gota em diferentes posições, resultando em diferentes ângulos de incidência γ com a normal no ponto de contato. Em contrapartida, o ângulo de azimute ϕ é um dos determinados pelo plano P (se ϕ é um deles, $(\phi + \pi) mod 2\pi$ é o outro). Assim, para expressar a direção de saída $L_n = (\theta_n, \phi)$,



(a) (b)

Figura 5.4: (a): Refrações e reflexões da luz após atingir a gota. (b): para 5 raios atingindo a gota, a reversibilidade da luz garante que todas as contribuições outgoing possam ser computadas.

nós podemos nos concentrar apenas em obter θ_n, o que é feito da seguinte maneira:

I) Conforme a lei de Snell, na refração que ocorre no ponto de incidência, o ângulo formado pela luz refratada e a normal é dado por:

$$\beta = arcsin\left(\frac{sin(\gamma)}{\rho}\right), \tag{5.4}$$

sendo ρ o índice de refração da gota em relação ao ar. Isso significa que o ângulo formado por essa luz refratada com a incidente é dado por $\beta - \gamma$.

II) Dado que os ângulos adjacentes à base de um triângulo isósceles são iguais, e as normais em cada ponto de uma superfície esférica são radiais, os ângulos formados com as normais em dois pontos sucessivos em que a luz é refletida dentro da gota com a direção dela entre esses pontos é o mesmo. Como além disso os ângulos da luz incidente e refletida em uma reflexão especular são os mesmos, resulta que todos os ângulos formados pela direção da luz incidente ou refletida dentro da gota com a normal no ponto de reflexão são iguais a β. Isso acarreta que em cada ponto de reflexão a luz varia de um ângulo dado por $\pi - 2\beta$. Como são $n - 2$ reflexões, entre a direção da luz refratada no ponto em que ela entra na gota e a direção com que ela atinge o ponto de saída, existe um ângulo de $(n-2)(\pi-2\beta)$.

III) No ponto de saída, por simetria, o ângulo formado pela direção da luz incidente (que vem de dentro da gota) e a que finalmente sai da gota é, novamente, $\beta - \gamma$.

IV) Somando as contribuições dos itens I-III, temos que o ângulo formado pela luz que incide na gota e a que sai dela é dado por:

$$\psi_n = 2(\beta - \gamma) - (n-2)(\pi - 2\beta) = (n-1)(2\beta - \pi) - 2\gamma + \pi. \tag{5.5}$$

Um ângulo equivalente a ψ_n no intervalo $[-\pi, \pi]$ é dado por:

$$\psi'_n = (\psi_n mod 2\pi) - \pi. \tag{5.6}$$

Como em coordenadas esféricas o ângulo com a normal (θ_n) deve ser não negativo, temos que $\theta_n = ||\psi'_n||$. Considerando conjuntamente as três igualdades dadas se obtém finalmente que:

$$\theta_n = \Theta_n(\gamma) \triangleq ||(((n-1)(2\beta - \pi) - 2\gamma + \pi) mod 2\pi) - \pi||. \tag{5.7}$$

A expressão de ψ_n dada acima tem em

$$m_n = acos\left(\sqrt{\frac{\rho^2 - 1}{(n-1)^2 - 1}}\right)$$

seu único extremo dentro do intervalo $[0, \pi/2]$. Isso significa que, dado um valor de ψ_n, há no máximo dois valores de γ que produzirão esse valor. Na verdade, como a gota é mais densa que o ar, $\rho > 1$, e nesse caso m_n é um máximo de ψ_n. Além disso, como $\beta \leq \gamma$, se $n \leq 2$, temos que ψ_n se situa no intervalo $[-n\pi, 0]$, o que determina que dado ψ'_n existem não mais de $n/2$ valores de ψ_n que o produzirão. Finalmente para cada θ_n há dois valores de ψ'_n: θ_n e $-\theta_n$.

Considerando conjuntamente as observações acima, concluímos que não há mais de $2n$ valores de γ associados ao mesmo valor de θ_n. Como o valor de n é baixo, neste trabalho utilizamos $n \leq 9$, torna-se viável discretizar o espaço dos θ_n, e para cada ponto dessa discretização obter numa etapa de pré-processamento, a contribuição de todos os γ para os quais θ_n define a direção de saída após $(n-2)$ reflexões internas. A direção de saída é que está diretamente relacionada com os valores dos *pixels* da imagem da gota. Melhor, portanto, que ela seja amostrada regularmente e com uma resolução controlada, do que fazer isso em relação aos

ângulos de entrada e ter uma distribuição de espaçamento variável na saída apesar da vantagem de, nesse caso, se poder usar diretamente a expressão (5.7). Essa segunda opção acabaria determinando que um procedimento de regularização da amostragem dos θ_n tivesse de ser efetuado.

Considerando a intensidade da luz de saída (I_n) em função de θ_n, deve ser ressaltado o seguinte. Pela própria maneira como são definidas θ_n e γ (enquanto θ_n define a direção de saída independente do ponto de saída, γ depende do ponto de entrada), a intensidade da luz de saída (I_n) pode apresentar um valor infinito. É o que acontece se o índice de refração da gota for 1. Nesse caso, para qualquer valor de γ, θ_n é zero e I_n tem, portanto, um impulso nesse ponto. Mas mesmo quando $\rho > 1$, o simples fato de $\frac{d\psi_n}{d\gamma}$ se anular em $\gamma = m_n$ faz com que o mesmo se dê com $\frac{d\theta_n}{d\gamma}$ nesse ponto, e em consequência I_n tenha um impulso em $\Theta_n(m_n)$.

Para evitar essas singularidades optou-se por particionar o espaço dos Θ_n, que é $[0, \pi]$, em intervalos iguais e considerar a intensidade média ao longo de cada um deles. Essa intensidade média é sempre finita, porque o fluxo de entrada (e em decorrência o de saída) é finito e portanto, a intensidade só poderia se tornar infinita se o ângulo sólido considerado tendesse a zero. Mas se os ângulos sólidos considerados no cômputo das intensidades médias tiverem a forma $\Delta\theta_n \ \Delta\phi \ \sin(\theta_n)$ com esses três termos não nulos isso não pode acontecer.

Assim, dada uma precisão Δ, considere o intervalo $[0, \pi]$ subdividido em intervalos consecutivos J_m, $m = 0, \ldots, 2M-1$ de tamanho Δ. Seja ainda α_m o valor médio em J_m. Vamos usar o mesmo esquema (aproximando qualquer valor em J_m por α_m) para discretizar tanto θ_n quanto γ, que se situa no intervalo $[0, \frac{\pi}{2}]$. Indicamos a seguir como obtemos a RTF média num intervalo J_m do range dos θ_n.

1) Calculamos a parte não linear de Θ_n, ou seja ψ_n para:

$$\gamma \ = \ k \cdot \Delta, \quad \text{com} \ \ k = 0, 1, \ldots, M.$$

Assumimos Δ suficientemente pequeno para que o valor de ψ_n dentro do intervalo J_m seja aproximado com a precisão desejada pela interpolação linear de seu valor nos extremos do intervalo. Seja (Ψ_k^n), $k = 0, 1, \ldots, M$, o resultado desse cômputo. Obtemos (Ψ_k^n) somando $2\beta(k \cdot \Delta)$ à (Ψ_k^{n-1}). Como ψ_n é uma função côncava de γ tendo como máximo $\psi_n(m_n)$, a lista (Ψ_k^n) é varrida a partir desse ponto, primeiro decrescendo até:

$$\psi_n(0) \ = \ -(n-2)\pi,$$

e depois decrescendo até:

$$\psi_n\left(\frac{\pi}{2}\right) \;=\; -(n-1)\left[\arcsin\left(\frac{1}{\rho}\right) - \pi\right].$$

Em cada uma dessas varreduras, vamos percorrendo tanto o espaço dos Θ_n como os do ψ_n, o que possibilita que o cômputo da RTF seja feito em tempo linear. Não é preciso para cada intervalo J_m no espaço dos Θ_n varrer o espaço dos γ para procurar aqueles intervalos J_k que contribuem para a RTF em J_m, o que tornaria o cômputo quadrático.

2) Sejam $(\sigma_l^0 = \pi - l \cdot \Delta)$ e $(\sigma_l^1 = -\sigma_l^0)$. Cada uma das varreduras indicadas acima é realizada pela rotina Computar_RTF (listada no Algoritmo 1), cujos parâmetros são: dk valendo 1 ou -1 que indica o sentido de percurso da lista (Ψ_k^n), para a frente ou para trás, empregado na varredura; e Min_Psi_n o valor limite de ψ_n a ser atingido na varredura. Assim, a primeira das varreduras acima é executada por:

$$Computar\_RTF(-1, \psi_n(0)),$$

e a segunda varredura por:

$$Computar\_RTF\left(1, -(n-1)\left[\arcsin\left(\frac{1}{\rho}\right) - \pi\right]\right).$$

Antes de fazer a apresentação da rotina Computar_RTF é preciso especificar mais precisamente o processo de transformação da luz dentro da gota.

Antes de tudo é preciso expressar os fluxos locais de entrada e saída (dF_{in} e dF_n) em função das intensidades da luz incidente (I_{in}) e da de saída (I_n), respectivamente. A expressão desses fluxos é dada pelas equações:

$$dF_{in} = I_{in} \cos\gamma d\omega_{in} = I_{in} \cos\gamma \sin\gamma d\gamma d\phi, \tag{5.8}$$

$$dF_n = I_n d\omega_n = I_n \sin\theta_n d\theta_n d\phi. \tag{5.9}$$

Por outro lado, se a luz é submetida à k refrações e l reflexões internas com $k + l = n$, os fluxos dF_{in} e dF_n são relacionados por:

$$dF_n = [T(\gamma)]^k [R(\gamma)]^l dF_{in}. \tag{5.10}$$

Certamente os únicos valores possíveis são $k = 0$, $l = 1$, se $n = 1$ e $k = 2$, $l = n - 2$, se $n > 1$. Combinando, agora, (5.8), (5.9) e (5.10), e definindo θ_n^{-1} como o mapeamento inverso da função que leva γ em θ_n por intermédio da equação (5.7), podemos escrever que:

$$\frac{I_n(\overline{\theta})}{I_{in}} d\theta_n(\overline{\gamma}) \sin(\overline{\theta}) = \sum_{\overline{\gamma} \in \Theta_n^{-1}} \mu(\overline{\gamma}) d\gamma, \tag{5.11}$$

onde $\mu(\overline{\gamma}) = T(\overline{\gamma})^2 R(\overline{\gamma})^{n-2} sin(\overline{\gamma}) \cos(\overline{\gamma})$ se $n \geq 2$.

Se $n = 1$, $\mu(\overline{\gamma}) = R(\overline{\gamma}))sin(\overline{\gamma}) \cos(\overline{\gamma})$ e, como neste caso, $\Theta_1(\overline{\gamma}) = 2 \cdot \overline{\gamma}$ resulta que:

$$\frac{I_n(\overline{\theta})}{I_{in}} = (1/4) R(\overline{\theta}/2). \tag{5.12}$$

Uma aproximação discreta da equação (5.11) é dada por:

$$\zeta_m^n \triangleq \frac{I_n(\alpha_m)}{I_{in}} \Delta \sin(\overline{\alpha_m}) \cong \sum_{k | (J_k \cap \Theta_n^{-1}(J_m)) \neq \emptyset} \mu(\alpha_k) \cdot \text{extensão}(J_k \cap \Theta_n^{-1}(J_m)) \tag{5.13}$$

Ou, escrito de forma equivalente:

$$\frac{I_n(\alpha_m)}{I_{in}} = \frac{\zeta_m^n}{\Delta \sin(\alpha_m)} \triangleq D_m^n. \tag{5.14}$$

Observe que $sin(\alpha_m) > 0 \, \forall m$ uma vez que $\alpha_m \geq \Delta/2$. Além disso chamamos atenção para os seguintes fatos:

i) A função simples $F : (\alpha \in J_m \rightarrow D_m^n)$ converge, embora não uniformemente, para $\frac{I_n(\alpha)}{I_n} \, \forall \alpha \in [0, 2\pi]$.

ii) D_m^n pode alternativamente ser determinado pela expressão:

$$\sum_{\overline{\gamma} \in \Theta^{-1}(\alpha_m)} \mu(\overline{\gamma}) \frac{1}{\frac{d\theta_n}{d\gamma}}, \tag{5.15}$$

como proposto em [6]. Entretanto, como $\frac{1}{\frac{d\theta_n}{d\gamma}}$ pode tomar um valor infinito, estimativas impróprias do valor médio de $\frac{I_n}{I_{in}}$ no intervalo J_m podem ser produzidas.

Agora já temos todos os elementos para explicitar a rotina Computar_RTF. No Algoritmo 1 dado a seguir: $\frac{\text{extensão}(J_k \cap \Theta_n^{-1}(J_m))}{\Delta}$ é dado por L_{1-p} enquanto D_m^n armazena, ao final, o termo central das igualdades dadas em (5.14).

Finalmente, dado que $I_{out}(\alpha_m, \phi) = \sum_{n=1}^{\infty} I_{out}(\alpha_m)\ \forall \phi$, temos:

$$D(\alpha_m, \phi) = \frac{I_n(\alpha_m)}{I_{in}} \cong \sum_{n=1}^{n_{max}} D_m^n \triangleq D_m\ \forall \phi. \tag{5.16}$$

De acordo com [6] fazendo $n_{max} = 9$, o erro percentual resultante do corte da energia proveniente de trajetórias com um número de reflexões internas maior que n_{max} se reduz a 0.2%. Por isso neste trabalho adotamos esse limite.

Algorithm 1 : Computar_RTF(int dk, int Min_Psi_n)

para cada $i \in (0, 1)$ **faça**
  $k \leftarrow Mq$
  para $j = \lfloor \frac{\Theta_n(m_n)}{2\pi} \rfloor$ ate $j = \lceil \frac{Min\_Psi\_n}{2\pi} \rceil$ $j = j - 1$ **faça**
    $m \leftarrow 2M$
    $p \leftarrow 0$
    repita
      enquanto $(\Psi_k) > (\sigma_m^i - 2j\pi))$ **faça**
        $k+ = dk$
        $L_p \leftarrow \frac{(\sigma_m^i - 2j\pi) - \Psi_{k+dk}}{\Psi_k - \Psi_{k+dk}}$
        $m - -$
        $p \leftarrow 1 - p$
        enquanto $(\Psi_k) < (\sigma_m^i - 2j\pi))$ **faça**
          $dL \leftarrow \frac{(\sigma_m^i - 2j\pi) - \Psi_{k+dk}}{\Psi_k - \Psi_{k+dk}}$
          $L_{1-p}\ - = dL$
          $D_{m+1}^n\ + = \frac{L_{1-p} * \mu(\alpha_k)}{\sin(\alpha_k)}$
          $L_p\ + = dL$
          $m - -$
          $p \leftarrow 1 - p$
        fim enquanto
        $D_{m+1}^n\ + = \frac{L_{1-p} * \mu(\alpha_k)}{\sin(\alpha_k)}$
      fim enquanto
    até $m \geqslant 0$
  fim para
fim para

Os valores das RTFs indicados em $D_m, m = 0, \ldots 2M-1$ se referem à direção de entrada $(0,0)$. Para determinar RTFs relativas à direções de entrada genéricas comecemos por construir uma amostra discreta dessas direções. Para isso vamos discretizar o espaço do ângulo ϕ, isto é $[0, 2\pi]$, em Q amostras ϕ_q e restringir o cálculo das RTFs às direções obtidas pelas amostragens feitas para θ e ϕ tanto para a entrada como para a saída da luz da gota. Nesse contexto, represente por:

$$D_{(r,s,m,q)} = D((\alpha_r, \phi_s), (\alpha_m, \phi_q))$$

o valor da RTF quando a entrada tem a direção dada por (α_r, ϕ_s) e a saída por (α_m, ϕ_q). Seja ainda $R_{r,s}$ uma rotação que leva a direção $(0,0)$ na dada por (α_r, ϕ_s). Qualquer rotação com essa propriedade serve, por exemplo, aquela cujo eixo é dado pelo produto vetorial dessas direções. Devido ao modelo esférico da gota que estamos adotando as RTFs relativas (α_r, ϕ_s) são obtidas aplicando ao mapeamento das RTFs relativas a $(0,0)$ a rotação inversa $R_{r,s}^{-1}$. Como essa rotação produz valores que não pertencem a amostragem de θ e ϕ o valor final da RTF $(D_{(r,s,m,q)})$ é obtido por interpolação linear. A sequência abaixo indica, precisamente, o processo de determinação de $D_{(r,s,m,q)}$:

$$\begin{aligned} D_{(r,s,m,q)} &= D((\alpha_r, \phi_s), (\alpha_m, \phi_q)) = D(R_{r,s}(0,0), (\alpha_m, \phi_q)) = \\ &= D((0,0), (R_{r,s}^{-1}(\alpha_m, \phi_q))) = \\ &= D(\alpha'_m, \phi'_q) \end{aligned}$$

Se α'_m se escreve $m'\Delta + r'$ com m' inteiro e $r' < \Delta$ então se pode usar a aproximação:

$$D(\alpha'_m, \phi'_q) \cong (1 - (r'/\Delta))D_{m'} + (r'/\Delta)D_{m'+1}.$$

Com isso, extendemos a expressão das RTFs para qualquer direção de entrada a partir da calculada para $(0,0)$. O processo inteiro até produzir $D_{(r,s,m,q)}$ é linear, o que torna ainda mais vantajoso o emprego de um modelo fechado para a interação luz-gota do que usar dispendiosos métodos de *ray tracing*.

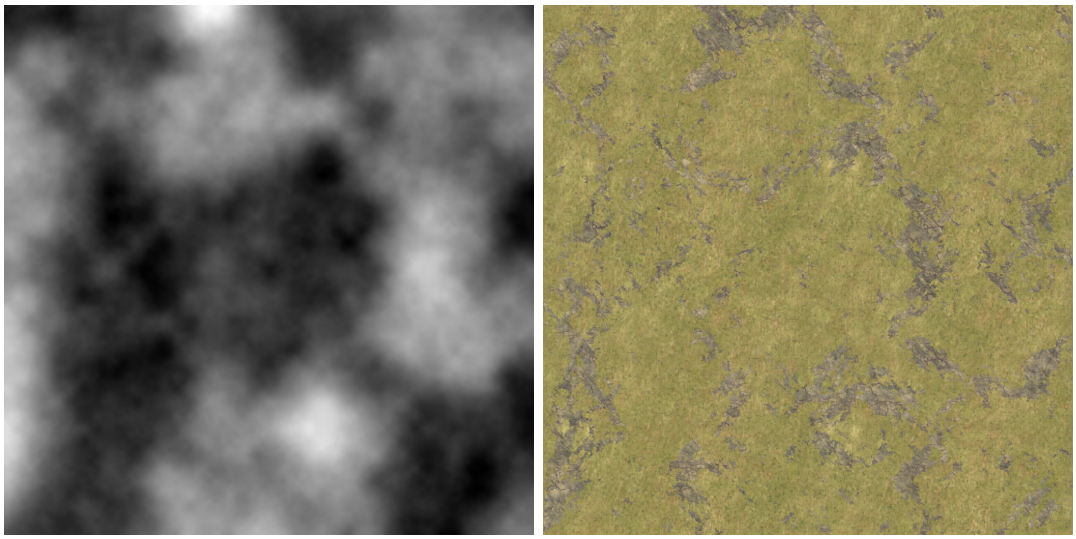
5.2 Modelo Digital de Terreno

Segundo a definição de [41], um Modelo Digital de Terreno (MDT) é uma representação matemática e computacional de um determinado fenômeno espacialmente

distribuído numa região limitada da superfície terrestre. Um MDT será caracterizado por: um conjunto de amostras, que define a geometria do terreno; uma estrutura de dados, que permite definir relações topológicas/proximidade entre as amostras; e por fim um interpolador, responsável pelo processo de reconstrução da superfície do terreno.

Terrenos são entidades contínuas e irregulares. Por essa razão, pode ser muito difícil encontrar uma representação matemática explícita que se ajuste ao terreno. Assim, MDTs normalmente descrevem um terreno construindo uma aproximação, contínua ou linear por partes, da sua superfície real. Para a construção dessa aproximação, primeiramente um processo de amostragem deve ser aplicado, resultando em uma coleção de pontos que tipicamente armazenam a elevação para cada ponto amostral.

O MDT utilizado nesse trabalho é do tipo DEM (*Digital Elevation Model*), ou Modelo Digital de Elevação. Nesse tipo de MDT uma estrutura de grade regular mantém as altitudes (coordenadas z) associadas aos pares (x, y) definidos implicitamente pela grade, ou seja, é um mapa de alturas. O processo de amostragem se faz de forma uniformemente espaçada tomando-se um ponto por célula da grade.



(a) (b)

Figura 5.5: Mapas de terreno gerados pelo aplicativo L3DT: (a) mapa 8 bits de alturas. (b) mapa 32 bits de textura.

O *software* L3DT [42] é utilizado para a geração do mapa de alturas, bem como para gerar um mapa de textura do terreno (Figura 5.5). Esse *software* permite uma geração procedural de pontos amostrais, a partir de um conjunto de parâmetros fornecidos pelo usuário, por algoritmos *perlin noise* e fractais. Após

carregar a imagem com o mapa de alturas nosso sistema atribui, para cada nó (x, y) da grade, um valor de altura mapeado através do tom de cinza do *pixel* (i, j). Em seguida uma triangulação regular ligando esses pontos (Figura 5.6) é construída.

Por fim a coordenada de textura de cada vértice da triangulação é extraída do mapa de texturas. Com o intuito de inserir detalhes sem custos de armazenamento, essa textura está em uma escala superamostrada. Essa super amostragem é realizada em uma etapa de pré-processamento, pelo *software* utilizado para a geração dos modelos, e não gera custos adicionais de computação. Essa versão suavizada da textura é aplicada no modo *Modulate* do OpenGL, que possibilita a modulação entre a cor da textura e a fonte de iluminação ativa. Dessa forma a cor final do vértice não depende só da cor vinda da textura, ela passa a ser alterada em tempo real pela fonte de iluminação.

5.3 Simulação de Fluxo Superficial com SPH

A simulação do fluxo superficial proposta nesse trabalho utiliza o mesmo método para animação de fluidos, introduzido na comunidade de computação gráfica por Muller [27]. As diferenças básicas são a geração de partículas devido à precipitação e a evolução sobre uma geometria complexa, a geometria de um modelo de terreno. Outro diferencial é a utilização de um método alternativo para a extração da geometria da superfície livre do fluido [13]. Esse método constrói uma representação explícita da superfície que é bastante apropriada para a representação de fluxos superficiais para águas rasas. O método também é mais eficiente que o tradicional *marching cubes*.

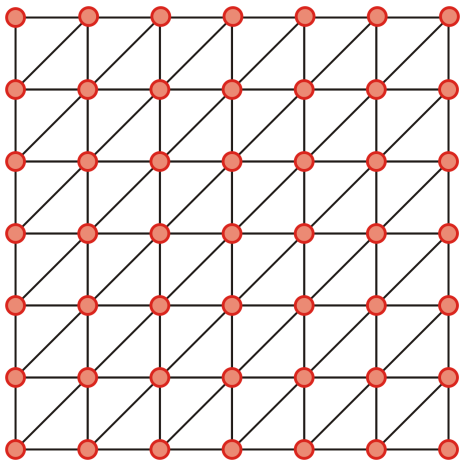


Figura 5.6: Triangulação regular construída a partir do mapa de alturas. Cada *pixel* do mapa de alturas passa a ser representado por um vértice na triangulação.

5.3.1 Dinâmica do Fluido

A dinâmica do fluido se dá através da contribuição da dinâmica de cada partícula individual. Uma partícula é atualizada, para cada instante de tempo, de acordo com o Algoritmo 2 descrito abaixo.

Algorithm 2 : Fluxo Superficial

Computar Densidade:
    Acumular contribuições das partículas vizinhas
    Aplicar equação da densidade
Computar Forças:
    Força de pressão
    Força de viscosidade
    Força de tensão superficial
    Aplicar Gravidade
Atualizar posição da partícula:
    Computar a aceleração
    Atualizar velocidade e posição
Computar Colisão com Terreno:
    Refletir velocidade e posição

Densidade

O primeiro passo para a computação da equação (3.10) é a atualização da densidade de cada partícula. Cada partícula representa uma porção finita de fluido com volume dado por:

$$V_i = \frac{m_i}{\rho_i}. \tag{5.17}$$

Uma vez que a massa de cada partícula é constante, a densidade varia, tornado o fluido compressível. Assim a densidade deve ser atualizada a cada instante da simulação. A densidade ρ na posição r é computada como a soma da massa de cada partícula vizinha suavizada pela função *kernel*:

$$\rho_s(r) = \sum_j m_j W(\mathbf{r}_{ij}, h). \tag{5.18}$$

Uma vez atualizada a densidade, nos passos seguintes serão computadas as forças responsáveis pela aceleração, e consequentemente o movimento, de cada partícula. As moléculas em um fluido estão sujeitas à forças atrativas e repulsivas das moléculas vizinhas. Essas forças são a pressão, a viscosidade e a tensão superficial. O SPH modela cada uma dessas forças aplicando os operadores diferencias

(gradiente e laplaciano) nas funções *kernel*.

Pressão

Antes de computar a força de pressão é preciso computar o valor da pressão na posição de cada partícula. A pressão pode ser computada por uma versão modificada da equação de estado dos gases ideais, sugerida por Desbrun [10]:

$$p = k(\rho - \rho_0), \tag{5.19}$$

onde k é uma constante do gás dependente da temperatura, ρ_0 é a densidade da partícula em repouso, e ρ é a densidade que acabou de ser computada. Esse modelo introduz algumas limitações, podendo tornar o fluido compressível, mas é aceitável para um modelo simples de fluidos.

Quando uma força é aplicada sobre um fluido, ela é propagada por suas moléculas, as mais próximas da região onde a força é aplicada empurram suas vizinhas, gerando pressão. Pressão é força por unidade de área, onde a força resultante terá um sentido que aponta de regiões de alta pressão para regiões de baixa pressão. No SPH a força de pressão é computada utilizando a média das pressões das partículas vizinhas. Essa formulação tem como objetivo a produção de forças simétricas quando temos apenas 2 partículas. Caso não houvesse simetria de forças para o caso de 2 partículas a formulação estaria incoerente com a terceira lei de Newton.

$$\vec{f}_i^{\text{press}} = - \bigtriangledown p = - \sum_j m_j \frac{p_i + p_j}{2\rho_j} \bigtriangledown W(\mathbf{r}_{ij}, h). \tag{5.20}$$

Viscosidade

A outra força que atua sobre o movimento dos fluidos é a viscosidade. A viscosidade pode ser interpretada como uma medida de resistência do fluido sobre o fluxo da sua própria velocidade. Essa resistência resulta em difusão do momento e do campo velocidade dentro do fluido. Ela tende a fazer com que uma porção de fluido se mova com velocidade igual à média das velocidades encontradas na vizinhança. Isso ocorre pelo fato da energia cinética ser transformada em calor devido ao atrito das moléculas do fluido. No SPH a força de viscosidade é computada utilizando a diferença das velocidades das partículas vizinhas. Essa formulação, como na computação da força de pressão, também visa a simetria de forças.

$$\vec{f}_i^{visc} = \mu \bigtriangledown^2 \vec{u} = \mu \sum_j m_j \frac{\vec{u}_j - \vec{u}_i}{\rho_j} \bigtriangledown^2 W(\mathbf{r}_{ij}, h). \tag{5.21}$$

Gravidade

A gravidade é considerada uma força externa e é aplicada diretamente nas partículas, sem a utilização de nenhum kernel do SPH.

$$\vec{f}_i^{\text{grav}} = \rho_i g. \tag{5.22}$$

Tensão Superficial

Além das forças de pressão e viscosidade, para um resultado fisicamente correto na computação da superfície livre, é preciso computar a força de tensão superficial. Diferente das forças de pressão e viscosidade, que atuam em todas as partículas do fluido, a força de tensão superficial é aplicada apenas nas partículas da superfície livre do fluido. No interior do fluido, as forças repulsivas e atrativas (pressão e viscosidade) são iguais em todas as direções, tendendo a um estado de equilíbrio. Já na superfície do fluido essa simetria não ocorre, fazendo com que essas partículas sejam "empurradas" para fora pelas partículas internas. Esse problema ocorre pelo fato do gradiente da densidade ser alto na fronteira do fluido (Figura 5.7).

As forças de tensão superficial agem na direção normal da superfície, no sentido do interior do fluido, minimizando a curvatura da superfície. O objetivo dessa força é "puxar", de forma suave, as partículas para a superfície do fluido.

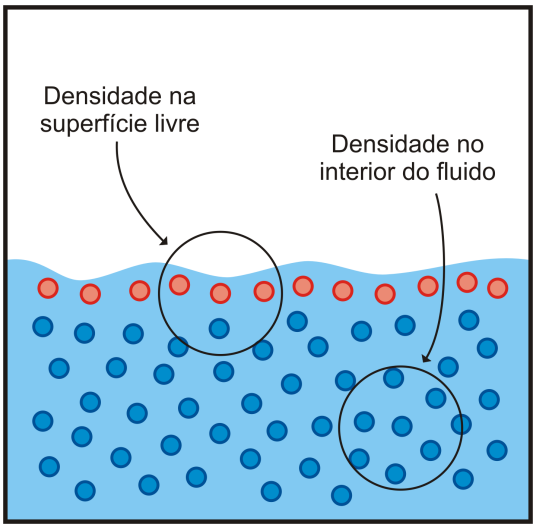


Figura 5.7: Densidade computada para as partículas próximas à superfície livre do fluido.

Nesse trabalho foi utilizado o método proposto em [27], com algumas alterações, para a computação da força de tensão superficial. A diferença básica entre o método

proposto e o que foi utilizado nesse trabalho está na identificação das partículas da superfície. Em [27] a superfície do fluido é encontrada usando um campo quantitativo adicional c_S, chamado pelo autor de campo de cor. Esse atributo possui valor 1 na localização da partícula e 0 em qualquer outra localização. A versão suavizada do campo de cor é dada por:

$$c_S(r) = \sum_j m_j \frac{1}{\rho_j} W(\mathbf{r}_{ij}, h). \tag{5.23}$$

O gradiente do campo de cor fornece o vetor normal da superfície apontando para dentro do fluido: $\vec{n} = \bigtriangledown c_S$. Então um simples teste de limiar identifica a partícula de superfície: $|\vec{n}(r_i)| > l$, onde l é um limite inferior. Alguns testes foram realizados, e esse algoritmo apresentou problemas na identificação das partículas de algumas regiões do fluido. Então optamos por realizar esse processo durante a etapa de reconstrução da superfície livre do fluido.

Como será descrito em detalhes na seção 5.3.2, esse método faz um subdivisão regular do domínio, onde estão as projeções das partículas, identificando a partícula de maior altura encontrada em cada célula como partícula de superfície. Esse processo não apresentou problemas em nenhuma região do fluido, além de não adicionar custos computacionais, uma vez que ele já era realizado para a reconstrução da superfície. Uma vez identificadas as partículas da superfície, o restante do processo segue a proposta de [27].

A curvatura da superfície também será utilizada para a computação da força de tensão. Ela é medida pela divergência do vetor normal, ou seja, o laplaciano do campo de cor:

$$k = \frac{-\bigtriangledown^2 c_S}{|\vec{n}|}. \tag{5.24}$$

Então a força de tensão superficial é dada por:

$$\vec{f}^{tensao} = -\sigma\kappa \frac{\vec{n}}{||\vec{n}||}, \tag{5.25}$$

onde σ é o coeficiente de tensão (dependente dos dois fluidos), $\vec{n}$ e κ são o vetor normal e a curvatura da partícula de superfície respectivamente. Isto é, a força de tensão vai ser dada pelo produto entre o coeficiente de tensão e a curvatura, enquanto o vetor unitário da normal fornece a direção, apontando para o interior do fluido.

Integração Temporal

A aceleração das partículas é computada utilizando a equação abaixo. Ela é obtida substituindo as expressões (5.20)-(5.22) e (5.25) nas equações de Navier-Stokes:

$$\frac{d\vec{u}_i}{dt} = \frac{Q_i^t}{\rho_i}, \tag{5.26}$$

onde $Q_i^t = \vec{f}_i^{press} + \vec{f}_i^{visc} + \vec{f}_i^{grav} + \vec{f}_i^{tensao}$.

A posição de cada partícula é atualizada através do esquema de *Leapfrog*, como o utilizado em [11]. Nesse esquema a velocidade é atualizada utilizando a aceleração computada através da equação (5.26).

$$\vec{v}_i^{t+\delta t} = \vec{v}_i^t + \frac{\delta t}{2} Q_i^t, \tag{5.27}$$

Então essa velocidade é usada para computar a nova a posição:

$$r_i^{t+\delta t} = r_i^t + (\delta t) \cdot \vec{v}_i^{t+\delta t}, \tag{5.28}$$

onde δt é o passo de tempo.

A colisão de cada partícula de SPH com a superfície do terreno é computada e tratada (em 2 etapas) sem aproximações. O primeiro passo é identificar em qual face da triangulação, que representa o terreno, ocorreu a colisão. As coordenadas baricêntricas dessa face são utilizadas para realizar a interpolação do vetor normal no ponto exato da colisão. Em seguida, o vetor normal interpolado é utilizado para refletir o vetor velocidade da partícula que colidiu, como mostra a Figura 5.8. No modelo utilizado, o vetor velocidade também sofre uma espécie de amortecimento, expressado pela multiplicação por um escalar entre 0 e 1. A posição também é refletida, na vertical, para cima do terreno. Apesar de simples, essa solução apresentou-se satisfatória para o problema em estudo, produzindo efeitos visuais plausiveis.

5.3.2 Extração da Superfície Livre do Fluido

Uma vez que no SPH não há uma malha de suporte para a solução das equações diferenciais, é necessário a aplicação de alguma técnica para reconstrução da geometria da superfície livre do fluido. Como já mencionado anteriormente, nesse

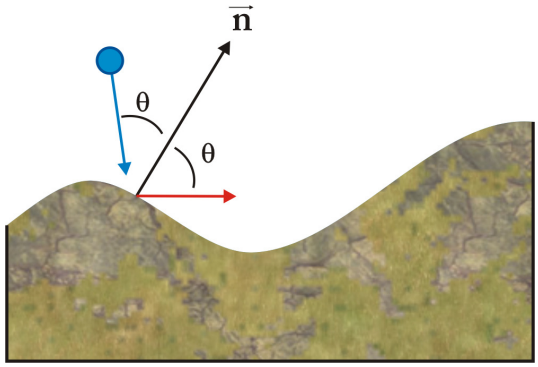


Figura 5.8: Quando um partícula colide com o terreno o seu vetor velocidade é refletido pelo vetor normal do terreno.

trabalho foi implementado o método proposto em [13] para extração e *rendering* da superfície do fluido (Figura 5.9). A ideia é representar essa superfície utilizando uma estrutura chamada pelo autor de *carpet* ou tapete.

O *carpet* pode ser visto como uma malha regular que "cobre" todo o domínio onde estão as partículas do fluido. Essa estrutura consiste basicamente de duas sub-estruturas: uma malha (virtual) $n \times n$ e uma *quadtree*. A *quadtree* tem o objetivo de reduzir a complexidade da varredura da malha. Cada vértice (x, y, z) do *carpet* armazena, na coordenada z, a maior altura encontrada nas partículas cujas projeções estão naquela região. Nas partes onde não há fluido o vértice tem altura definida com altura menor que a altura do terreno para o mesmo ponto (x,y), sendo eficientemente excluído do processo de *rendering* através de uma *quadtree*.

Construção

Essa malha é inicializada como um plano e posicionada abaixo de qualquer altura do terreno. A *quadtree* armazena em cada um de seus nós a altura mínima do terreno para aquele domínio, a altura da superfície livre do fluido para aquele domínio e um valor de velocidade vertical. As partículas do SPH são armazenadas nas folhas da *quadtree*. Cada folha armazena a maior altura encontrada nas partículas cujas projeções estão naquela região, com isso a estrutura é "levantada" para o nível das partículas de superfície. O passo final da etapa de construção do *Carpet* é a interpolação das alturas para os nós internos da *quadtree*. Esse processo é realizado percorrendo a *quadtree* no sentido de baixo para cima, interpolando a altura de cada nó "pai" utilizando a média das alturas de seus nós "filho".

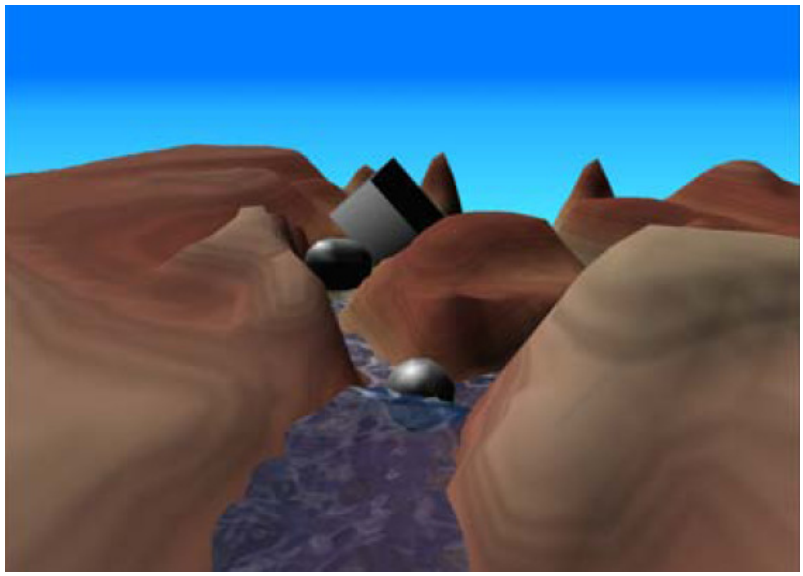


Figura 5.9: Superfície do fluido reconstruída através do *carpet* interagindo com corpos rígidos. (Imagem retirada de [13].)

Rendering

A superfície do fluido é renderizada percorrendo a *quadtree*, de baixo para cima, de forma recursiva. Nas regiões do terreno onde não existe fluido, a recursão é abortada nos primeiros níveis. O recursão é abortada, e nada é feito, toda vez que uma altura menor que a altura do terreno é encontrada em um nó interno da *quadtree*. Por fim, a velocidade vertical é acelerada pela gravidade e o *carpet* "cai" suavemente.

Enquanto existe uma partícula nesse nó, a velocidade vertical do nó é zero, quando a partícula deixa o nó a velocidade é acelerada até chegar outra partícula ou a altura for menor que a altura do terreno. Isso faz o *carpet* descer lentamente nas bordas, produzindo o efeito semelhante a um rastro de fluido deixado sobre a superfície do terreno. Nesse trabalho, efeitos visuais de *environment mapping*, bem como a lei de refração/reflexão de Fresnel [43], são utilizados no *rendering* da superfície.

5.4 Animação dos Respingos

A animação da chuva envolve efeitos secundários que ocorrem em uma escala menor, como é o caso dos respingos nas superfícies. Uma animação sem a presença desses efeitos se torna pobre visualmente e pouco convincente. Em computação gráfica, esses efeitos são animados geralmente através de formulações baseadas em texturas [2] ou partículas [4]. Existem também formulações mais robustas que

utilizam as equações de Navier-Stokes [44].

Nesse trabalho optamos por uma proposta que visa o desempenho, mesmo que sacrificando um pouco o realismo visual. Contudo, devido a escala desses detalhes, os impactos no aspecto visual da cena gerada são praticamente desprezíveis. Nossa proposta é baseada no trabalho de [2] onde a evolução do respingo no tempo é representada por uma sequência de 6 texturas, como mostra a Figura 5.10.

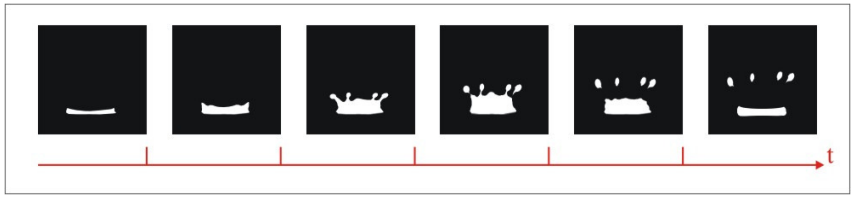


Figura 5.10: Sequência de imagens utilizada para o *rendering* dos respingos nas superfícies do fluido e do terreno.

Na realidade existem 3 sequências de 6 texturas cada uma, totalizando 18 texturas. A sequência ilustrada na Figura 5.10 é utilizada para regiões planas e de baixa declividade. Quando a declividade do ponto de inserção aumenta, é necessário que o respingo tenha um aspecto direcional, dando a impressão de uma pequena cachoeira em regiões de alta declividade. Dessa forma, de acordo com o range da declividade do ponto de inserção do respingo, uma das 3 sequências é escolhida. Essas texturas também são rotacionadas de acordo com o ângulo entre a componente z, do vetor velocidade da gota, e o vetor normal do ponto de colisão. Os respingos são uniformemente distribuídos, ao longo da projeção (sobre a superfície do *carpet*) da área de precipitação, e inseridos na cena através de *blending*.

5.5 Integração dos Modelos

A integração do sistema de partículas utilizado para animar a chuva com a animação de fluxo superficial (SPH) se resume basicamente na transformação das partículas da chuva, ao passo que estas atingem o solo, em partículas do SPH. Outro aspecto é a comunicação entre GPU e CPU, uma vez que o sistema da chuva foi implementado em GPU e o SPH roda em CPU. Após o *setup* das condições iniciais, o processo de simulação é inicializado. Na Figura 5.11 é possível observar a interação entre os modelos para um instante de tempo.

Inicialmente a CPU precisa enviar para a GPU uma matriz com as alturas (mapa de alturas) da superfície livre do fluido, onde não há fluido é armazenada a altura

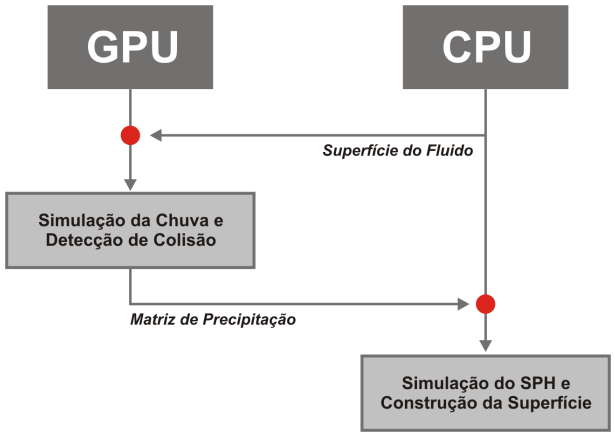


Figura 5.11: Esquema de interação entre modelos para um instante de simulação.

do terreno no ponto. A partícula de chuva irá evoluir até encontrar essa superfície. Esse teste não é realizado no ponto exato da colisão, por razões de desempenho, ele é aproximado para o canto inferior esquerdo da célula:

$$z \leq \varphi\left(\lfloor x \rfloor, \lfloor y \rfloor, t\right). \tag{5.29}$$

Uma vez que a matriz de alturas possui uma resolução alta, essa aproximação na detecção da colisão não introduz erros significativos. Quando a colisão ocorre, a partícula é excluída do modelo da chuva, e a matriz de precipitação para o fluxo superficial é atualizada na posição da colisão. Essa matriz é um campo de contadores, do tipo inteiro, que armazena quantas partículas colidiram no ponto $\varphi(i, j)$. Ela é atualizada de acordo com a expressão:

$$cont\left(\lfloor x \rfloor, \lfloor y \rfloor, t\right) = cont\left(\lfloor x \rfloor, \lfloor y \rfloor, t\right) + 1. \tag{5.30}$$

Após a computação das colisões e a atualização da matriz de precipitação $cont(i, j, t)$, a GPU precisa retornar essa matriz para o modelo de fluxo superficial. A CPU irá então realizar a inserção das novas partículas no modelo SPH utilizando as informações dessa matriz. Cada nova partícula é inserida na posição da colisão $(\varphi(i, j))$ de acordo com a expressão:

$$N(i,j,t) = \begin{cases} 0 & para \quad cont(i,j,t) < S \\ 1 & para \quad cont(i,j,t) \geq S \end{cases} \tag{5.31}$$

onde $N(i, j, t)$ é o número de novas partículas de SPH inseridas no tempo t, e S é o fator de escala da precipitação, que faz uma relação entre partículas no sistema da chuva e partículas no modelo SPH. Ele irá indicar quantas partículas de chuva

precisam colidir no ponto $\varphi(i, j)$, para a criação de uma nova partícula do SPH. Quanto maior seu valor, menor o volume do fluxo superficial produzido pela chuva.

Após a inserção da partícula de SPH na posição $\varphi(i, j)$, o valor armazenado na posição (i, j) da matriz de precipitação é zerado e a contagem das colisões recomeça. Em seguida o SPH segue seu processo de atualização normalmente. Na Figura 5.11 a relação entre a atualização do sistema da chuva e do SPH é um pra um. No entanto, como o SPH é um método baseado em equações diferenciais parciais (computacionalmente custoso) e está implementado em CPU, esse esquema pode ser alterado; por exemplo, não é necessário atualizar o SPH a cada instante de simulação de chuva. Podemos fazer isso após um numero $m > 1$ de instantes de simulação da chuva. Através dessa heurística é possível conseguir um aumento de performance considerável. Empiricamente concluímos que um *tradeoff* considerável entre performance e qualidade visual para o valor m é 3.

Capítulo 6

Resultados e Discussões

Nesta seção serão descritos alguns experimentos realizados com o protótipo desenvolvido nesse trabalho. Nos experimentos demonstramos o comportamento do sistema em diferentes modelos de terreno e *environment maps*. A intenção é destacar aspectos que podem ser utilizados para aplicações de tempo real, como jogos eletrônicos, aplicações de realidade virtual e simuladores. Ressaltamos que o *framework* proposto também pode ter utilidade na área da engenharia. Nesse contexto ele poderá ser aplicado como ferramenta auxiliar no estudo de recursos hídricos e controle de alagamentos. Áreas de possíveis inundações, formação de rios, além de outros impactos podem ser facilmente identificadas. Também discutiremos alguns resultados que ilustram a utilidade do protótipo para realizar a inserção da chuva em vídeos. Esse tipo de aplicativo pode ter aplicação direta na produção de filmes, seriados, comerciais, dentre outros.

O código fonte foi desenvolvido utilizando as linguagens de programação *C/C++*. A linguagem CUDA da NVidia [15] também foi utilizada para desenvolvimento das rotinas responsáveis pela atualização do sistema da chuva. A visualização foi desenvolvida em *OpenGL* [45], com todos os *shaders* escritos em *OpenGL Shading Language* [46].

Os experimentos foram realizados em um computador *Intel Core i7* 3.33 GHz, com 8 GBytes de memória *ram* e uma placa gráfica *NVidia GeForce* GTX 480, rodando o sistema operacional Ubuntu Linux.

6.1 *Rendering* em uma Cena Virtual

Nos experimentos descritos nessa seção, a chuva é animada em uma cena contendo um MDT (Modelo Digital de Terreno) e um *environment map*. Os testes foram realizados sob diferentes topografias e mapas de ambiente. A malha com o mapa de alturas representando a superfície do fluido (o *carpet*) possui resolução de 1024×1024. Essa resolução mostrou-se aceitável para o desempenho desejado, além de proporcionar resultados visuais bastante razoáveis. Os MDTs são representados por modelos digitais de elevação, cuja resolução é 512×512. Também foram utilizadas texturas na resolução de 2048×2048 com o intuito de inserir detalhes, a baixo custo computacional, na renderização do terreno. A atualização das posições das partículas do SPH e da chuva seguem as expressões (5.27)-(5.28) e (5.1)-(5.3) respectivamente, com $\delta t = 0.15$ e $\Delta t = 0.025$.

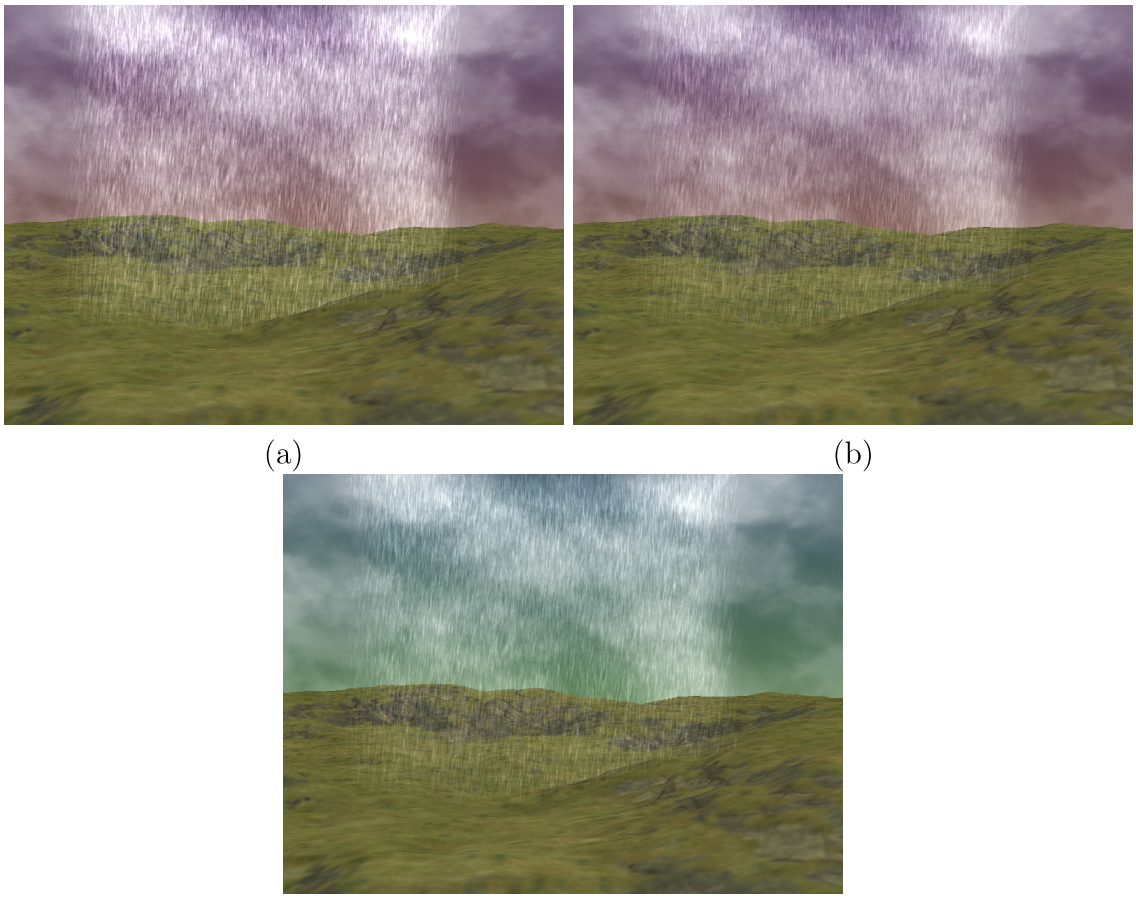
Nos testes realizados tentamos destacar algumas características do sistema que podem torná-lo atrativo tanto do ponto de vista da computação gráfica, quanto da engenharia. A seguir detalhamos essas características, ilustrando com imagens geradas pelo sistema.

1. *Rendering* da Chuva

O objetivo desses exemplos é ilustrar o realismo alcançado com o método de animação da chuva, destacando os efeitos que a mudança do *environment map* provocam no *rendering* da chuva. Para esses testes foi utilizada apenas a animação da chuva sobre o MDT, sem fluxo superficial ou respingos. A Figura 6.1 ilustra os resultados obtidos utilizando o método descrito na seção 5.1, onde é possível observar efeitos realísticos produzidos com o uso de *environment lightning*. Essa iluminação ambiente é computada em tempo de execução através da PRT (*Precomputed Radiance Transfer*). A chuva ilustrada na Figura 6.1(a) (produzida sem a iluminação do ambiente) possui um tom esbranquiçado pouco realista. A chuva ilustrada nas Figuras 6.1(b) e 6.1(c) possui aparência mais realista, pois foi renderizada utilizando a iluminação do ambiente. Nas Figuras 6.1(b) e 6.1(c) é possível observar que embora a chuva possua os mesmos parâmetros, a aparência final é bastante alterada devido ao *environment map*.

2. Escala de Precipitação

Nesses exemplos faremos uma análise visual dos efeitos provocados pela alteração do parâmetro escala de precipitação S na expressão (5.31). Aumentando esse parâmetro ocorre uma diminuição no número de partículas inseridas no modelo SPH, resultando em um menor volume de fluxo superficial. As Figuras 6.2(a,c,e) ilustram



(a) (b)

(c)

Figura 6.1: Efeitos da mudança da cor do *environment map* no *rendering* da chuva: (a) sem *environment lightning*, em (b) a chuva possui um tom avermelhado, acompanhando *environment map*, enquanto em (c) seu tom é esverdeado.

a configuração do fluido nos instantes 250, 1200 e 2600 da simulação quando foi utilizado $S = 2$, totalizando 21684 partículas de SPH, enquanto as Figuras 6.2(b,d,f) ilustram os mesmos instantes de simulação, porém utilizando $S = 4$ (10620 partículas de SPH).

Como esperado, nas Figuras 6.2(b,d,f) temos uma quantidade menor da água acumulada sobre o terreno. É possível observar similaridades entre os padrões de fluxo superficial nas Figuras 6.2(a,c,e) e 6.2(b,d,f), indicando que é possível variar esse parâmetro para diminuir a vazão do fluxo, porém sem deixar de reproduzir um efeito fisicamente plausível. Outro aspecto que também pode ser observado nessas figuras é a formação de acúmulo, ou lagos, elas ilustram uma região côncava do terreno bem como a água acumulada ao longo da simulação. Esse exemplo indica que esse tipo de simulação pode ser útil não só para aplicações de computação gráfica, mas também para estudos de recursos hídricos e controle de alagamento.

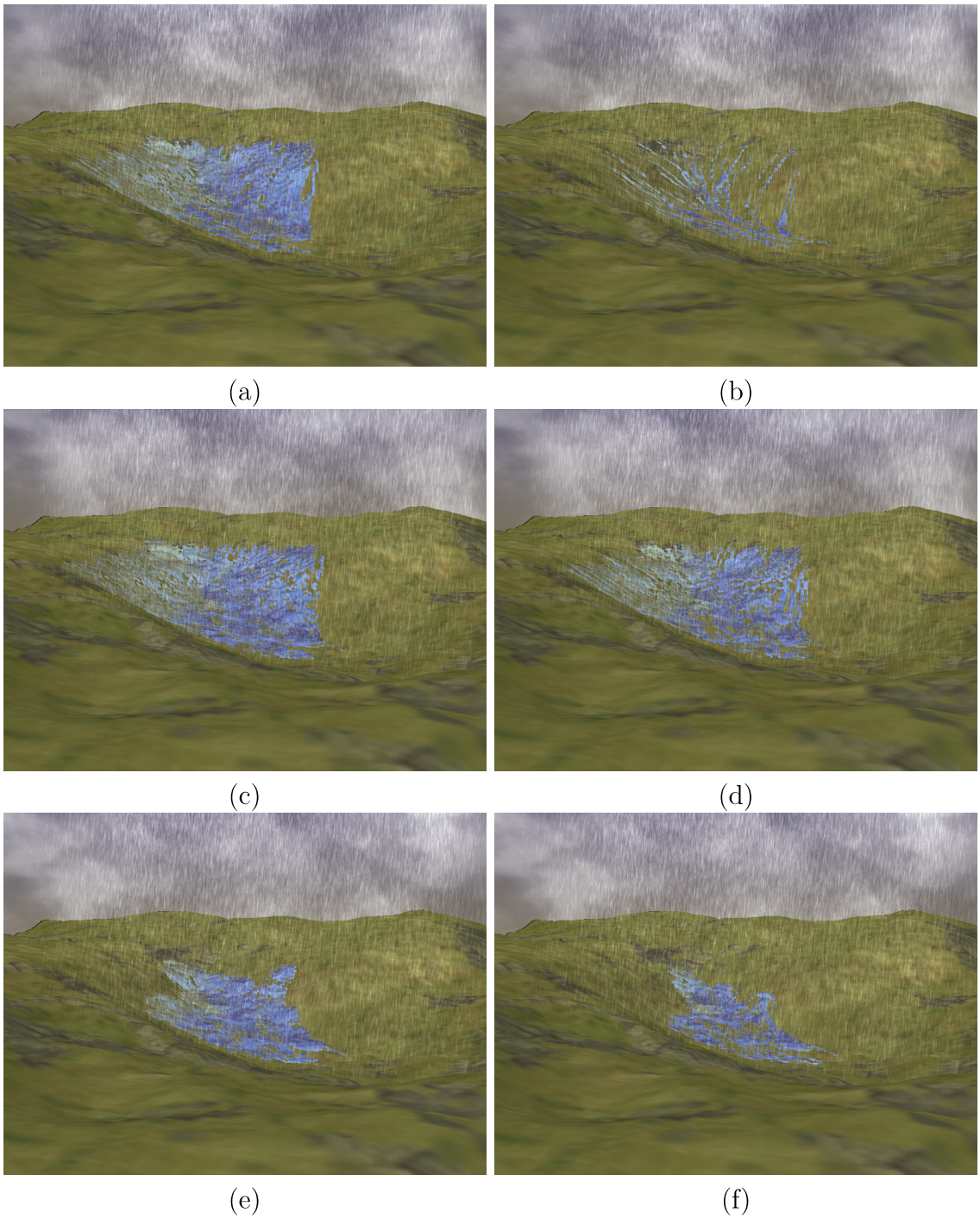


Figura 6.2: Alagamento de uma região côncava. (a,c,e) Água acumulada após 250, 1200 e 2600 instantes de simulação com $S = 2$. (b,d,f) Mesma configuração de (a,c,e), porém usando $S = 4$.

3. *Carpet*

O exemplo da Figura 6.3 demonstra as potencialidades do framework proposto para a simulação de rios, formação de trajetórias e acúmulo sobre uma topografia complexa. Nessa figura, temos uma fonte de chuva localizada no topo de uma montanha. O objetivo desses testes é ilustrar o rio formado na região de vale, além das trajetórias formadas pela água ao longo da descida da montanha. Nas Figuras 6.3(a,c,e) temos uma visualização do fluxo superficial utilizando apenas as partículas do SPH para os instantes 400, 1000 e 3800. Nas Figuras 6.3(b,d,f) observamos a visualização, para os mesmos instantes de tempo, utilizando a superfície reconstruída pelo método do *carpet*. Fica claro que o fluido segue uma trajetória que vai da parte mais montanhosa para a região de vale.

O *carpet* mostrou-se bastante realístico para a simulação de rios. Embora existam regiões do MDT com distribuições de partículas esparsas, a estrutura "cobre" toda a topografia, como acontece com um rio na natureza. As trajetórias deixadas pelas partículas ao descer a montanha também são outras características do método. Quando uma partícula deixa um nó do *carpet* ele "desce" suavemente até atingir o terreno, produzindo esse efeito. Efeitos visuais de *environment mapping*, bem como a lei de refração/reflexão de Fresnel, são utilizados no *rendering* da superfície.

4. Tensão Superficial

Agora serão demonstrados os resultados da aplicação da força de tensão superficial definida na expressão (5.25). As Figuras 6.4(a,c,e)-(b,d,f) foram geradas utilizando os mesmos parâmetros, para os instantes 400, 1000 e 2000, porém na primeira sequência não foi utilizada a força de tensão superficial. Na Figura 6.4(a) apesar de não haver aplicação de força de tensão a superfície se comporta bem, isso ocorre porque o número de partículas é baixo. Nas Figuras 6.4(c,e) observamos instabilidades na superfície livre, produzidas pelo aumento do gradiente da densidade nas partículas próximas à superfície. De fato, as partículas tendem a "escapar" do fluido, o que provoca os padrões observados em algumas regiões côncavas. Fica claro que a superfície livre ilustrada nas Figuras 6.4(b,d,f) (onde foi utilizada a força de tensão) é bem mais realística e suave que a ilustrada nas Figuras 6.4(c,e).

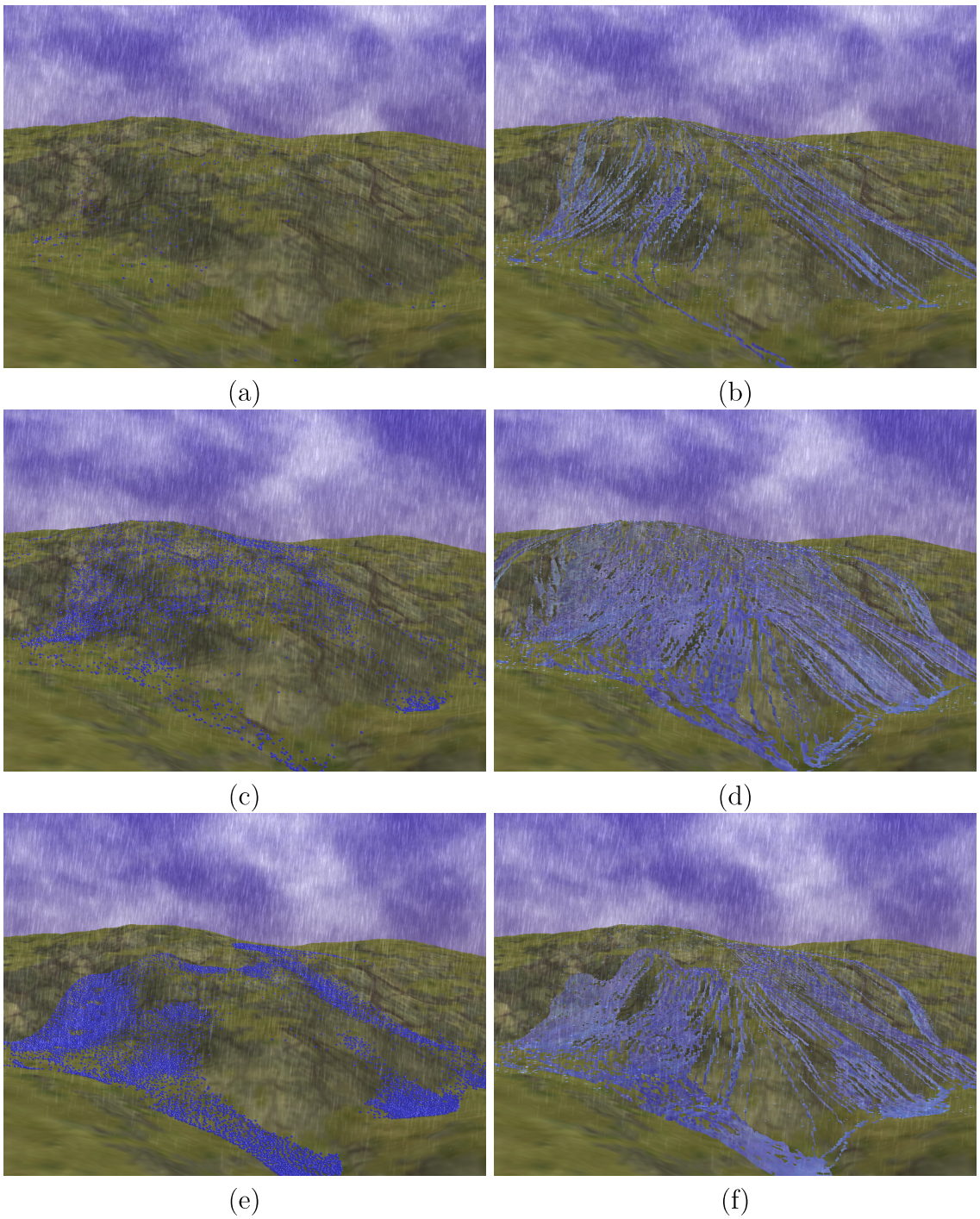


Figura 6.3: (a,c,e) Visualização do fluxo superficial utilizando apenas as partículas do SPH. (b,d,f) Visualização utilizando a superfície reconstruída pelo método do *carpet*.

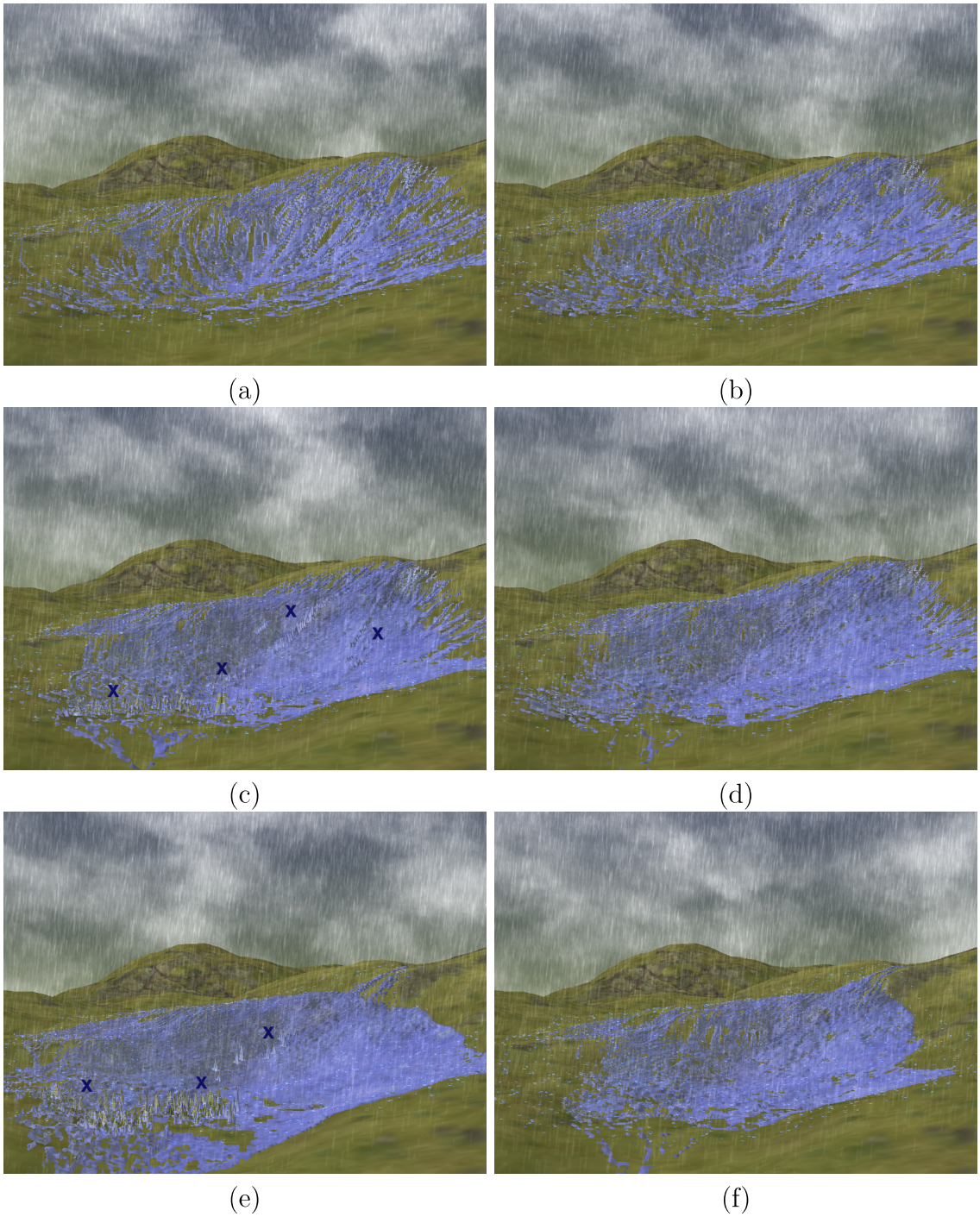


Figura 6.4: (a,c,e) Simulação do fluxo sem computar a tensão superficial. (b,d,f) Utilizando coeficiente de tensão $\sigma = 0.7$.

5. Animação dos Respingos

Na Figura 6.5 é possível observar alguns respingos (*splashes*) produzidos pelas gotas ao atingirem a superfície do fluido. A evolução temporal de cada respingo é representada por uma sequência de texturas semelhante a ilustrada na Figura 5.10. A orientação local da superfície também é considerada, dependendo da declividade as gotas podem produzir diferentes respingos. Ressaltamos que, para efeitos de ilustração, os respingos foram produzidos em um escala aumentada.

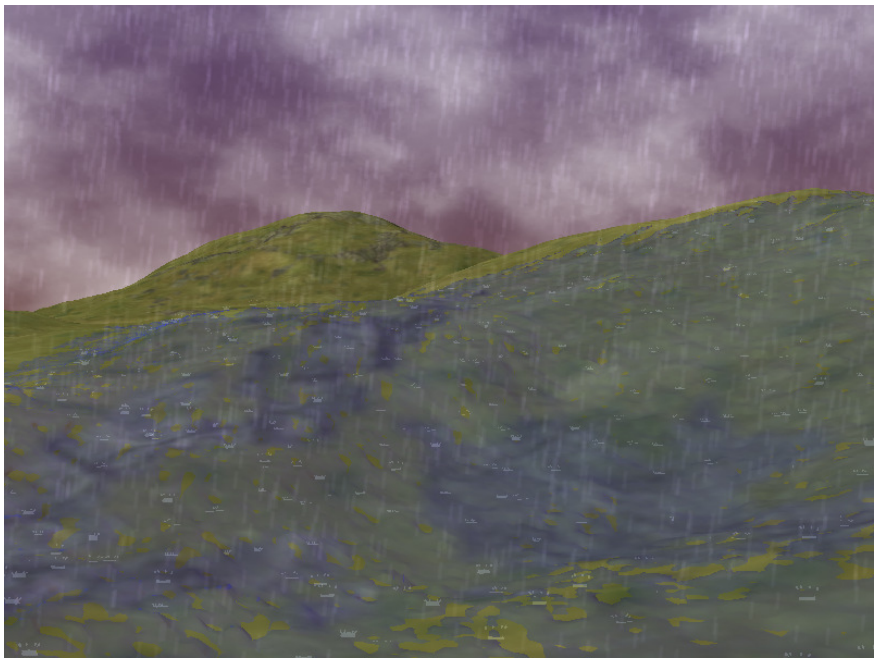


Figura 6.5: Efeitos (em uma escala aumentada) obtidos com a animação dos respingos.

6.2 *Rendering* da Chuva em Vídeos Reais

Para os testes que serão apresentados nessa seção utilizamos apenas o sistema da chuva, sem fluxo superficial e respingos. O objetivo desses testes é demonstrar a aplicabilidade do protótipo para aplicações de processamento de vídeo. Com esse objetivo em mente produzimos alguns vídeos, em cenários distintos, para a inserção da chuva simulada pelo nosso sistema.

Esse processo é realizado de forma bem simples, renderizando cada *frame* do vídeo como uma imagem bidimensional. Em seguida o sistema de partículas da chuva é renderizado de forma que exista uma intersecção entre o plano da imagem e o volume do cubo definido pelo sistema da chuva.

A fonte de iluminação utilizada pela PRT na renderização das partículas é o *environment map* gerado a partir das imagens do vídeo. Desta forma, a cor computada para as partículas da chuva sofre influência do ambiente relativo a cada vídeo, tornando o resultado final ainda mais realista. Porém, é preciso ter em mente que a chuva é praticamente imperceptível em imagens estáticas. Boa parte da percepção que temos é devido ao seu movimento, além dos efeitos relacionados ao fenômeno da chuva, tais como respingos e acúmulo d'água nas superfícies [1].

O sistema foi avaliado em 4 vídeos, produzidos em cenários distintos, como é possível observar nas Figuras 6.7, 6.8, 6.9 e 6.10. Como foi mencionado anteriormente, buscamos produzir esses vídeos em ambientes condizentes com o contexto do fenômeno em questão. Desta forma, os vídeos foram gerados em ambientes externos, em um dia nublado. Além disso, para auxiliar na percepção, os vídeos que contém o lago (Figuras 6.8 e 6.9) foram produzidos em um dia com presença de vento, o que forneceu movimento à superfície do lago, dando a impressão que a chuva inserida foi a responsável por tal movimento.

Figura 6.6: Imagem real de uma tarde chuvosa.

Observando uma imagem real de chuva (Figura 6.6) é possível constatar que a geometria e o padrão de brilho das gotas reais é bastante semelhante ao produzido por nosso framework. Outro efeito observado, também presente nas Figuras 6.7, 6.8, 6.9 e 6.10, diz respeito à percepção das gotas: a chuva torna-se mais perceptível nas regiões das imagens que apresentam maior contraste. Um aspecto negativo observado nos resultados apresentados é a ausência de efeitos de *fog* e *blur*. Embora esses efeitos sejam secundários, sua modelagem correta produz resultados mais próximos da realidade. Em suma, o nível de realismo observado nos resultados mostrou-se satisfatório, indicando que o sistema pode ser utilizado para processamento de vídeos.

Figura 6.7: Imagem da inserção da chuva em um video com uma avenida.

Porém a falta de interação com objetos e personagens da cena acaba gerando um problema. Não é possível inserir a chuva em vídeos que contenham muitos objetos, ou mesmo que tenha a imagem de chão, pois não haverá a interação da chuva e nem o acúmulo d'água nas superfícies. A solução seria a utilização de alguma metodologia para a extração de um mapa de profundidade da cena. Embora esse processo não seja trivial, ele pode ser auxiliado por informações da cena obtidas durante a produção do vídeo. Com o mapa de profundidade em mãos, é possível detectar as colisões das partículas da chuva com a cena, e inserir efeitos como fluxo superficial e respingos.

Figura 6.8: Imagem da inserção da chuva em um vídeo com lago e hotel ao fundo.

Figura 6.9: Imagem da inserção da chuva em um vídeo com lago e pedalinhos passando.

Figura 6.10: Imagem da inserção da chuva em um vídeo de uma floresta.

6.3 Eficiência Computacional

Essa seção apresenta alguns resultados que demostram a eficiência do *framework* proposto. Os tempos de simulação alcançados tornam o sistema atrativo para aplicações em tempo real.

Para a realização dos testes apresentados nas tabelas 6.1 e 6.2 o parâmetro escala de precipitação S foi variado na expressão (5.31), resultando em diferentes configurações de fluxo superficial. Lembramos que o parâmetro S controla a geração das partículas de SPH no sistema. As taxas de FPS foram medidas para três configurações diferentes: (1) Simulação de chuva, detecção de colisão e animação de partículas com SPH (Ch+DC+SPH), observado na primeira linha da Tabela 6.1; (2) Simulação de Ch+DC+SPH mais o método do *Carpet*; (3) Simulação de Ch+DC+SPH+C além da animação de respingos (Ch+DC+SPH+C+R).

O MDT utilizado é ilustrado na Figura 6.3. Todos os exemplos foram simulados até 2500 *frames*. Em torno de 75000 partículas de chuva foram inseridas ao longo da simulação. A Tabela 6.1 lista a taxa de FPS por componente simulado. A última linha da tabela fornece o número total de partículas de SPH geradas ao final de 2500 *steps* de simulação.

Tabela 6.1: Taxa de FPS por componente simulado com atualização um pra um: Simulação da Chuva (Ch), Detecção de Colisão (DC), Animação de Partículas com SPH (SPH), *Carpet* (C) e Animação de Respingos (R).

	Escala de Precipitação (K)			
	4	8	10	
Ch+DC+SPH	1.4	3.6	13.4	FPS
Ch+DC+SPH+C	1.4	3.1	8.9	
Ch+DC+SPH+C+R	1.3	2.9	8.3	
Partículas de SPH	14010	8486	2675	

Observamos na tabela 6.1 que variações pequenas (em torno de 5000) no número de partículas do SPH provocam variações até $3x$ no FPS. Para a geração dessa tabela a relação entre a atualização do sistema da chuva e do SPH é um pra um. Como já foi dito esse esquema pode ser alterado, nos resultados ilustrados na Tabela 6.2 o SPH é atualizado a cada 3 instantes de simulação da chuva. Como a tabela ilustra, o aumento de performance obtido é considerável. O *framework* proposto atingiu taxas consideradas tempo interativo para $S = 8$, bem como para

Tabela 6.2: Taxa de FPS por componente simulado com atualização a cada 3 instantes.

	Escala de Precipitação (K)			
	4	8	10	
Ch+DC+SPH	7.2	14.0	46.3	FPS
Ch+DC+SPH+C	7.6	12.7	29.4	
Ch+DC+SPH+C+R	5.9	10.5	23.1	
Partículas de SPH	14010	8486	2675	

$S = 10$ e taxas próximas para $S = 4$. Os testes também deixaram claro que os resultados são diretamente dependentes do número de partículas do SPH. Ao passo que o número de partículas aumenta (quando diminuímos S) a taxa de FPS cai drasticamente. Esperamos resolver o problema com uma implementação em CUDA do SPH.

Para os testes de inserção da chuva em vídeo, como os ilustrados nas Figuras 6.7-6.10, o número de partículas no sistema da chuva foi variado. A animação foi realizada com 35360, 83200, e 128960 partículas, resultando em taxas de FPS de 332, 252, e 208 respectivamente. Essas taxas indicam que o sistema proposto pode ser utilizado para processamento de video em tempo real. Ressaltamos que esses resultados não incluem o tempo de carregamento do vídeo, apenas seu *rendering* com o sistema de partículas iluminado através da PRT.

Capítulo 7

Conclusões e Trabalhos Futuros

Esse trabalho teve como foco a animação da chuva, bem como efeitos relacionados, para aplicações de computação gráfica. Para tal, foi desenvolvido um *framework* composto por diversas técnicas conhecidas na área. Além disso, uma parte do sistema foi desenvolvida em CUDA, buscando usar os recursos de placas gráficas em prol do ganho de desempenho computacional. Através do *framework* desenvolvido é possível realizar uma animação puramente virtual, com fluxo superficial sobre um MDT qualquer, ou ainda inserir a chuva (sem fluxo superficial) em um vídeo real. Nos resultados experimentais enfatizou-se a potencialidade da técnica proposta quando combinada com métodos eficientes de *rendering*. As taxas de FPS demonstradas atingem tempo real em vários *setups*, tornando o framework bastante promissor para aplicações em tempo real.

Direções futuras nesse trabalho incluem a implementação paralela do SPH, utilizando CUDA ou alguma plataforma *multicore*, uma vez que nos resultados experimentais foi constatado que o SPH é o "gargalo" da simulação. Também temos a intenção de inserir o fluxo superficial (SPH) juntamente com a chuva no vídeo real.

Referências Bibliográficas

[1] STANIK, S., WERMAN, W. "Simulation of rain in videos", *Int. J. Comput. Vis. Texture*, pp. 95–100, 2003.

[2] FENG, Z.-X., TANG, M., DONG, J.-X., et al. "Real-time rendering of raining animation based on the graphics hardware acceleration", *Int. Conf. on Comp. Supp. Cooperative Work in Design*, v. 2, pp. 734–739, 2005.

[3] TARIQ, S. "Rain". In: *Technical report (NVIDIA)*, 2007.

[4] TATARCHUK, N. "Artist-directable real-time rain rendering in city environments". In: *SIGGRAPH Courses*, pp. 23–64, 2006.

[5] PUIG-CENTELLES, A., RIPOLLES, O., CHOVER, M. "Creation and control of rain in virtual environments", *The Visual Computer*, v. 25, n. 11, pp. 1037–1052, nov. 2009.

[6] WANG, L., LIN, Z., FANG, T., et al. "Real-time rendering of realistic rain". In: *SIGGRAPH*, p. 156, 2006.

[7] GARG, K., NAYAR, S. "Photorealistic rendering of rain streaks", *ACM Trans. Graph.*, v. 25, n. 3, pp. 996–1002, 2006. ISSN: 0730-0301. doi: http://doi.acm.org/10.1145/1141911.1141985.

[8] GREEN, R. "Spherical Harmonic Lighting: The Gritty Details", *Archives of the Game Developers Conference*, March 2003. Disponível em: <`http://citeseer.ist.psu.edu/contextsummary/2474973/0`>.

[9] BARCELLOS, B., APOLINÁRIO, A. L., GIRALDI, G. A. *Um Modelo Digital de Terreno Baseado em Triangulação Retangular Adaptativa.* Relatório técnico, Laboratório Nacional De Computação Científica, http://virtual01.lncc.br/ barcellos/TC/Um20Baseado

[10] DESBRUN, M., GASCUEL, M.-P. "Smoothed particles: a new paradigm for animating highly deformable bodies". In: *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pp. 61–76, New

York, NY, USA, 1996. Springer-Verlag New York, Inc. ISBN: 3-211-82885-0.

[11] LIU, G. R., LIU, M. B. *Smoothed particle hydrodynamics : a meshfree particle method*. New Jersey, World Scientific, 2003. ISBN: 9812384561.

[12] MÜLLER, M., CHARYPAR, D., GROSS, M. "Particle-Based Fluid Simulation for Interactive Applications". In: *Proceedings of ACM SIGGRAPH symposium on Computer animation*, 2003.

[13] KIPFER, P., WESTERMANN, R. "Realistic and interactive simulation of rivers". In: *Proc. of Graphics Interface*, pp. 41–48, 2006.

[14] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., et al. "A Survey of General-Purpose Computation on Graphics Hardware". In: *Eurographics 2005, State of the Art Reports*, pp. 21–51, ago. 2005.

[15] NVIDIA CORPORATION. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2010.

[16] STONE, J. E., GOHARA, D., SHI, G. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems", *Computing in Science and Engineering*, v. 12, pp. 66–73, 2010. ISSN: 1521-9615. doi: http://doi.ieeecomputersociety.org/10.1109/MCSE.2010.69.

[17] WANG, N., WADE, B. "Rendering falling rain and snow". In: *SIGGRAPH*, p. 14, 2004.

[18] WANG, N., WADE, B. "Rendering falling rain and snow". In: *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pp. 14–, New York, NY, USA, 2004. ACM. ISBN: 1-58113-896-2. doi: http://doi.acm.org/10.1145/1186223.1186241. Disponível em: <`http://doi.acm.org/10.1145/1186223.1186241`>.

[19] REEVES, W. T. "Particle Systems: A Technique for Modeling a Class of Fuzzy Objects", *ACM Trans. Graph.*, v. 2, n. 2, pp. 91–108, 1983. ISSN: 0730-0301. doi: http://doi.acm.org/10.1145/357318.357320.

[20] ROUSSEAU, P., JOLIVET, V., GHAZANFARPOUR, D. "Realistic real-time rain rendering", *Computers & Graphics*, v. 30, n. 4, pp. 507–518, 2006.

[21] HEO, N., KO, H.-S. "Detail-preserving fully-Eulerian interface tracking framework". In: *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA

'10, pp. 176:1–176:8, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0439-9. doi: http://doi.acm.org/10.1145/1866158.1866198. Disponível em: <http://doi.acm.org/10.1145/1866158.1866198>.

[22] CRANE, K., LLAMAS, I., TARIQ, S. "Real-Time Simulation and Rendering of 3D Fluids". In: Nguyen, H. (Ed.), *GPU Gems 3*, Addison Wesley Professional, cap. 30, ago. 2007. Disponível em: <http://my.safaribooksonline.com/9780321545428/ch29>.

[23] HARADA, T., KOSHIZUKA, S., KAWAGUCHI, Y. "Smoothed particle hydrodynamics on GPUs". In: *Computer Graphics International*, pp. 63–70, 2007.

[24] GOSWAMI, P., SCHLEGEL, P., SOLENTHALER, B., et al. "Interactive SPH simulation and rendering on the GPU". In: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pp. 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. Disponível em: <http://portal.acm.org/citation.cfm?id=1921427.1921437>.

[25] FOSTER, N., FEDKIW, R. "Practical Animation of Liquids". In: *SIGGRAPH '01*, 2001.

[26] STAM, J. "Stable Fluids". In: *Siggraph 1999*, pp. 121–128. Addison Wesley Longman, 1999.

[27] MÜLLER, M., CHARYPAR, D., GROSS, M. "Particle-Based Fluid Simulation for Interactive Applications". In: *Proceedings of ACM SIGGRAPH symposium on Computer animation*, 2003.

[28] PREMOZE, S., TASDIZEN, T., BIGLER, J., et al. "Particle-Based Simulation of Fluids", *Computer Graphics Forum*, v. 22, n. 3, pp. 401–410, 2003. doi: 10.1111/1467-8659.00687. Disponível em: <http://dx.doi.org/10.1111/1467-8659.00687>.

[29] BRIDSON, R. *Fluid simulation for computer graphics*. A.K. Peters, 2008.

[30] HARADA, T., KOSHIZUKA, S., KAWAGUCHI, Y. "Smoothed Particle Hydrodynamics on GPUs". In: *Proc. of Computer Graphics International*, pp. 63–70, 2007.

[31] LORENSEN, W. E., CLINE, H. E. "Marching cubes: A high resolution 3D surface construction algorithm". In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH

'87, pp. 163–169, New York, NY, USA, 1987. ACM. ISBN: 0-89791-227-6. doi: http://doi.acm.org/10.1145/37401.37422. Disponível em: <http://doi.acm.org/10.1145/37401.37422>.

[32] MONAGHAN, J. J. "Smoothed particle hydrodynamics", *Reports on Progress in Physics*, v. 68, n. 8, pp. 1703–1759, ago. 2005. ISSN: 0034-4885. doi: 10.1088/0034-4885/68/8/R01. Disponível em: <http://dx.doi.org/10.1088/0034-4885/68/8/R01>.

[33] LIU, M., LIU, G. "Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments", *Archives of Computational Methods in Engineering*, v. 17, n. 1, pp. 25–76, mar. 2010. ISSN: 1134-3060. doi: 10.1007/s11831-010-9040-7. Disponível em: <http://dx.doi.org/10.1007/s11831-010-9040-7>.

[34] KROG, Ø. E. *GPU-based Real-Time Snow Avalanche Simulations*. Tese de Mestrado, Norwegian University of Science and Technology, jun. 2010. Disponível em: <http://code.google.com/p/gpusphsim/>.

[35] SLOAN, P.-P., LUNA, B., SNYDER, J. "Local, deformable precomputed radiance transfer". In: *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pp. 1216–1224, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1186822.1073335. Disponível em: <http://doi.acm.org/10.1145/1186822.1073335>.

[36] KAUTZ, J., SLOAN, P.-P., LEHTINEN, J. "Precomputed radiance transfer: theory and practice". In: *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1198555.1198682. Disponível em: <http://doi.acm.org/10.1145/1198555.1198682>.

[37] GOURAUD, H. "Continuous Shading of Curved Surfaces", *IEEE Transactions on Computers*, v. C-20, n. 6, pp. 623–629, 1971.

[38] KAJIYA, J. T. "The rendering equation", *SIGGRAPH Comput. Graph.*, v. 20, n. 4, pp. 143–150, 1986. ISSN: 0097-8930. doi: http://doi.acm.org/10.1145/15886.15902.

[39] SLOMP, M. P. B., OLIVEIRA, M. M., PATRÍCIO, D. I. "A Gentle Introduction to Precomputed Radiance Transfer", *RITA*, v. 13, n. 2, pp. 131–160, 2006.

[40] SLOAN, P.-P., KAUTZ, J., SNYDER, J. "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments". In: *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 527–536, New York, NY, USA, 2002. ACM. ISBN: 1-58113-521-1. doi: http://doi.acm.org/10.1145/566570.566612.

[41] ROCHA, C. H. B. *Geoprocessamento: Tecnologia Transdiciplinar*. Books Editora, 2000.

[42] L3DT. "A Windows application for generating terrain maps and textures". . http://www.bundysoft.com/L3DT/ (Último acesso em 14 de Março de 2011).

[43] SCHLICK, C. "An Inexpensive BRDF Model for Physically-based Rendering", *Computer Graphics Forum*, v. 13, pp. 233–246, 1994.

[44] MIHALEF, V., METAXAS, D., SUSSMAN, M. "Simulation of two-phase flow with sub-scale droplet and bubble effects", *Computer Graphics Forum*, v. 28, n. 2, pp. 229–238, 2009. ISSN: 0167-7055. doi: 10.1111/j.1467-8659.2009.01362.x. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2009.01362.x>.

[45] OPENGL, SHREINER, D., WOO, M., et al. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, ago. 2005. ISBN: 0321335732. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0321335732>.

[46] ROST, R. J. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, jan. 2006. ISBN: 0321334892. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0321334892>.