Universidade Federal Fluminense

JOSÉ RICARDO DA SILVA JUNIOR

Simulação computacional em tempo real de fluidos utilizando o método SPH em ambiente heterogêneo CPU/GPU

NITERÓI

2010

JOSÉ RICARDO DA SILVA JUNIOR

Simulação computacional em tempo real de fluidos utilizando o método SPH em ambiente heterogêneo CPU/GPU

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Computação Visual e Interfaces.

Orientador:

Prof. Esteban Walter Gonzalez Clua, D.Sc.

Universidade Federal Fluminense

NITERÓI

2010

Simulação computacional em tempo real de fluidos utilizando o método SPH em ambiente heterogêneo CPU/GPU

José Ricardo da Silva Junior

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Prof. Esteban Walter Gonzalez Clua, D.Sc. / IC-UFF
(Orientador)

Prof. Paulo Aristarco Pagliosa, D.Sc. / UFMS
(Co-Orientador)

Prof. Marcelo Dreux, Ph.D. / PUC-RIO

Prof. Anselmo Antunes Montenegro, D.Sc. / IC-UFF

Niterói, 06 de Agosto de 2010.

Dedico este trabalho aos meus pais pelo apoio, carinho e dedicação que sempre tiveram por mim.

Agradecimentos

Ao meu orientador, professor Esteban Clua pelo seu incentivo e apoio nos momentos difíceis, e pela confiança depositada em mim desde o meu ingresso no mestrado.

Ao meu co-orientador, professor Paulo Pagliosa pela ajuda, explicações e incentivo para a realização deste trabalho.

À minha esposa Giselle pelo amor, paciência e compreensão da minha ausência e por ser uma das pessoas que sempre me apoiou e me deu forças para continuar esta jornada.

Agradeço a minha família, pais e irmãos, pois sempre me apoiaram e me ajudaram durante os momentos difíceis.

Aos professores Otton Teixeira e Anselmo Antunes pelas suas excelentes aulas, apoio e contribuição no desenvolvimento deste trabalho.

Ao pessoal do MediaLab pela ajuda e amizade que foi desenvolvida.

Ao amigo Erick Passos, pela suas excelentes explicações e ajuda durante o desenvolvimento do meu primeiro paper.

Ao amigo Marcelo Zamith pela sua disponibilidade e atenção durante as vezes em que precisei de sua ajuda no desenvolvimento deste trabalho.

À CAPES pelo apoio financeiro para o desenvolvimento deste trabalho.

A todo o pessoal do Instituto de Computação pelo apoio e amizade desenvolvida ao longo desse tempo.

Resumo

Fenômenos físicos vêm sendo estudados desde a década de 70 devido à sua importância para a humanidade. Em particular, o sangue que circula nas artérias, a força do vento que balança a copa das árvores e a força de arrasto que é causada pelo ar ao redor das asas de um avião são exemplos de fenômenos físicos estudados em dinâmica de fluidos e que desempenham um papel importante na vida das pessoas. Inicialmente, esses fenômenos só eram possíveis de ser estudados através de experimentos reais de baixa escala ou estudos teóricos.

Com o advento da computação, uma nova ferramenta surgiu para auxiliar neste estudo: a simulação numérica do comportamento físico de fluidos, dando origem a uma nova área do conhecimento chamada *dinâmica de fluidos computacional*, responsável pelo desenvolvimento de algoritmos numéricos que possibilitam sua resolução utilizando a computação.

Através do crescimento da capacidade de processamento da GPU, a simulação numérica em computador encontrou um ambiente capaz de resolver equações e apresentar os seus resultados em tempo real, na maioria das vezes, além de permitir interações complexas como troca de calor e interação de fluidos com o ambiente. O trabalho desenvolvido nesta dissertação tem por objetivo construir uma arquitetura heterogênea inovadora entre GPU e CPU, que permite o processamento da simulação de fluidos e corpos rígidos com dupla interação entre as partículas e destas com o ambiente onde estão inseridos. Consequentemente, também é desenvolvida uma nova estrutura de aceleração computacional capaz de permitir a comunicação entre processadores diferentes e memórias localizadas em espaços independentes de endereçamento.

Abstract

Physical phenomena is being studied since 70's due its importance to humanity. Particularly, the blood that flows in the arteries, the wind's forces that moves trees and the drag force caused by the air over a plane's wings are examples of physical phenomena studied by fluid dynamics that plays an important role in people's life. At first, these phenomena were only possible to be studied through small scaled real experiments or theoretically.

After the computation advent, a new tool grow up to help these research: the numerical simulation of the physical fluid's behaviour, originating a new research field called *computational fluid dynamics*, responsible for development of numerical algorithms to solve the fluid equation computationally.

Through the GPU's processing capability, the numerical simulation using computers found a new ambient capable of solving these equations and presents the results of it in real time, most of times, beyond allowing for complex interactions like heat exchange and fluid interactions within the ambient where they live. The objective of the work developed in this master dissertation is to develop an innovative heterogeneous architecture using GPU and multicore CPU to allows for the simulation processing of fluids and rigid bodies and these with the ambient where they live in. Beyond this, a new acceleration structure is developed to allow the communication between different processors and memory space located at different address.

Palavras-chave

1. Hidrodinâmica de partículas suavizadas
2. Dinâmica de Fluidos Computacional
3. Simulação de Fluidos
4. Balanceamento de Carga GPU/CPU

Glossário

CPU	:	Unidade de Processamento Central
CUDA	:	Computer Unified Device Architecture
DFC	:	Dinâmica de Fluidos Computacional
FBO	:	Frame Buffer Object
Fragment Shader	:	Programa executado em GPU para cada fragmento renderizado
GLSL	:	OpenGL Shader Language
GPGPU	:	Processamento de tarefas de propósito geral em GPU
GPU	:	Unidade gráfica de processamento
Shader	:	Programa é executado em determinados estágios da GPU
SIMT	:	Single Instruction Multiple Thread
SPH	:	Smoothed Particle Hydrodynamics
Wrapper	:	Mapeamento de funções em uma linguagem de programação para outra

Sumário

Lista de Figuras **xi**

Lista de Tabelas **xiv**

1 Introdução **1**

1.1 Visão geral do problema 2

1.2 Contribuições 3

1.3 Organização da dissertação 4

2 Fundamentos **5**

2.1 Simulações numéricas 5

2.2 Discretização espacial 6

2.2.1 Métodos baseados em malha 6

2.2.2 Métodos baseados em partículas 8

2.3 Dinâmica dos fluidos 9

2.3.1 Equações governantes 10

2.4 Smoothed particle hydrodynamics 11

2.4.1 Equação de forças usando o SPH 12

2.4.2 Funções de suavização 13

2.4.3 XSPH 16

2.5 Trabalhos relevantes 17

3 Simulação **21**

3.1 Simulação de fluidos 21
3.1.1 Cálculo da vizinhança 22
3.1.2 Cálculo da densidade 22
3.1.3 Cálculo das forças internas 22
3.1.4 Cálculo das forças externas 22
3.1.5 Integração 23
3.2 Simulação de corpos rigidos 23
3.3 Detecção de colisão 24
3.4 Representação de corpos rígidos 25
3.4.1 Depth peeling 26
3.4.2 Algoritmo de discretização 27

4 Estrutura de aceleração 31

4.1 Subdivisão espacial 31
4.2 Definição da estrutura de aceleração 33
4.3 Mapeamento entre estruturas de aceleração distintas 35
4.4 Computação em GPU 36
4.4.1 CUDA - Compute Unified Device Architecture 36
4.5 Organização das informações de estado em memória 38

5 Ambiente heterogêneo (CPU-GPU) 42

5.1 Identificação das tarefas 42
5.2 Modelo de tarefas 43
5.3 Distribuição de tarefas 45
5.3.1 Classificação de partículas 46
5.3.2 Calculo de densidade 47
5.3.3 Cálculo das forças internas 48

5.3.4 Broad phase 48

5.3.5 Cálculo das forças externas 50

5.3.6 Integração da equação de movimento 52

6 Resultados **54**

6.1 Estrutura de aceleração 54

6.2 Simulação de fluidos e corpos rígidos 55

7 Conclusão e trabalhos futuros **62**

Referências **64**

Lista de Figuras

2.1 Simulação Lagrangeana: à esquerda a malha sem nenhuma deformação; à direita a malha deformada após a simulação utilizando o algoritmo *Imesh* [3]. 7

2.2 Exemplo de simulação Euleriana: o fluido desloca-se sobre a grade fixa no domínio da simulação. 8

2.3 Diferentes tipos de escoamento. Inicialmente a fumaça começa com um escoamento laminar e após algum tempo passa para o turbulento através do escoamento transicional. Figura adaptada de http://content.answers.com. 10

2.4 Gráfico da função de suavização utilizando h=1. Figura adaptada de [18]. . 14

2.5 Gráfico da função de suavização utilizando h=1. Para uma melhor visibilidade, o gráfico possui escalonamento diferente. Figura adaptada de [18]. . 15

2.6 Gráfico da função de suavização utilizando h=1. Para uma melhor visibilidade, o gráfico possui escalonamento diferente. Figura adaptada de [18]. . 16

2.7 Simulação de queda d'água utilizando partículas [44]. 17

2.8 Simulação de gases utilizando método semi-Lagrangeano proposto em [45]. 18

2.9 Enchimento de um copo d'água utilizando SPH [30]. 19

2.10 Simulação de escoamento de fluido sobre terrenos acidentados [19]. 19

2.11 Interação de fluidos com corpos rígidos [21]. 20

3.1 Tarefas necessárias para simulação de fluidos. 21

3.2 Um exemplo de geometria e seus mapas de profundidade. Extraído de [49]. 26

3.3 Esquerda: corpo rígido a ser discretizado; direita: corpo rígido discretizado utilizando raio elevado, causando interpenetração de partículas de fluido (em vermelho). 28

3.4 Processamento do bule utilizando vários raios como parâmetro: (a) raio 0.3 com 7337 partículas; (b) raio 0.5 com 2674 partículas; (c) raio 0.9 com 600 partículas; (d) raio 1 com 668 partículas. 30

4.1 Configuração de célula utilizada no SPH. 34

4.2 Mapeamento de uma esfera entre células de diferentes tamanhos. 35

4.3 Arquitetura de CPU e GPU. Extraída de [36]. 37

4.4 Comparação de desempenho entre CPU e GPU. Extraída de [36]. 37

4.5 Organização de memória em GPU. Extraída de [36]. 38

4.6 Organização do estado das partículas de fluido em GPU. 40

4.7 Relação entre as partículas e o centro de massa do corpo rígido. 40

4.8 Estado dos corpos rígidos armazenados em GPU. 41

5.1 Dependência entre as tarefas, representada pelas setas tracejadas. 43

5.2 Comportamentos durante a simulação. De cima para baixo: corpo rígido sem interação com outro corpo rígido; corpo rígido sem interação com fluido e corpo rígido sem interação com fluido e outro corpo rígido. 44

5.3 Modelo implementado para execução da simulação de corpos rígidos e fluidos com dupla interação. 45

5.4 Diagrama de distribuição de tarefas entre GPU e CPU. No bloco da GPU, em linhas cheias, as tarefas inerentes à simulação de fluidos, enquanto em linhas tracejadas as tarefas inerentes à simulação de corpos rígidos. 46

5.5 Execução da *broad phase* utilizando múltiplos núcleos em CPU. Rk_{bb} representa o *bounding box* do corpo rígido k. F representa colisão com fluido e C com corpo rígido. 49

5.6 Informações geradas após o processamento da etapa de *broad phase*. Partículas que colidiram com corpos rígidos são armazenadas independentemente daqueles que colidiram com fluidos. 51

5.7 Verificação de colisão da partículas em amarelo com as células adjacentes. . 52

6.1 Gráfico comparativo da utilização da estrutura de aceleração desenvolvida. 56

6.2 Gráfico comparativo da simulação de fluidos em GPU e CPU. 57

6.3 Gráfico comparativo da simulação de corpos rígidos em GPU e CPU. . . . 58

6.4 Renderização de uma cena da simulação de 430 corpos rígidos totalizando 32680 partículas. 59

6.5 Gráfico comparativo da simulação de fluidos e corpos rígidos em GPU e CPU sem interação. 59

6.6 Gráfico comparativo da simulação de fluidos e corpos rígidos em GPU, CPU e modo heterogêneo(GPU e CPU) com dupla interação. 60

6.7 Renderização de uma cena onde verifica-se o comportamento do corpo rígido através da aplicação de forças pelo fluido. 61

6.8 Renderização de uma cena onde verifica-se o comportamento do fluido através da aplicação de forças pelo corpo rígido. 61

Lista de Tabelas

3.1 Variáveis utilizadas na simulação de fluidos. 22

3.2 Variáveis e seus respectivos tipos utilizados na representação de corpos rígidos. 23

3.3 Especificação do volume de visualização. 29

4.1 Tipos de memórias disponíveis em GPU. 39

6.1 Tempo de acesso, em segundos, utilizando uma e duas grades regulares. . . 55

6.2 Tempo de acesso, em segundos, utilizando uma e três grades regulares. . . 55

6.3 Resultado da simulação de fluidos em GPU e CPU. 56

6.4 Resultado da simulação de corpos rígidos em GPU e CPU. CR: corpos rígidos. 57

6.5 Resultado da simulação de fluidos e corpos rígidos em GPU e CPU sem interação. CR: corpos rígidos; CRP: numero de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema. . 58

6.6 Resultado da simulação de fluidos e corpos rígidos em GPU com dupla interação. CR: corpos rígidos; CRP: numero de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema. . 58

6.7 Resultado da simulação de fluidos e corpos rígidos em CPU com dupla interação. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema. . 60

6.8 Resultado da simulação de fluidos e corpos rígidos em modo heterogêneo de CPU e GPU com dupla interação. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema. 60

6.9 Comparação da arquitetura heterogênea de CPU e GPU e da arquitetura de GPU. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema. . 61

Capítulo 1

Introdução

Animações de fenômenos naturais tais como explosões nucleares, interação de fótons com superfície de aspecto rugoso entre outras têm atraído a atenção de muitos pesquisadores, conforme podemos observar na qualidade de imagens e filmes gerados por computador vistos atualmente. Dessa forma podemos corretamente supor que investimentos tanto de cunho intelectual como financeiro vêm sendo aplicados na simulação desses fenômenos naturais, principalmente pela indústria de entretenimento.

Especificamente, fenômenos naturais tais como o escoamento de água sobre uma superfície, a fumaça emitida por veículos, a ação do vento sobre as asas de um avião são classificados como fluidos com diferentes tipos de comportamentos. Tais fenômenos, por mais simples que possam parecer, possuem um alto grau de complexidade para serem simulados numericamente.

Antes do advento da computação, o estudo desses fenômenos era feito através de estudos práticos com a simulação real do fenômeno a ser estudado em ambiente controlado ou de estudos teóricos, aprimorando as equações de movimento de cada tipo de escoamento. Porém, em alguns casos, nem sempre o tratamento teórico e/ou experimental são satisfatórios devido a duas razões [11]: (1) o fenômeno a ser observado nem sempre é passível de reprodução em laboratório, mesmo em escala reduzida; e (2) proibitivos custos de tempo e de montagem. Como exemplo de (1) podemos citar a previsão do tempo que trata dos movimentos do ar atmosférico com influência direta no tempo e o movimento do sangue nas artérias. Como exemplo de (2) temos o estudo da aerodinâmica em túneis de vento de alta velocidade que são caros de serem operados, geralmente fornecendo amostras pequenas das quantidades físicas tais como pressão e velocidade.

Com o advento da computação, surgiu uma nova área do conhecimento chamada dinâmica de fluidos computacionais (DFC), responsável pelo estudo de métodos numéricos que permitem a simulação do comportamento de cada tipo de fluido por computador. A simulação numérica computacional, de forma alguma, surgiu para colocar de lado os experimentos reais e estudos teóricos, mas sim complementá-los, permitindo que seus resultados possam ser utilizados para a escolha de melhores experimentos para o primeiro caso e a validade de novos modelos utilizando resultados da simulação numérica e experimentos para o segundo caso.

A dinâmica de fluidos computacional estuda métodos numéricos para a resolução das equações de Navier-Stokes, as quais definem o modelo matemático do comportamento de um fluido. Como essas equações não possuem uma solução analítica, utilizam-se métodos numéricos tais como diferenças finitas, volumes finitos ou qualquer outro método de discretização dessas equações para sua resolução.

A simulação por computador de escoamento de fluidos deve levar em consideração o resultado a ser atingido. Simulações onde são prezados resultados precisos exigem maior esforço computacional, geralmente sendo executadas *offline*. Porém, em simulações onde se deseja resultado em tempo real, devem-se efetuar algumas simplificações nas equações de Navier-Stokes, o que pode tornar seus resultados inaceitáveis do ponto de vista científico, porém, visualmente convincentes.

Outro aspecto importante durante a simulação de fluidos computacional é a interação destes com os objetos ao seu redor. Em fenômenos naturais de escoamento de fluidos, é possível observar que estes interagem com os corpos aos quais tenham contato, gerando forças nesses. Por sua vez, estes mesmo corpos também exercem forças nas partículas de fluido, gerando o que chamamos *acoplamento de dupla interação*. Devido à sua complexidade, muitas vezes esse tipo de interação não é considerado durante a simulação de escoamento de fluido em tempo real.

1.1 Visão geral do problema

Muitos trabalhos realizam a simulação de fluidos utilizando CPU e mais recentemente utilizando GPU de forma isolada. O mesmo se pode observar quando se trata de simulação independente de corpos rígidos ou em conjunto com fluidos. Isso ocorre devido à degradação de desempenho que pode ocorrer durante a utilização da CPU em conjunto com a GPU, caso não seja executada de forma apropriada, ou seja, levando em consi-

deração a capacidade de processamento da GPU e CPU. Muitas vezes, essa degradação ocorre devido às constantes trocas de informações entre espaços de memórias de GPU e CPU, bem como o grau de paralelismo suportado por cada processador. Além disso, também pode ocorrer em casos de tarefas mal distribuídas entre CPU e GPU durante o processamento.

Durante a simulação de fluidos e corpos rígidos, poder-se-ia adotar um modelo de distribuição de tarefas conforme proposto em [16], obtendo uma distribuição de carga entre GPU e CPU de forma automática. Porém, um fator primordial aqui é a dependência que existe durante a simulação de fluidos e corpos rígidos com dupla interação. Nesse caso, fluidos interagem com corpos rígidos e vice-versa, através de forças repulsivas. Dessa forma, podemos notar que existe uma dependência de informações entre fluidos e corpos rígidos e, consequentemente, uma dependência de tarefas que deve ser resolvidas a fim de evitar problemas de inconsistências durante o acesso/armazenamento dessas informações.

Além disso, simulação de corpos rígidos e fluidos podem existir em ambientes separados, cada um com uma estrutura de dados que mais lhe seja conveniente. Com a dupla interação entre corpos rígidos e fluidos, é necessário que essas diferentes estruturas de aceleração interajam entre si, podendo as mesmas serem completamente independentes e ainda estar localizadas em diferentes espaços de memória.

Dessa forma, a simulação de fluidos e corpos rígidos em ambiente heterogêneo de CPU e GPU com dupla interação deve considerar a troca de informações entre espaços de memórias diferentes, a distribuição de tarefas entre GPU e CPU e a dependência dessas tarefas durante sua execução. Com isso, pode-se utilizar a capacidade computacional da GPU paralelamente à CPU, maximizando a carga de processamento de cada um destes processadores.

Neste trabalho, pretende-se desenvolver uma arquitetura híbrida de múltiplos núcleos de CPU e GPU para simulação de fluidos com dupla interação, ou seja, fluidos gerando forças nos corpos rígidos e estes gerando forças nos fluidos. Em paralelo, também se deve desenvolver uma estrutura de aceleração capaz de acomodar trocas de informações entre estas diferentes arquiteturas.

1.2 Contribuições

A principal contribuição deste trabalho é o desenvolvimento de uma arquitetura heterogênea original entre CPU e GPU para a simulação de fluidos e corpos rígidos com dupla

interação, explorando os vários núcleos disponíveis na CPU. Para isso, deve-se desenvolver uma arquitetura que permita a ocupação de ambos os processadores e evite uma interdependência entre os mesmos durante a simulação. Após consulta na literatura, não é de conhecimento do autor o tratamento específico da simulação desenvolvida em ambiente heterogêneo de CPU e GPU.

Outra contribuição relevante deste trabalho consiste na elaboração de uma estrutura de aceleração capaz de lidar com interações entre corpos rígidos e partículas de fluido. Durante a simulação, várias estruturas de aceleração são utilizadas em GPU, exigindo uma comunicação entre as mesmas. Com isso, essa estrutura de aceleração deve ser capaz de possibilitar a comunicação com outras estruturas localizadas em posições independentes de memória, além de evitar possíveis problemas de sincronização que possam ocorrer durante seu acesso.

Após comparações com outros trabalhos disponíveis na literatura, resultados favoráveis destas contribuições foram observados.

1.3 Organização da dissertação

O restante da dissertação está composta por seis capítulos. O segundo capítulo apresenta os fundamentos da dinâmica de fluidos computacional, bem como suas equações governantes e sua discretização numérica e do domínio do problema. Além disso, também apresenta os fundamentos do *Smoothed Particle Hydrodynamics* para simulação numérica de fluidos baseado em partículas. O terceiro capítulo apresenta uma visão geral do funcionamento da simulação de fluidos e corpos rígidos e a metodologia de discretização utilizada neste trabalho enquanto o quarto capítulo a estrutura de aceleração desenvolvida. No quinto capítulo é apresentada a arquitetura do ambiente heterogêneo de CPU e GPU para a simulação de fluidos e corpos rígidos. Finalmente o sexto capítulo apresenta os resultados obtidos enquanto o sétimo capítulo conclui o trabalho e propõe pesquisas futuras.

Capítulo 2

Fundamentos

Neste capítulo são dados os primeiros passos para o entendimento dos elementos envolvidos na simulação de fluidos computacional. Primeiramente é feita uma introdução às simulações numéricas (Seção 2.1). Após essa introdução são apresentadas as formas de discretização do domínio do problema (Seção 2.2) e posteriormente a teoria da dinâmica de fluidos (Seção 2.3). Em seguida é apresentado o método SPH (Seção 2.4) e finalmente os trabalhos relacionados a este (Seção 2.5).

2.1 Simulações numéricas

Com o aumento da capacidade de processamento, as simulações numéricas se tornaram um meio poderoso para investigação científica. Fenômenos físicos que antes eram estudados através de experimentos reais e suposições teóricas encontram nas simulações numéricas computacional uma valiosa ferramenta para validação de modelos matemáticos desenvolvidos, muitas vezes sem a necessidade de recriação do experimento real para observação.

Inicialmente são desenvolvidos modelos matemáticos utilizando um conjunto de equações governantes que representem o comportamento do fenômeno físico em estudo com condições de contorno e/ou condições iniciais adequadas. Essas equações são, em sua maior parte, formadas por equações diferenciais ordinárias e parciais que não possuem solução analítica.

Para serem resolvidas numericamente em computador, o domínio do problema precisa ser dividido em componentes discretos bem como as equações governantes que são dadas geralmente em sua forma contínua. A partir desta discretização, são utilizados algorit-

mos numéricos para o cálculo das variáveis de campo do problema nesses componentes discretos, criando-se vários cenários de simulação para posterior análise dos resultados obtidos.

Aqui é importante observar que a fidelidade dos resultados do método numérico está intimamente relacionada com a resolução da discretização do domínio do problema, isto é, quanto maior for a resolução do espaço discreto, mais pontos de interpolação existirão e consequentemente mais cálculos precisarão ser efetuados. Além disso, a resolução das equações governantes do fenômeno físico depende do método utilizado para a discretização espacial.

2.2 Discretização espacial

Além da discretização das equações governantes de um fenômeno físico, é necessário a discretização do espaço ou domínio da simulação. Nesta seção, serão apresentadas duas maneiras de efetuar esta discretização do domínio da simulação, de acordo com tipo de algoritmo numérico a ser utilizado para a resolução das equações governantes do fenômeno em estudo.

2.2.1 Métodos baseados em malha

Nos métodos baseados em malha, o espaço de domínio da simulação é discretizado em uma grade composta por um conjunto de vértices conectados por células nos quais são aplicados os métodos numéricos para o cálculo das variáveis do fenômeno físico em estudo.

Estas malhas podem ser de dois tipos [43]: malhas estruturadas e não estruturadas. Nas malhas estruturadas as células vizinhas de cada célula são conhecidas enquanto que nas malhas não estruturadas é necessário o armazenamento explícito da vizinhança de cada célula, utilizando uma lista encadeada, por exemplo.

Conforme descrito em [24], as equações governantes de fenômenos físicos baseados em malhas podem ser divididas em dois tipos: abordagem espacial Euleriana e abordagem material Lagrangeana.

No processo de discretização Lagrangeana, a grade é fixa no material, movendo-se com ele, podendo resultar em expansões e contrações das células durante a deformação do material simulado, conforme observado na figura 2.1. Com isso, percebem-se os seguintes benefícios [24]:

- Utilização de malha somente no domínio do problema, evitando possíveis processamento desnecessário em outras células que não fazem parte do problema;
- Utilização de malha irregular para o tratamento de geometria irregular ou complexa;
- Obtenção facilitada do histórico de todas as variáveis do material em um determinado ponto, visto que todos os pontos se movem junto com o material.

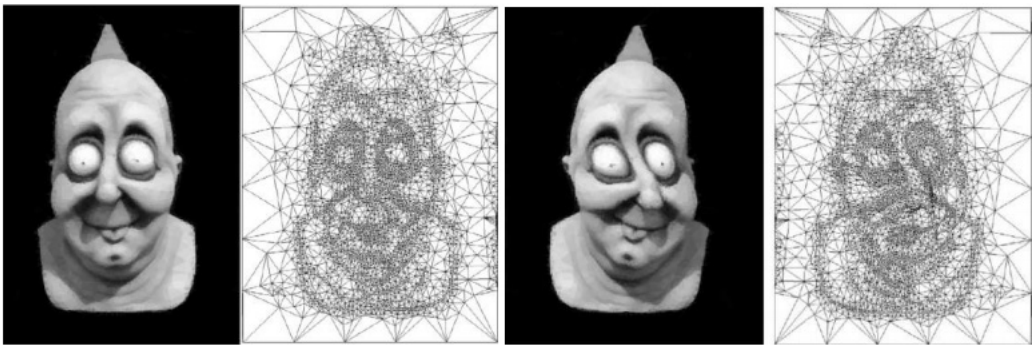

Figura 2.1: Simulação Lagrangeana: à esquerda a malha sem nenhuma deformação; à direita a malha deformada após a simulação utilizando o algoritmo *Imesh* [3].

Porém, podem-se encontrar problemas em trabalhar com métodos Lagrangeanos. Visto que estes se adaptam ao material que são simulados, grandes deformações podem ocorrer na malha. Quando essas deformações ocorrem, a eficiência do método matemático e até os resultados da simulação podem ser prejudicados, já que regiões podem ficar com poucos pontos, enquanto outras com uma quantidade exagerada de pontos de cálculo. Uma das soluções seria o *remesh* do domínio do problema, utilizando métodos Eulerianos para o cálculo das novas variáveis. Porém, esse processo pode ser computacionalmente caro e tornar-se inviável em aplicações interativas.

Na abordagem Euleriana, ao contrário da Lagrangeana, a grade é fixa no domínio da simulação e o material simulado desloca-se sobre essa grade, conforme pode ser visto na figura 2.2. Com isso, deformações muito grandes no material não interferem no resultado numérico da simulação. Apesar desta vantagem, a discretização Euleriana apresenta as seguintes desvantagens:

- Dificuldade do cálculo da superfície livre entre dois fluidos distintos;
- Necessidade de conversão de geometrias irregulares e complexas do domínio do problema em um domínio regular, requerendo muitas vezes um processo de mapeamento numérico computacionalmente complexo;

- Devido a sua posição fixa no espaço, o método Euleriano requer a discretização de todo o domínio da simulação onde o material poderá estar localizado. Com isso, a fim de manter a eficiência para grandes domínios, geralmente diminui-se a precisão do método numérico através da utilização de uma discretização com menor resolução;

- Maior quantidade de memória necessária, visto a necessidade de representação de todo o espaço do domínio da simulação.

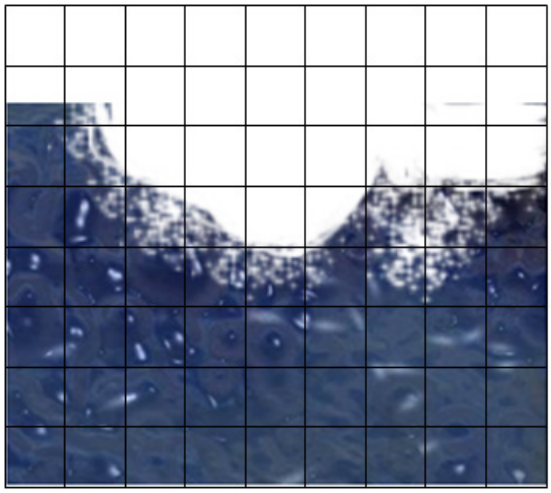

Figura 2.2: Exemplo de simulação Euleriana: o fluido desloca-se sobre a grade fixa no domínio da simulação.

Além dos métodos Lagrangeanos e Eulerianos para resolução de equações governantes, existem os métodos mistos, chamados *ALE (Arbitrary Lagrangian-Eulerian)*[26], os quais utilizam os métodos Lagrangeanos e Eulerianos em conjunto para resolução das equações governantes.

2.2.2 Métodos baseados em partículas

Os métodos baseados em partículas, conhecidos também como *Meshfree Methods* (MM), são métodos de discretização espacial utilizando partículas no domínio da simulação. Possuem as seguintes vantagens em relação aos métodos baseados em malha [24]:

- Possibilitam grandes deformações do domínio da simulação, visto que as informações de conectividade entre os pontos é parte inerente da computação;

- Podem representar objetos com elevada precisão, já que estes objetos podem ser representados por partículas;

- Devido à possibilidade de inserção de novos pontos no domínio da simulação, permitem facilmente o refinamento adaptativo para regiões do domínio da simulação que necessitam de resultados mais precisos;
- Fácil representatividade em sua estrutura de dados, podendo ser utilizado um array de pontos, por exemplo.

Dentre as desvantagens, podemos citar as seguintes [10]:

- Necessidade de um cálculo da vizinhança para cada interação devido à falta de informação de conectividade nos métodos baseados em partículas;
- Esforço computacional necessário para o cálculo da vizinhança, podendo ser superior aos métodos baseados em malha devido ao grande número de partículas necessário para simulações numericamente precisas;
- Configuração do estado inicial das partículas, que tem forte influência nos métodos numéricos sem malha.

Como exemplo dos métodos numéricos sem malha podemos citar o *Smoothed Particle Hydrodynamics* (SPH) que será discutido na seção 2.4 e o método de elementos difusos [33].

Para este trabalho, é adotado o método SPH para simulação de fluidos, já que apresenta melhores benefícios à simulação desenvolvida neste trabalho em relação aos métodos Eulerianos.

2.3 Dinâmica dos fluidos

As dificuldades da simulação numérica do escoamento de fluidos são devidos ao comportamento complexo do fluido, resultado da interação entre vários fenômenos como convecção, difusão superficial, difusão e turbulência [40].

O comportamento de um fluido pode ser classificado da seguinte maneira [11], conforme observado na figura 2.3:

- *Escoamentos Laminares*: são aqueles nos quais camadas finas de fluido parecem deslizar uns sobre os outros;

- *Escoamentos Turbulentos*: são aqueles nos quais o movimento das partículas de fluido se dá caoticamente ou desordenadamente.

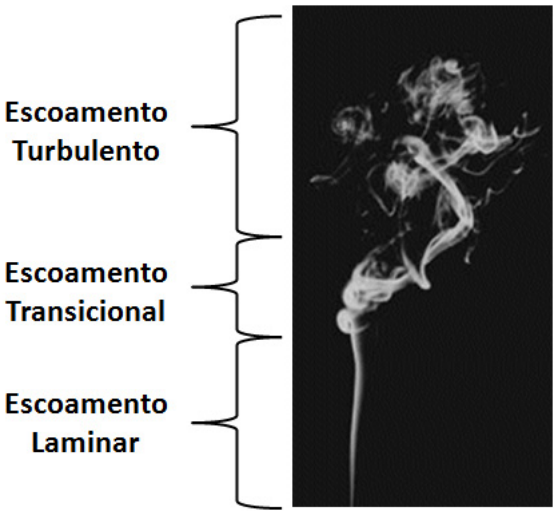

Figura 2.3: Diferentes tipos de escoamento. Inicialmente a fumaça começa com um escoamento laminar e após algum tempo passa para o turbulento através do escoamento transicional. Figura adaptada de http://content.answers.com.

Além desse comportamentos, tem-se também o comportamento *transicional*, sendo este a fase de transição entre o escoamento laminar e o turbulento. É importante salientar que esses movimentos não são propriedades intrínsecas do fluido mas sim de seu estado.

2.3.1 Equações governantes

Descobertas por Claude Navier em 1822 e George Stokes em 1845, as equações de Navier-Stokes são um conjunto de equações que modelam matematicamente o movimento de um fluido, sendo consideradas o melhor modelo matemático para simulação de fluidos.

Existem diferentes formas para sua representação. As equações

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}.\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v} \tag{2.1}$$

e

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho\mathbf{v}) = 0 \tag{2.2}$$

apresentam o modelo para fluidos Newtonianos e não compressíveis. Nessas equações ρ representa a densidade do material, $\mathbf{v}$ representa o campo de velocidade, p o campo de pressão, $\mathbf{g}$ representa as forças externas tal como força da gravidade e μ a constante de viscosidade do fluido.

A equação (2.1) é chamada de *equação da quantidade de movimento* e impõe que a alteração na velocidade deve ser igual à soma das forças que atuam no sistema, sendo resultado direto da segunda lei de Newton. O termo convectivo $\mathbf{v}.\nabla\mathbf{v}$ representa a variação de uma propriedade qualquer do elemento de fluido que se move de uma posição para outra. Nas simulações que utilizam métodos Eulerianos, esse termo é necessário visto que a posição da grade não muda com o movimento do fluido; enquanto que nos métodos baseados em partículas esse termo pode ser desconsiderado, devido ao próprio material que se move.

A equação (2.2) é chamada de *equação de continuidade* ou *conservação de massa* e impõe que na ausência de sumidouros ou sorvedouros a massa do sistema deve ser constante. Para os métodos baseados em partículas, esta equação não é necessária pois é implicitamente satisfeita, já que cada partícula carrega uma quantidade de massa constante [30].

O termo ∇p representa a força de pressão por volume, causando o deslocamento de partículas localizadas em campo de alta pressão para campo de baixa pressão. Em um gás ideal, a pressão está relacionada à densidade através da equação

$$p = k\rho, \tag{2.3}$$

onde k é uma constante que depende da temperatura. Na ausência de forças, essa relação causa a expansão indefinida de um fluido no espaço, muitas vezes sendo um comportamento indesejado. Para a simulação de fluidos incompressíveis, Desbrun [5] sugere a seguinte adaptação:

$$p = k(\rho - \rho_0). \tag{2.4}$$

Com esta adaptação no cálculo da pressão, a densidade do fluido tende a se aproximar de ρ_0, o que representa a densidade de um fluido em repouso. Como exemplo, [30] configura este valor para 1000 kg/m^3 a fim de simular água.

2.4 Smoothed particle hydrodynamics

Smoothed Particle Hydrodynamics (SPH) pertence à classe de métodos Lagrangeanos sem malha. Nesse método numérico, o fluido é representado por um conjunto de partículas e o cálculo de variáveis tais como pressão e densidade é realizado através de interpolação

destas variáveis nas partículas vizinhas. Foi introduzido na comunidade científica em 1977 através dos trabalhos de Lucy [25] e Gingold e Monaghan [13] para modelagem de problemas astrofísicos.

Apesar de permitir que o fluido seja tratado como um espaço contínuo, o cálculo das quantidades físicas é geralmente efetuado na localização das partículas. A equação utilizada pelo SPH para calcular uma quantidade escalar A do fluido na partícula i é dada por [28]

$$A(\mathbf{r}_i) = \sum_j m_j \frac{A_j}{\rho_j} W(\|\mathbf{r}_i - \mathbf{r}_j\|, h), \tag{2.5}$$

onde o subscrito j percorre todas as partículas vizinhas de i, $\mathbf{r}$ representa a posição da partícula, ρ a densidade e W a função de suavização de raio h, explicada posteriormente. Nota-se que massa sobre densidade representa o volume ν. Dessa forma, a equação 2.5 pode ser reescrita na forma

$$A(\mathbf{r}_i) = \sum_j \nu_j A_j W(\|\mathbf{r}_i - \mathbf{r}_j\|, h). \tag{2.6}$$

Conforme visto na equação (2.1), as variáveis de força de pressão e viscosidade do fluido requerem o gradiente e Laplaciano da função, respectivamente. Usando SPH, o gradiente da função é obtido através do cálculo do gradiente sobre a função núcleo W

$$\nabla A(\mathbf{r}_i) = \sum_j \nu_j A_j \nabla W(\|\mathbf{r}_i - \mathbf{r}_{j\|}, h). \tag{2.7}$$

O Laplaciano é calculado da mesma forma, através do cálculo do Laplaciano sobre a função núcleo W

$$\nabla^2 A(\mathbf{r}_i) = \sum_j \nu_j A_j \nabla^2 W(\|\mathbf{r}_i - \mathbf{r}_j\|, h). \tag{2.8}$$

2.4.1 Equação de forças usando o SPH

O cálculo das forças internas de pressão e viscosidade que agem em um fluido utilizando o método SPH pode ocorrer de várias maneiras. Monaghan [28], por exemplo, utiliza a constante da velocidade do som na equação de viscosidade do material a ser simulado.

Neste trabalho, o cálculo das forças internas de pressão e viscosidade serão baseados nas equações propostas em [30]. Utilizando a formulação SPH, para o cálculo da força de pressão, substituem-se o termos da pressão da equação (2.1) por (2.7), originando a

equação

$$f^{pressão}(\mathbf{r}_i) = -\sum_j \nu_j p_j \nabla W(\|\mathbf{r}_i - \mathbf{r}_j\|, h), \tag{2.9}$$

onde p representa a pressão na posição $\mathbf{r}$. Da mesma forma, a viscosidade é obtida através da substituição da força de viscosidade da equação (2.1) por (2.8) originando

$$f^{viscosidade}(\mathbf{r}_i) = \mu \sum_j \nu_j \mathbf{v}_j \nabla^2 W(\|\mathbf{r}_i - \mathbf{r}_j\|, h), \tag{2.10}$$

onde μ é a viscosidade do material e $\mathbf{v}$ sua velocidade.

Porém, conforme pode-se observar para o caso de duas partículas, essas forças não são simétricas, violando a terceira lei de Newton. Para contornar este problema, a simetria da força da pressão é obtida através da equação

$$f^{pressão}(\mathbf{r}_i) = -\sum_j \nu_j \frac{p_i + p_j}{2} \nabla W(\|\mathbf{r}_i - \mathbf{r}_j\|, h), \tag{2.11}$$

enquanto que a força de viscosidade é dada por

$$f^{viscosidade}(\mathbf{r}_i) = \sum_j \frac{\mu_i + \mu_j}{2} \nu_j (\mathbf{v}_j - \mathbf{v}_i) \nabla^2 W(\|\mathbf{r}_i - \mathbf{r}_j\|, h). \tag{2.12}$$

2.4.2 Funções de suavização

Todo cálculo em SPH requer a utilizacão de uma função de suavização, ou *kernel*, para a interpolação das quantidades físicas do fluido. Essa função atribui pesos às quantidades físicas das partículas vizinhas baseada em sua distância durante o cálculo. Partículas que se encontram fora do raio h, chamado de "suporte compacto"por Monaghan [28], não são consideradas.

Müller et al. [30] afirmam que a estabilidade, precisão e velocidade do método SPH depende fortemente da escolha da função núcleo utilizada.

De acordo com [28], a função de suavização precisa possuir as propriedades

$$\int_\Omega W(r, h)\, dr = 1 \tag{2.13}$$

e

$$\lim_{h \to 0} W(r, h) = \delta(r), \tag{2.14}$$

onde δ é a função delta de Dirac

$$\delta(r) = \begin{cases} \infty & r = 0 \\ 0 & \textit{caso contrário.} \end{cases} \tag{2.15}$$

A equação (2.13) define que a função de suavização precisa ser normalizada. Além disso, a função de suavização também precisa ser positiva

$$W(r, h) \geq 0, \tag{2.16}$$

a fim de garantir um comportamento de interpolação [7]. Para garantir invariância sobre rotação do sistema de coordenadas, a função de interpolação deve possuir a propriedade

$$W(r, h) = W(-r, h). \tag{2.17}$$

Caso as equações (2.13) e (2.17) sejam satisfeitas, a interpolação tem precisão de segunda ordem [28].

O uso de três funções de suavização é utilizado neste trabalho, sendo aconselhável por [30]. Dessa forma, para o cálculo da densidade a função de suavização

$$W(r, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \textit{caso contrário} \end{cases} \tag{2.18}$$

é utilizada, onde r representa a magnitude do vetor $\mathbf{r}$. Uma das vantagens desta função de suavização é o não requerimento do cálculo de raiz quadrada, conforme pode ser observado. O gráfico desta função pode ser observado na figura 2.4, cujo gradiente tende a *0* à medida que as partículas se aproximam.

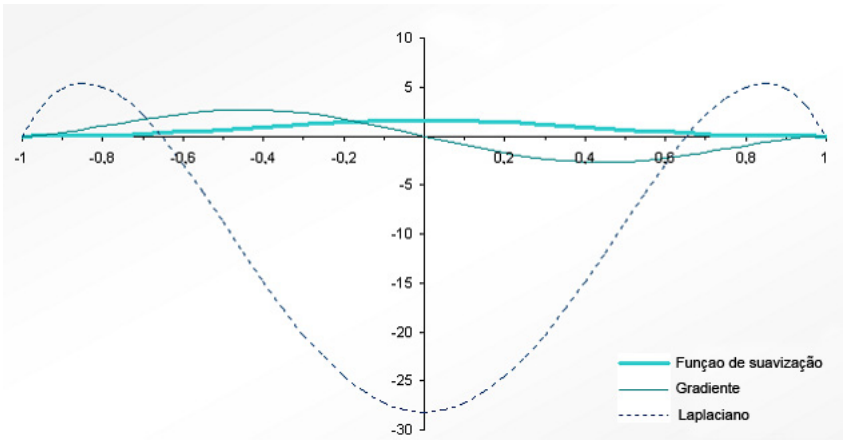

Figura 2.4: Gráfico da função de suavização utilizando h=1. Figura adaptada de [18].

A utilizacão desta função para o cálculo da força de pressão acarretaria no agrupamento de partículas, um comportamento indesejado para simulação de escoamento de fluidos. Para resolver este problema, [30] utiliza a função de suavização

$$W^{pressão}(r,h) = \frac{15}{\pi h^6} \begin{cases} (h-r)^3 & 0 \leq r \leq h \\ 0 & caso\ contrário, \end{cases} \tag{2.19}$$

com o gradiente definido por

$$\nabla W^{pressão}(\mathbf{r},h) = -\frac{45}{\pi h^6}\frac{\mathbf{r}}{\|\mathbf{r}\|}(h-\|\mathbf{r}\|)^2 \tag{2.20}$$

e Laplaciano definido por

$$\nabla^2 W^{pressão}(r,h) = -\frac{90}{\pi h^6}\frac{1}{r}(h-r)(h-2r), \tag{2.21}$$

cujo gráfico é apresentado na figura 2.5.

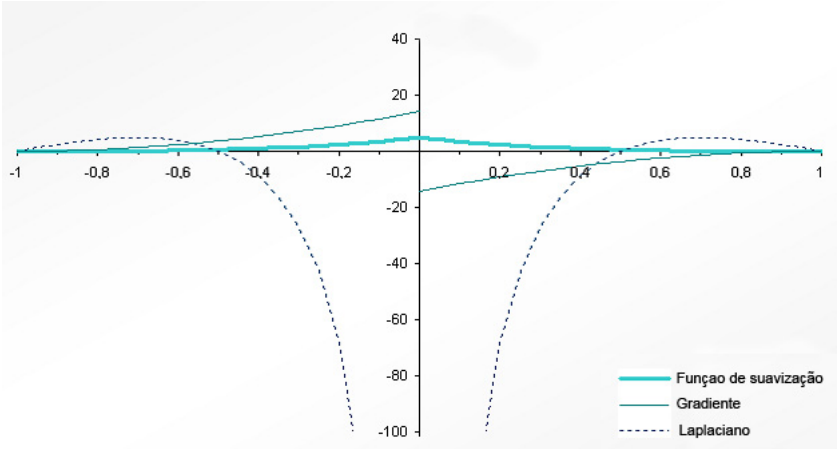

Figura 2.5: Gráfico da função de suavização utilizando h=1. Para uma melhor visibilidade, o gráfico possui escalonamento diferente. Figura adaptada de [18].

Finalmente, a última função de suavização para o cálculo da viscosidade foi desenvolvida após ser verificado que a utilização da equação (2.18) diminui a estabilidade da simulação. Com isso a função de suavização

$$W^{viscosidade}(r,h) = \frac{15}{2\pi h^3} \begin{cases} \frac{-r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & caso\ contrário \end{cases} \tag{2.22}$$

é utilizada, cujo gradiente é definido por

$$\nabla W^{viscosidade}(\mathbf{r},h) = \frac{15}{2\pi h^3}\mathbf{r}(-\frac{3\|\mathbf{r}\|}{2h^3} + \frac{2}{h^2} - \frac{h}{2\|\mathbf{r}\|^3}) \tag{2.23}$$

e Laplaciano definido por

$$\nabla^2 W^{viscosidade}(r,h) = \frac{45}{\pi h^6}(h-r). \tag{2.24}$$

A figura 2.6 apresenta o comportamento desta função de suavização.

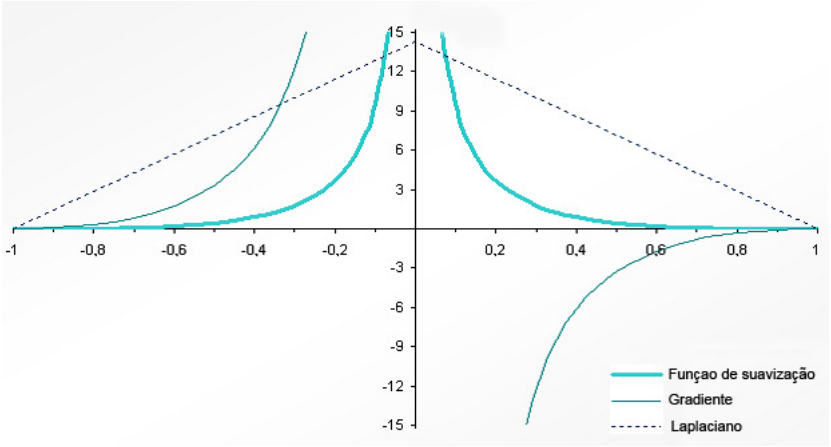

Figura 2.6: Gráfico da função de suavização utilizando h=1. Para uma melhor visibilidade, o gráfico possui escalonamento diferente. Figura adaptada de [18].

2.4.3 XSPH

Tradicionalmente o SPH permite que partículas movimentem-se livremente pelo domínio do problema, sem verificação de colisão, podendo, inclusive as mesmas assumirem a mesma posição. Desta forma, inconsistências numéricas podem surgir, visto que partículas na mesma posição podem possuir diferentes atributos tais como pressão e temperatura; ou divisão por zero na avaliação da equação (2.7). Monaghan [28] resolveu este problema desenvolvendo o XSPH, o qual pondera a velocidade de uma partícula com a dos seus vizinhos através da equação

$$\mathbf{v_i} = \varepsilon \sum_j \frac{m_j}{\rho_j}(\mathbf{v_i} - \mathbf{v_j}) W(\|\mathbf{r}_i - \mathbf{r}_j\|, h), \tag{2.25}$$

onde ε é uma constante no intervalo $0 \leq \varepsilon \leq 1$. Dessa forma, partículas tendem a mover-se com maior coerência, obtendo uma velocidade média entre as interações.

2.5 Trabalhos relevantes

A simulação do escoamento de fluidos pode ser dividida em simulações utilizando CPU e simulações que utilizam GPU. Primeiramente são abordados os trabalhos que utilizam CPU para simulação.

Um trabalho clássico na simulação de fluidos computacional foi proposto por Kass [17]. Nesse, os autores introduziram um método rápido e estável para simulação de água baseado nas equações de águas rasas, as quais são simplificações das equações de Navier-Stokes. Nesta abordagem, os autores utilizam uma função de altura para representar a superfície do fluido e colunas de fluido para modelar o fluxo e transporte de propriedades de elementos do fluido. A discretização do domínio é feita seguindo uma abordagem Euleriana, onde é aplicado o método de Euler implícito a fim de obter um método numericamente estável. Devido a abordagem proposta, não é possível efetuar a simulação de escoamento de fluidos turbulentos.

Karl Sims [44] utiliza uma arquitetura paralela para simulação de vários fenômenos físicos tais como fogo e queda d'água baseado na utilização de partículas e na aplicação da segunda lei de Newton para o movimento das partículas. Em seu trabalho, cada partícula é atribuída a um processador virtual, o qual é responsável pela dinâmica da partícula durante o seu tempo de vida. Na simulação da queda d'água, o autor simula a espuma gerada na queda variando a tonalidade da cor das partículas. A figura 2.7 ilustra o resultado obtido.

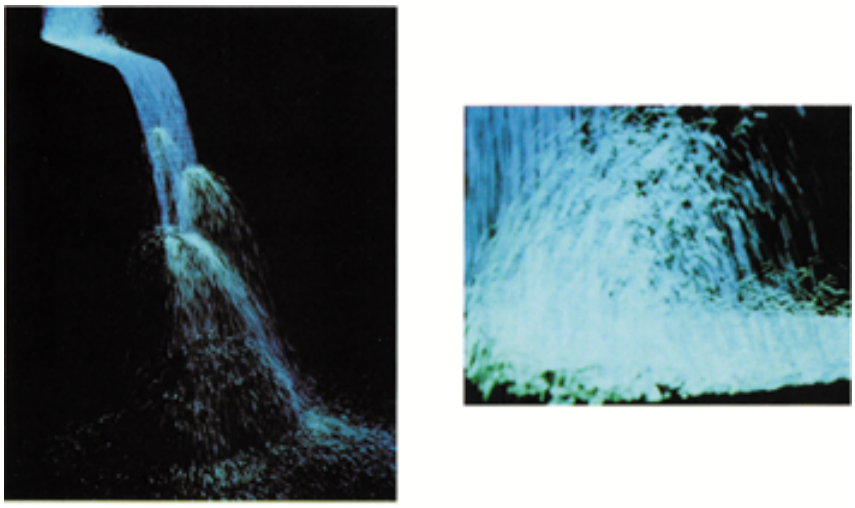

Figura 2.7: Simulação de queda d'água utilizando partículas [44].

O'Brien e Hodgins [37] estenderam a técnica apresentada por Kass [17] para permitir o borrifo e a geração de ondas durante a interação de objetos rígidos com a superfície do

fluido. Para alcançar este objetivo, foram desenvolvidos dois subsistemas. Um deles é responsável pela geração dos borrifos durante o impacto com corpos rígidos utilizando um sistema de partículas, caso a variação de altura de uma coluna de fluido ultrapassasse um valor limite. Para o tratamento das forças externas na superfície, um outro é responsável pela verificação de colisão entre objetos rígidos e a superfície do fluido, gerando forças para o sistema responsável pela simulação do volume do fluido.

Stam [45] pode ser considerado um dos trabalhos pioneiros na simulação de fenômenos físicos utilizando a equação de Navier-Stokes. Neste trabalho, o autor apresenta um método incondicionalmente estável para simulação de gases utilizando uma abordagem semi-Lagrangeana. O resultado alcançado pode ser visto na figura 2.8. Mais tarde, apresenta um trabalho [46] sobre simulação de fluidos utilizando equação da variação da densidade para jogos de maneira incondicionalmente estável.

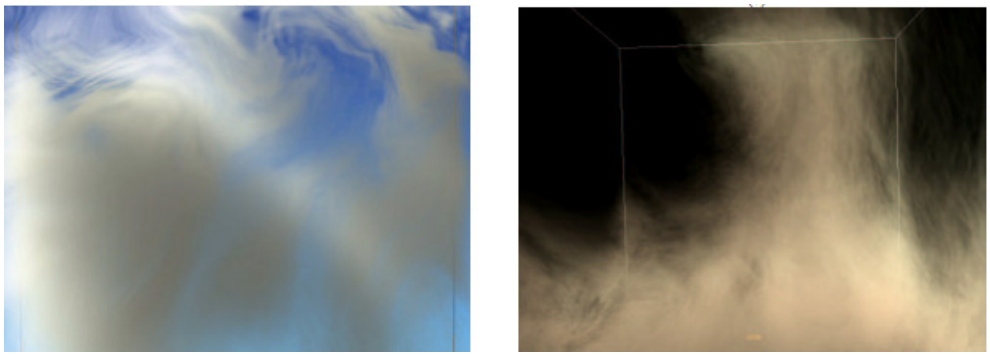

Figura 2.8: Simulação de gases utilizando método semi-Lagrangeano proposto em [45].

Desbrun [5] apresenta a utilização do método SPH para simulação de objetos deformáveis, com uma extensão em [47] para simulação de lava através do acoplamento da viscosidade à temperatura.

Müller [30] utiliza o método SPH para simulação de fluidos em tempo real, derivando as forças de densidade e viscosidade diretamente da equação de Navier-Stokes. O resultado desse trabalho é mostrado na figura 2.9. Em trabalhos futuros, Müller acrescenta interação de fluidos com corpos rígidos [31] para simulação de cirurgias virtuais. Para isso, Müller utiliza quadratura Gaussiana para distribuir partículas virtuais na superfície dos polígonos rígidos e utiliza o método de Movimento de Partículas semi-implícita para simulação do fluido. Em [32], Müller estende seu modelo para simulação de interação de estruturas fluido-fluido.

Figura 2.9: Enchimento de um copo d'água utilizando SPH [30].

Abordando simulações em GPU, Kipfer [19] utiliza SPH para simular o escoamento de fluidos em terrenos com deformações utilizando *shader*. Neste trabalho, interações com outros objetos na cena não são abordados.

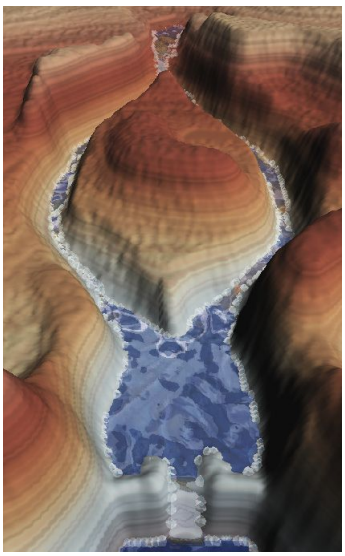

Figura 2.10: Simulação de escoamento de fluido sobre terrenos acidentados [19].

Posteriormente, Zhang [52, 53] também utilizou SPH para simulação e renderização de fluidos em GPU através do uso de *shader*.

Kurose [21] trata da interação entre fluidos e corpos rígidos através da discretização dos últimos em um conjunto de partículas. Após a verificação de colisão, as forças de contato são calculadas resolvendo o método de Problema de Complementaridade Linear e as devidas forças são aplicadas nos corpos rígidos e partículas de fluidos. O resultado alcançado pelo autor pode ser visto na figura 2.11.

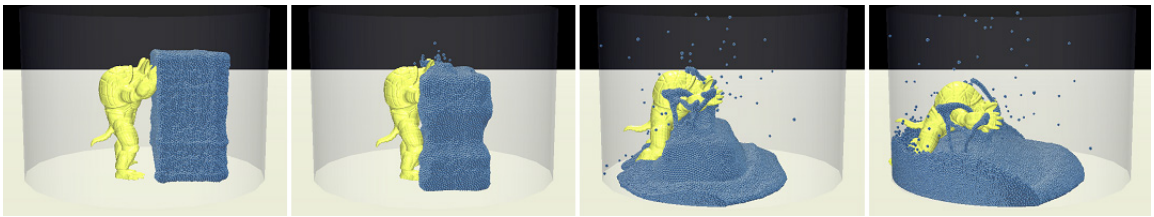

Figura 2.11: Interação de fluidos com corpos rígidos [21].

Capítulo 3

Simulação

Durante a simulação de fluidos computacional e corpos rígidos, um conjunto ordenado de tarefas precisam ser executados. Além disso, durante a interação de fluidos com corpos rígidos e destes com outros corpos rígidos, forças externas devem ser aplicadas a estes elementos. Neste capítulo são discutidas as etapas necessárias para simulação de fluidos (Seção 3.1) e corpos rígidos (Seção 3.2). Também é discutida a detecção de colisão entre fluidos e corpos rígidos (Seção 3.3) assim como a metodologia utilizada para representação de corpos rígidos (Seção 3.4) para o cálculo de colisão.

3.1 Simulação de fluidos

A simulação de fluidos utilizando o modelo SPH é realizada através da execução de um conjunto ordenado de tarefas. Cada partícula de fluido é representada por um estado com as informações e tipos presentes na tabela 3.1. O estado de cada partícula é atualizado e integrado conforme a metodologia proposta pelo SPH, através das tarefas exibidas na figura 3.1. Essas tarefas são discutidas detalhadamente nas próximas seções.

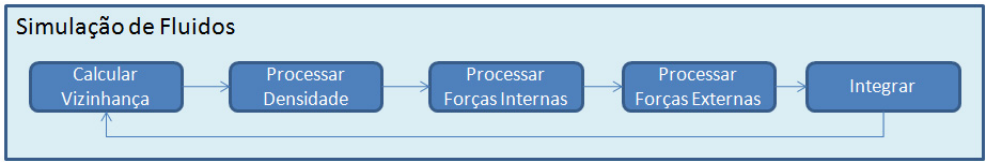

Figura 3.1: Tarefas necessárias para simulação de fluidos.

Tabela 3.1: Variáveis utilizadas na simulação de fluidos.

Variável	Tipo de Dado
Posição	vector4
Força	vector4
Velocidade	vector4
Densidade/Pressão	vector2

3.1.1 Cálculo da vizinhança

Para integração de variáveis em uma partícula, o SPH utiliza informações de um conjunto de partículas vizinhas dentro do suporte compacto h, assim como uma função *kernel* de suavização. Dessa forma, a primeira etapa durante o processamento de fluidos utilizando SPH é a localização das partículas vizinhas de cada partícula. Geralmente, nessa etapa utiliza-se uma estrutura de aceleração a fim de minimizar o esforço computacional durante a localização de partículas vizinhas.

3.1.2 Cálculo da densidade

Com as partículas vizinhas previamente localizadas, é efetuada a etapa de cálculo de densidade de cada partícula, o qual é necessário para o cálculo de todas as variáveis utilizando o método SPH, já que a contribuição de qualquer quantidade do fluido nas partículas é baseado no volume que a mesma ocupa no espaço, além de sua distância em relação a partícula processada.

3.1.3 Cálculo das forças internas

Com a densidade processada em cada partícula, é possível efetuar o cálculo da pressão utilizando a equação de estado apresentada por [5]. Além disso, essa etapa também é responsável pelo cálculo das forças viscosas presentes no fluido.

3.1.4 Cálculo das forças externas

Forças externas ao fluido são consideradas neste trabalho como aquelas relacionadas a gravidade e as forças geradas pelos corpos rígidos durante sua interação com os mesmos. Seu cálculo é independente do modelo SPH, podendo ser utilizado qualquer método que gerem forças repulsivas. A metodologia utilizada para o cálculo de forças externas neste trabalho é apresentada na seção 5.3.5.

3.1.5 Integração

Calculadas as forças internas e externas em cada partícula, essa etapa é responsável pela integração dessas forças para o cálculo da aceleração e velocidade. Devido ao movimento não rígido das partículas de fluido, cada partícula pode integrar sua posição independentemente das outras partículas, ou seja, pode ser executada em paralelo sem nenhuma dependência. A integração da velocidade e posição dessas partículas é realizada neste trabalho utilizando o método de Euler explícito.

3.2 Simulação de corpos rigidos

Corpos rígidos podem ser representados por uma coleção de partículas distantes de uma posição fixa do seu centro de massa. São caracterizados por movimentos rígidos, ou seja, essas partículas devem manter a mesma posição relativa ao seu centro de massa durante movimentos de translação e rotação. Dessa forma, para representar seu estado, é necessário apenas armazenar o estado da posição do centro de massa, representadas aqui utilizando as informações presentes na tabela 3.2 com seu respectivo tipo de dado.

Tabela 3.2: Variáveis e seus respectivos tipos utilizados na representação de corpos rígidos.

Variável	Tipo de Dado
Posição	vector4
Massa	float
Momento linear	vector4
Momento angular	vector4

Forças aplicadas em um corpo rígido devem ser aplicadas diretamente em seu centro de massa, variando sua velocidade e, consequentemente, posição, sendo considerado um movimento transacional. Além disso, a força aplicada pode gerar um torque, dependendo da sua posição relativa ao centro de massa, produzindo então, um movimento rotacional.

As forças aplicadas nos corpos rígidos consideradas neste trabalho são aquelas provenientes da gravidade e da interação dos corpos rígidos com os fluidos. A integração é realizada através da integração da velocidade e posição do centro de massa utilizando o método de Euler explícito.

3.3 Detecção de colisão

Um aspecto muito importante durante a simulação de fluidos computacional e corpos rígidos é o tratamento da interação entre estes com os elementos do ambiente no qual estejam inseridos, sejam elementos estáticos ou dinâmicos. Quando estas interações não são executadas de forma correta, a simulação pode apresentar comportamentos não verificados em uma simulação real, tal como interpenetrações de volumes. Dessa forma, quanto melhor for a técnica de tratamento de colisão utilizada, melhor será a precisão dessa interação quando comparadas à realidade, porém com exigência de um maior esforço computacional. A verificação de colisão entre corpos rígidos é tratada na literatura utilizando vários algoritmos. Uma lista extensa desses algoritmos pode ser encontrada em [29, 6].

A verificação de colisão entre objetos geralmente é feita através de duas fases: *broad phase* e *narrow phase*. Na *broad phase* é feita uma verificação grosseira de colisão entre objetos, utilizando geralmente uma malha mais simples daquela utilizada durante a renderização. Um algoritmo extensivamente utilizado nesta fase é chamado de *axis aligned bounding box* (AABB) onde é utilizado um retângulo, em duas dimensões, ou um paralelepípedo em 3 dimensões, envolvente do objeto. Caso o retângulo ou o cubo se intersectem, então pode ter ocorrido uma colisão e esses objetos passam para a *narrow phase*. Nesta, é feita uma verificação mais precisa de colisão entre os objetos cujos volumes envolventes colidiram na fase anterior, gerando informações como ponto de contato, normal do ponto de contato, distância de penetração, entre outras.

Com isso, fica claro a necessidade de representação de uma malha de um corpo rígido em malhas mais simples, a fim de acelerar o processo de verificação de colisão entre corpos rígidos com um menor esforço computacional.

Diversos métodos podem ser aplicados no tratamento de colisão entre corpos rígidos e fluidos, classificados em métodos analíticos ou geométricos. Enquanto que nos métodos analíticos os corpos rígidos são representados utilizando uma função matemática para representação da superfície, nos métodos geométricos sua representação é feita utilizando volumes, geralmente mais simples que a geometria de renderização, a fim de acelerar o processo de verificação de colisão. A avaliação de representações analíticas matemáticas são, em geral, executadas mais rapidamente, devido ao fato destas requererem apenas a avaliação de uma função. Nos métodos geométricos, existe a necessidade de avaliar a co-

lisão geometricamente, muitas vezes considerando um número muito grande de geometria na *narrow phase*.

Durante a simulação computacional de fluidos, este recebe o foco principal na maioria das vezes. Com isso o tratamento da interação entre fluidos e corpos rígidos deve ser o mais simples e eficiente possível, a fim de alocar um maior tempo de processamento à simulação do fluido. Neste caso, devido à sua maior eficiência, neste trabalho são utilizadas representações matemáticas de superfícies de objetos para o processamento da interação destes com os fluidos, bem como na interação que deve ser avaliada entre os próprios corpos rígidos. Com isso, além de um ambiente para a simulação da dinâmica de fluidos computacional, é necessário também um ambiente para a simulação de corpos rígidos, executado do forma concorrente ou paralela à simulação principal [2].

3.4 Representação de corpos rígidos

Durante a simulação de fluidos computacional e corpos rígidos, geralmente existe a necessidade do tratamento da interação entre estes e corpos rígidos. Com isso, representações eficientes e, ao mesmo tempo, representativas desses modelos, tanto de natureza analítica como geométrica, devem ser buscadas para sua simulação.

Em [9] o autor utiliza método Euleriano para a simulação de fluidos, enquanto utiliza o método Lagrangeano para simulação da interação dos mesmos com corpos rígidos. Neste trabalho o autor considera cada vértice do modelo como um ponto de massa ligado a vários outros pontos de massa, utilizando uma conexão elástica, considerando também o componente viscoso. Para o tratamento de colisão, são utilizadas somente primitivas implícitas.

A utilização da dinâmica de fluidos para simulação de explosões e sua propagação é apresentada em [51], onde o autor utiliza a metologia Euleriana para simulação do fluido. Corpos rígidos possuem duas formas de representação, sendo uma poligonal, utilizada para aplicação das forças provenientes do fluido e outra baseada em voxels, utilizando o volume do objeto para definir a interação dos corpos rígidos nos fluidos. A geração desses voxels é feita em tempo real, dependendo apenas da posição do corpo rígido.

3.4.1 Depth peeling

Métodos analíticos matemáticos para representação das superfícies de um objeto são utilizado neste trabalho devido à sua maior eficiência em relação ao tempo necessário para avaliação de colisão. Assim, a partir de um conjunto de polígonos de um corpo rígido, é extraída sua função implícita matemática adequada para sua representação.

Para efetuar esta discretização, é utilizada uma técnica chamada *depth peeling* [8] com algumas diferenças a serem discutidas posteriormente. *Depth peeling* foi originalmente uma técnica utilizada para o tratamento de objetos transparentes. Nesta técnica, a cena é renderizada N vezes, onde N é a resolução de profundidade desejada. Em cada passo de renderização i, a profundidade de cada pixel é armazenada na camada i, dependendo do seu valor e do armazenado na camada *i-1*. Após essa etapa, os objetos transparentes são renderizados N, vezes a fim de atribuir o valor correto à posição do pixel renderizado. Este algoritmo possui ordem de complexidade $O(N^2)$, sendo N também chamado de *depth complexity*.

A partir deste trabalho surgiram vários outros com a mesma ideia, porém, outras aplicabilidades. Em [49], o autor efetua vários passos de renderização a fim de gerar texturas classificadas em ordem de profundidade, ou *layered depth images* (LDI). Dessa forma, é possível obter os pixels mais próximos da câmera, depois os pixels mais próximos da câmera menores que os primeiros, depois os pixels mais próximos considerando o segundo plano de profundidade e assim sucessivamente, conforme mostrado na figura 3.2. Neste trabalho, o autor utiliza esta técnica para verificação de interseção entre um ponto e uma geometria, transformando o ponto do espaço 3D para o espaço de textura, normalmente compreendido entre o intervalo [0,1], para cada coordenada.

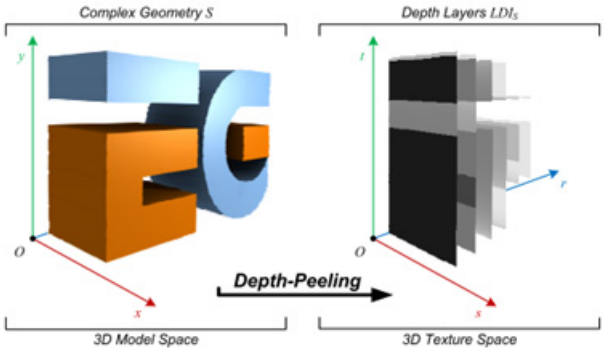

Figura 3.2: Um exemplo de geometria e seus mapas de profundidade. Extraído de [49].

3.4.2 Algoritmo de discretização

Nesta etapa, foi buscada uma solução que fosse eficiente para ser processada em paralelo pela GPU e, simultaneamente, permitir que objetos mais complexos pudessem ser utilizados na simulação. Para isso, um pré-processamento é efetuado na geometria do objeto a fim de gerar um conjunto de primitivas implícitas. Neste trabalho, a primitiva utilizada para discretização foi um conjunto de esferas de mesmo raio. Dessa forma, os seguintes benefícios foram obtidos:

- Utilização de métodos analíticos em geometrias complexas. Nesse caso, foi utilizado a equação da esfera, sendo esta equação uma das mais rápidas de ser avaliada matematicamente dentro do conjunto das equações implícitas disponíveis;
- A arquitetura CUDA é otimizada para um modelo de execução do tipo *uma instrução para múltiplas threads* [1], ou seja, uma instrução computacional comum é executada para um diferentes conjunto de dados. Nesse caso, o teste de colisão sempre será executado utilizando a mesma função matemática com diferentes tipos de dados (raios), obtendo uma maior performance, visto que não é necessário testes de colisão entre primitivas diferentes;
- O SPH utiliza partículas para simulação de fluidos. Estas partículas podem ser consideradas esferas primitivas, possibilitando que corpos rígidos possam também ser considerados um conjunto de partículas durante a simulação.

Porém, a utilização desta técnica para discretização de corpos rígidos em esferas apresenta as seguintes desvantagens:

- Corpos rígidos discretizados em esferas de raio elevado podem gerar interpenetrações de outros objetos discretizados com um raio menor, tal como as partículas de fluido. Nesse caso, deve-se escolher um valor de raio das partículas que não prejudique a detecção de colisão. Esse problema pode ser visto na figura 3.3, onde um corpo rígido é discretizado utilizando um raio muito superior ao raio da partícula de fluido.
- Apesar de funcionar muito bem para modelos complexos com grandes curvaturas, essa técnica é pouco eficiente para corpos rígidos que possuem grandes áreas planas alinhadas aos planos de projeção. Isso ocorre devido à perda de informação desses polígonos durante a projeção ortogonal, já que polígonos paralelos ao eixo de projeção não são renderizados em projeção ortogonal;

[1] Do inglês single instruction multiple thread (SIMT)

- Necessidade de uma etapa de pré-processamento da malha dos corpos rígidos utilizados durante a simulação.

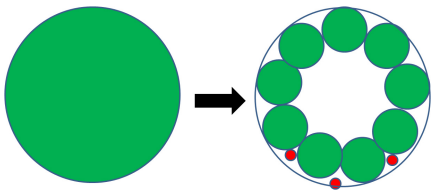

Figura 3.3: Esquerda: corpo rígido a ser discretizado; direita: corpo rígido discretizado utilizando raio elevado, causando interpenetração de partículas de fluido (em vermelho).

A discretização da geometria em partículas esféricas utiliza o *depth peeling* como base. Em [14], este método é utilizado para discretização da geometria durante o tratamento de colisão entre corpos rígidos, subdividindo seu volume em um conjunto de células ou *voxels* e obtendo-se, a partir daí, as células que pertencem à geometria do corpo rígido sendo discretizado, utilizando traçado de raios.

No método desenvolvido neste trabalho, o número de renderizações a ser executado já é conhecido a *priori*, antes de sua execução, baseado no raio R parametrizado na simulação. Além disso, aqui não é necessário o armazenamento da profundidade do fragmento do pixel renderizado na textura de saída. Somente uma *flag* é utilizada a fim de indicar que este fragmento pertence à geometria renderizada. Todo o controle dos fragmentos que devem ser considerados na geração dos voxels no layer L são executados pelo *fragment shader*, apresentado no algoritmo 1, implementado em linguagem *GLSL (OpenGL Shader Language)*.

Algoritmo 1: Código para extração do volume da geometria.

Entrada: float radius, float3 minBB, float3 maxBB, float layer
Saída: void
int numlayers = (int) ceil((maxBB.z - minBB.z) / (radius * 2.0));
float unitLayer ← 1.0 / (float) numlayers;
float unitRadius ← 1.0 / ((maxBB.z - minBB.z) / (radius * 2.0));
float currentLayer ← unitLayer * layer;
float layerEnd ← currentLayer + unitRadius;
se *(gl_FragCoord.z < currentLayer) ou (gl_FragCoord.z > layerEnd)* **então**
  discard;
fim
gl_FragColor ← float4(1.0, 1.0, 1.0, 1.0);

No algoritmo 1, a variável *layer* indica o layer a ser renderizado, enquanto que a variável *radius* indica o raio do voxel nos eixos X, Y e Z. Estas variáveis estão definidas

entre os valores *[0,1]* onde 0 indica o *near plane* e *1* indica o *far plane* do volume de visualização. As varíaveis *minBB* e *maxBB* são responsáveis por armazenar o *bounding box* do objeto.

Previamente à geração das informações de profundidade do objeto, é definida uma projeção capaz de capturar todas as informações deste objeto. Para isso, dado um objeto O que possui um *bounding box* O_{bb}, é definida uma projeção ortográfica que compreende os valores da tabela 3.3, transformando o modelo em um volume unitário compreendido entre os valores $[-1, 1]$, obtendo um bom aproveitamento da textura utilizada na renderização sem distorção da perspectiva.

Tabela 3.3: Especificação do volume de visualização.

Esquerda	Direita	Superior	Inferior	Plano Near	Plano Far
$O_{bb.min.x}$	$O_{bb.max.x}$	$O_{bb.max.y}$	$O_{bb.min.y}$	$O_{bb.max.z}$	$O_{bb.min.z}$

Um fator importante a ser considerado é o tamanho da memória de textura utilizada. Nesta metodologia, apenas um canal de cor é utilizado, ao contrário do método de *Depth Peeling* original, onde quatro canais de cor são necessários (rgba), visto que o mesmo é utilizado para o tratamento de pixels transparentes. Com essa modificação, é obtido um menor consumo de memória daquele utilizado no método original da ordem de 75 %.

No processo de geração dos voxels que representam o volume do objeto processado O na textura volumétrica O_{Tex}, as informações presentes nesta textura volumétrica precisam ser convertidas do espaço de textura para o espaço do mundo. Sabendo-se que na projeção ortográfica a distribuição do espaço do mundo para o espaço de textura é linear, esta conversão pode ser efetuada utilizando

areaPixels.x = (int) ceil(($O_{Tex.width}$ / ($O_{bb.max.x}$ - $O_{bb.min.x}$)) * R * 2.0f);
areaPixels.y = (int) ceil(($O_{Tex.height}$ / ($O_{bb.max.y}$ - $O_{bb.min.y}$)) * R * 2.0f);

Com o espaço do modelo subdividido em voxels de tamanho *2R*, o processamento de cada voxel é efetuado através do processamento do subvolume da textura que corresponde à posição do voxel, de tamanho *areaPixels*. Nesse processamento, verifica-se em cada texel do subvolume selecionado, se a *flag* de ativação está presente, indicando se este voxel faz parte do volume do objeto O. No momento em que a primeira *flag* é encontrada, o processamento deste subvolume pode cessar. Com isso, além da redução no consumo de memória de vídeo para o processamento deste método, existe a possível redução do número de *texels* a serem verificados, no melhor caso, de R * R * R, o que não acontece

quando é utilizado o método de *depth peeling* original. A figura 3.4 apresenta algumas imagens dessa transformação aplicando este processo.

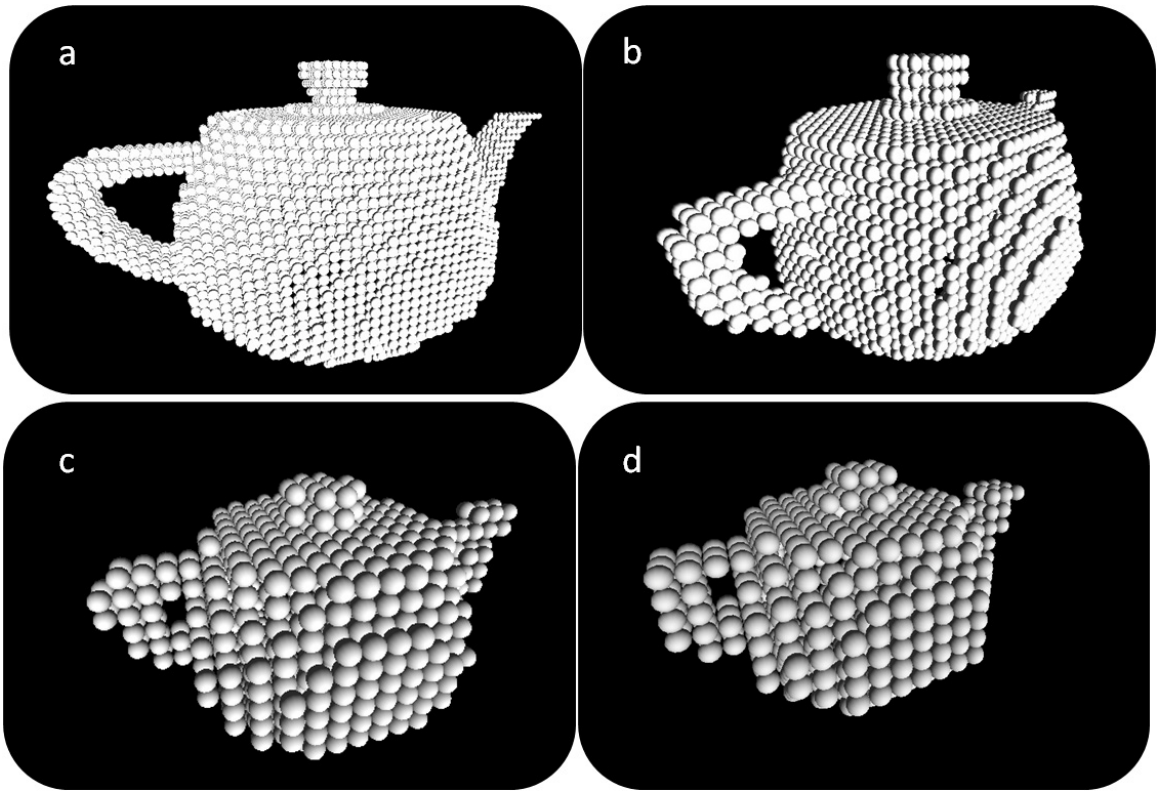

Figura 3.4: Processamento do bule utilizando vários raios como parâmetro: (a) raio 0.3 com 7337 partículas; (b) raio 0.5 com 2674 partículas; (c) raio 0.9 com 600 partículas; (d) raio 1 com 668 partículas.

Capítulo 4

Estrutura de aceleração

Neste capítulo é apresentada uma proposta de estrutura de aceleração utilizada para a simulação de fluidos e corpos rígidos estáticos e dinâmicos. Inicialmente é feita uma introdução aos métodos de subdivisão espacial (Seção 4.1) para posterior apresentação da estrutura de aceleração desenvolvida (Seção 4.2) com o objetivo de permitir a interação entre partículas de fluidos e corpos rígidos que encontram-se em espaços diferentes de memória de GPU e CPU. A seguir é apresentada a solução implementada para permitir a interação entre estruturas localizadas em diferentes espaços de memória (Seção 4.3). Posteriormente, é apresentada uma introdução à programação em GPU utilizando CUDA (Seção 4.4) e, finalmente, é apresentada a organização das informações necessárias para simulação de fluidos e corpos rígidos em memória (Seção 4.5), bem como sua complexidade.

4.1 Subdivisão espacial

O método SPH é um método baseado em partículas sem informação de conectividade entre as partículas vizinhas. Porém, a integração numérica é realizada utilizando o somatório das variáveis das partículas vizinhas dentro do suporte compacto, podendo estas partículas mudarem de posição com frequência durante a simulação. Utilizando força bruta para a localização das partículas vizinhas sem nenhuma estrutura de aceleração, é obtida uma complexidade de $O(n^2)$, onde n é o número de partículas total do sistema. Mesmo para um pequeno número de partículas, essa etapa pode exigir um esforço computacional muito grande, degradando consideravelmente o desempenho do sistema. Dessa forma, geralmente é empregada uma estrutura de aceleração com objetivo de diminuir esta complexidade.

Além disso, devido à representação proposta de corpos rígidos por meio de partículas, o número de interações entre partículas pode aumentar consideravelmente, dependendo da granularidade da discretização dos mesmos, bem como o número de corpos rígidos e partículas de fluido, visto que estamos tratando dupla interação entre fluidos e corpos rígidos.

Outro fator importante a ser considerado é o tamanho das partículas utilizado na simulação dos fluidos e aquelas utilizadas na discretização dos corpos rígidos, sendo as últimas geralmente maiores que as utilizadas na representação dos fluidos. Assim, deve-se escolher uma estrutura de aceleração que melhor acomode essas diferenças, apresentando melhor desempenho durante as buscas e utilizando baixo consumo de memória.

Na literatura, uma vasta gama de técnicas pode ser encontrada para estruturas de subdivisão espacial. Em [41] o autor apresenta a análise de implementação de vários métodos utilizados para subdivisão espacial. Como exemplos, podemos citar as estruturas de *octree*, *kd-tree* e *grade regular*.

Inicialmente, é estudada a viabilidade da utilização de uma estrutura de aceleração do tipo octree [39]. Esta estrutura é composta de nós que possuem, cada um, oito nós filhos. Cada nó filho é responsável por armazenar $\frac{1}{8}$ do volume representado pelo nó pai, sendo esta divisão geralmente realizada através da utilização dos planos definidos pelo sistema de coordenadas do espaço global. Com isso, é necessário o conhecimento, a *priori*, do domínio da simulação, impossibilitando que esta estrutura de aceleração seja utilizada fora do domínio inicial. Além disso, é desejável que cada nível desta estrutura seja representativo para a simulação, ou seja, cada nível de profundidade da árvore armazene dados que justifiquem seu processamento.

Quando se fala da simulação de fluidos, fala-se de partículas microscópicas chamadas *elementos de fluidos* [38] que são representadas utilizando um conjunto de esferas no método SPH. Com isso, a fim de manter uma relação entre o real e o computacional, o suporte compacto das partículas do SPH, h, deve ser escolhido cuidadosamente a fim de obter uma simulação fisicamente correta. Esse valor deve ser escolhido de forma que cada partícula tenha em média 5, 21 e 27 partículas vizinhas em uma, duas e três dimensões respectivamente [24]. Além disso, é necessário um número significativo de partículas para que a discretização das equações de Navier-Stokes apresente resultados numéricos satisfatórios [38].

Por outro lado, ao discretizar corpos rígidos em um conjunto de esferas de mesmo raio, verifica-se a necessidade de interação de elementos macroscópicos que devem interagir com

elementos microscópicos. Devido a esse fator, o processamento desses elementos em uma mesma estrutura de octree gera um grande número de níveis na estrutura que não serão utilizados, já que, dependendo da diferença entre os raios, existirão vários sub-níveis entre os níveis da árvore que as armazenam, desperdiçando tempo de processamento e, possivelmente, de memória. Além disso, as partículas utilizadas na simulação e na discretização do corpo rígido possuem tamanhos não variáveis ou uma variação controlada, como ocorre no caso de *level of detail* (LOD) de colisão. Nesse caso, a utilização de uma estrutura de aceleração desse tipo torna-se ineficiente para representação de geometrias que possuem volumes fixos distintos.

Devido à regularidade no tamanho das partículas que são utilizadas tanto na simulação do fluido quanto na discretização do corpo rígido em esferas, a utilização de uma grade regular apresenta maiores benefícios, pois permite que um número fixo de células seja processado por cada partícula durante a simulação. Dessa forma, a fim de possibilitar um espaço de simulação infinito, uma estrutura de aceleração baseada em um número fixo de células e uma função de hash [22], baseada na posição do objeto é empregada neste trabalho para efetuar o agrupamento de partículas de fluidos e corpos rígidos.

4.2 Definição da estrutura de aceleração

Durante a simulação, são utilizadas três tabelas de hash, diferentes e independentes. Uma tabela de hash é utilizada para classificar as partículas de fluidos, considerando um tamanho de célula de K_h, onde K_h representa o raio do suporte compacto utilizado para o cálculo de contribuição das partículas vizinhas. As partículas são inseridas na tabela com base em sua posição. Com isso, utilizando a figura de uma grade regular para representar as posições da tabela de hash, se faz necessário somente a verificação das partículas que se encontram na célula daquela partícula processada, além das células adjacentes, obtendo-se um total de vinte e sete células processadas em três dimensões, conforme se pode observar na figura 4.1 para uma espaço 2D.

Além desta tabela de hash, também é utilizada outra para representar os corpos rígidos estáticos em separado dos corpos rígidos dinâmicos, utilizando um tamanho de célula de R_r, onde R_r, representa o raio da esfera utilizada na discretização do corpo rígido. Essa separação de partículas de corpos rígidos estáticos e corpos rígidos dinâmicos foi efetuada baseando-se na frequência de atualizações, sendo executada somente no momento de inicialização para os corpos rígidos estáticos.

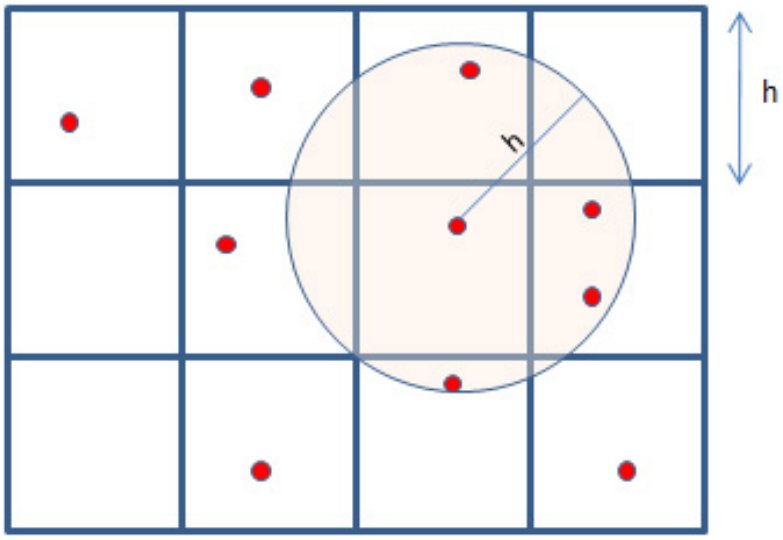

Figura 4.1: Configuração de célula utilizada no SPH.

Finalmente, utiliza-se também uma tabela de hash para os corpos rígidos dinâmicos, com as mesmas configurações daquela utilizada na representação de corpos rígidos estáticos, podendo variar somente o número de posições disponíveis na tabela de hash. Devido à natureza destes corpos rígidos, esta tabela é atualizada frequentemente com as novas posições dos corpos rígidos. Na algoritmo 2, é apresentado o método responsável pelo cálculo do índice na tabela hash. Esse algoritmo é baseado em [48], com ajuste de simetria, visto que na proposta original posições simétricas são mapeadas no mesmo código de hash, podendo gerar um número maior de partículas a serem verificadas durante a simulação.

Algoritmo 2: Método para geração de código de hash.

Entrada: float3 pos, float3 cell_size, int num_buckets, float3 world_limits
Saída: unsigned int hash_code
int x ← (int) ((pos.x + world_limits.x) / cell_size.x);
int y ← (int) ((pos.y + world_limits.y) / cell_size.y);
int z ← (int) ((pos.z + world_limits.z) / cell_size.z);
hash_code ← ((x * p1) *xor* (y * p2) *xor* (z * p3)) *mod* num_buckets;

No algoritmo 2, o parâmetro *world_limits* representa a aproximação da menor coordenada do mundo onde a simulação ocorre. *p1*, *p2*, *p3* são grandes números primos, usados aqui com os valores *73856093*, *19349663*, *83492791* [48], respectivamente.

4.3 Mapeamento entre estruturas de aceleração distintas

Devido à dupla interação entre corpos rígidos e fluidos, existe a necessidade de interação entre partículas de fluido e partículas utilizadas na representação dos corpos rígidos. Mais ainda, existe a necessidade de interação entre partículas localizadas em tabelas de hash que utilizam tamanhos de células diferentes localizadas em espaços de memória distintas.

A fim de solucionar este problema e permitir essa interação, é efetuado o mapeamento de posição entre tabelas distintas. Esse mapeamento é efetuado utilizando uma constante de proporcionalidade entre o tamanho da célula da tabela onde se encontra a posição a ser mapeada e o tamanho da célula da tabela para onde a posição será mapeada. Com essa constante de proporcionalidade, durante o mapeamento de uma posição de partícula em outra tabela, é especificado o número de células que devem ser processadas, conforme pode ser observado na imagem 4.2 para uma grade regular convencional, onde é feito o mapeamento de uma partícula de fluido para uma grade de corpos rígidos, cujo tamanho de célula é o dobro daquela na qual a partícula pertence. Nesse caso, se faz necessário a verificação do dobro de células durante o processamento desta partícula.

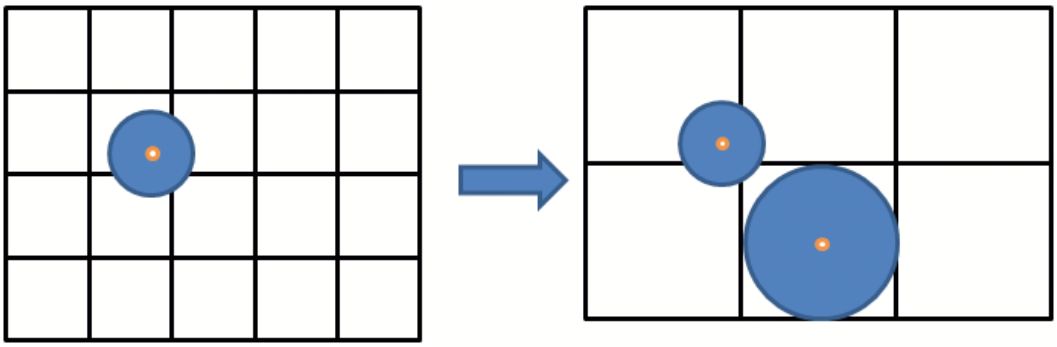

Figura 4.2: Mapeamento de uma esfera entre células de diferentes tamanhos.

A especificação dessa constante é dada através da fórmula $G_{FR} = Ceil(\frac{R_r}{K_h})$ para o mapeamento de partículas de fluido em células na tabela de corpos rígidos e $G_{CR} = Ceil(\frac{K_h}{R_r})$ para o mapeamento de partículas de corpos rígidos em células de tabela de fluidos. Com isso, um passo simples de processamento pode ser utilizado para efetuar a interação entre partículas de fluidos e corpos rígidos que encontram-se em diferentes tabelas de hash independentes na memória, como pode ser visto no trecho de algoritmo 3.

Com a utilização de várias tabelas de hash, é verificada uma melhor performance devido à utilização de uma tabela específica para cada caso, conforme observou-se nos

Algoritmo 3: Interação entre fluidos e corpos rígidos.

```
Entrada: float3 toCell_size, float ratio, int num_buckets
Saída: void
para (int x ← -ratio; x≤ ratio; x++) faça
    para (int y ← -ratio; y ≤ ratio; y++) faça
        para (int z ← -ratio; z ≤ ratio; z++) faça
            float3 map_pos ← float3(toCell_size.x * x, toCell_size.y * y,
            toCell_size.z * z);
            int bucket ← HashKey( map_pos, toCell_size, num_buckets);
            ProcessCell(bucket);
        fim
    fim
fim
```

resultados, pois permite que simplificações de inserção e busca sejam efetuadas de forma menos complexa daquela utilizando apenas uma tabela de hash.

4.4 Computação em GPU

Devido ao crescimento da capacidade de processamento de placas gráficas (GPU), as mesmas têm sido utilizadas para computação de propósito geral, ao invés de somente para processamento gráfico. Essa programação de propósito geral, que antes era realizada através de *shaders*, requerendo o conhecimento do pipeline gráfico, pode ser realizada de forma mais direta, sem o conhecimento do funcionamento do pipeline gráfico através da utilização da biblioteca CUDA, desenvolvida pela NVidia para suas placas gráficas.

4.4.1 CUDA - Compute Unified Device Architecture

CUDA [34] é uma arquitetura de computação paralela de propósito geral utilizada para resolver uma gama de problemas computacionais complexos não gráficos citenvidia:start em GPU.

Para sua programação, é utilizada a linguagem C, apesar de existirem *wrappers* que permitem a utilização de CUDA com outras linguagens. Programas desenvolvidos para funcionarem em CUDA precisam se encaixar no modelo de programação paralela imposta pela arquitetura CUDA. A programação em CUDA oferece suporte à programação heterogênea, onde parte do código pode ser executada em CPU, chamado *host*, e outra parte executada em GPU, chamada *device*, simultaneamente. CPU e GPU são tratadas como dispositivos distintos que possuem seu próprio espaço de memória.

Placas gráficas com suporte a CUDA possuem centenas de núcleos que podem executar milhares de threads em conjunto, utilizando recursos compartilhados como registradores e memória, evitando assim a necessidade de transferência de dados entre o barramento de sistema, o que costuma ser uma operação com alto custo computacional. Maiores informações sobre a arquitetura CUDA podem ser obtidas em [34].

Na figura 4.3 é apresentada a comparação, de alto nível, entre a arquitetura de GPUs e CPUs. Conforme observado, a GPU dispõe de todas as unidades da CPU, a saber: unidade de controles, ALUs (unidade lógica aritmética), cache e memória DRAM. Porém, grande parte da arquitetura de GPU é destinada à unidades lógica aritmética, o que torna a GPU capaz de executar muitos cálculos simultaneamente.

Figura 4.3: Arquitetura de CPU e GPU. Extraída de [36].

Comparando-se a capacidade de processamento em CPU e GPU, se pode observar na figura 4.4 que existe uma superioridade muito grande de desempenho entre estes processadores. Enquanto um processador Intel atinge alguns GFlops, uma GPU GTX480 pode atingir mais que 1.25 TeraFlops/s.

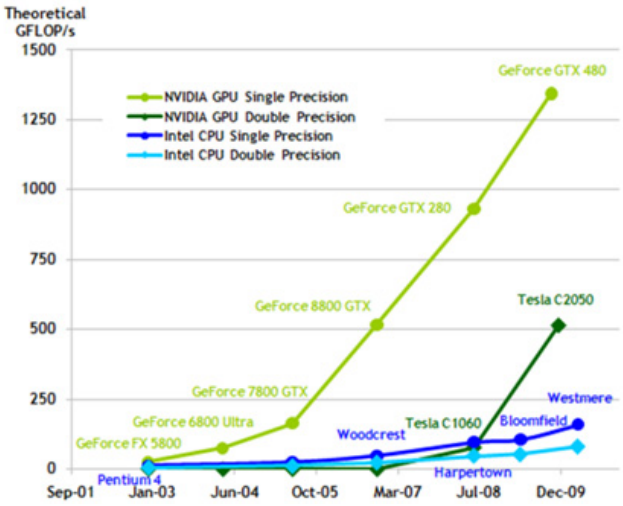

Figura 4.4: Comparação de desempenho entre CPU e GPU. Extraída de [36].

A organização de memória em GPU é bem diferente daquela realizada pela CPU. Dispositivos CUDA possuem diferentes espaços de memória que detém características particulares e influenciam diretamente no desempenho da aplicação. Geralmente, quanto menor a disponibilidade da memória, maior é sua velocidade de acesso. Essas memórias estão divididas em memória global, local, compartilhada, textura e registradores, conforme apresentado na figura 4.5. Maiores detalhes sobre cada tipo de memória é apresentado na tabela 4.1.

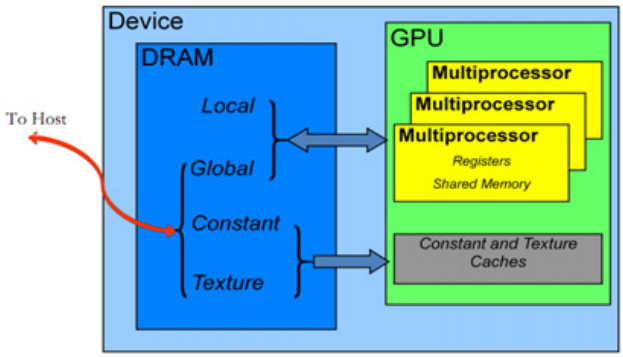

Figura 4.5: Organização de memória em GPU. Extraída de [36].

Programas que executam em GPU através da arquitetura CUDA são chamados *kernel*. Um *kernel* é executado por múltiplas threads durante o processamento, sendo a arquitetura CUDA chamada então de SIMT (Single Instruction, Multiple Threads).

4.5 Organização das informações de estado em memória

A fim de aumentar o desempenho do sistema, o acesso a memória global deve ser coalescido. Acesso coalescido é obtido quando a memória global é acessada em palavras de 32, 64 ou 128 bits. Com isso, a fim de obter acesso coalescido, geralmente é utilizada uma estrutura de array, com possível inserção de campos *dummy* para garantir este alinhamento.

O armazenamento das informações de estado das partículas de fluido em memória de GPU é realizada através da utilização de uma estrutura com quatro arrays, observado na figura 4.6. Os tipos de dados utilizados neste array garante a coalescencia durante seu acesso na memória global. Parâmetros globais como massa e viscosidade do fluido são armazenados em memória constante na GPU.

Tabela 4.1: Tipos de memórias disponíveis em GPU.

Memória	**Descrição**
Global	Possui um tempo de acesso grande se comparado com os demais tipos de memória. Pode ser utilizada para escrita e leitura sendo acessível globalmente por todos os núcleos. Não é on-chip
Local	Possui o mesmo tamanho e tempo de acesso equivalentes ao da memória global, não possui cache. Pode ser utilizada para escrita e leitura, sendo utilizada para salvar conteúdo de registradores quando não há espaço suficiente neles
Compartilhada	Possui um tamanho pequeno, apenas alguns KBytes por Streaming Processors (SM), porém possui um tempo de acesso muito baixo quando não há acesso simultâneo. É tão rápido quanto um registrador, possui cache, está alocada on-chip e threads do mesmo bloco podem acessá-la para cooperação
Textura	Possui tamanho igual à memória global, não está localizada on-chip, possui cache sendo otimizada para localidade espacial. Só é permitido leitura e está disponível para acesso por todas as threads e o *host*
Registradores	É o tipo de memória com o menor tamanho, localizada on-chip. Não possui cache, tendo seu acesso mais rápido que as outras. Acessar um registrador não consome nem um ciclo de clock por instrução

Corpos rígidos são representados através de partículas com raio constante. Além das informações de estado necessárias para sua simulação, também é necessário o armazenamento das k partículas que o definem, podendo este número ser diferente para cada corpo rígido. Essas partículas possuem uma posição relativa ao centro de massa do corpo rígido, conforme apresentado na figura 4.7. Para possibilitar a interação com outras partículas dentro do sistema, suas posições absolutas precisam ser calculadas a cada frame. Partículas de corpos rígidos estáticos inseridos na cena são armazenadas com posições absolutas, utilizando um array de dados em GPU. Além disso, são inseridas em uma tabela de hash diferente das partículas de fluido e de corpos rígidos dinâmicos. Isso se deve ao fato dessas partículas nunca precisarem ser atualizadas durante a simulação.

Partículas de corpos rígidos dinâmicos possuem posições que variam com a posição e orientação do centro de massa do corpo rígido ao qual pertencem. Dessa forma, a fim de evitar um grande tráfego de informações entre GPU e CPU, as posições relativas das partículas de corpos rígidos dinâmicos são armazenadas na GPU, assim como a posição do centro de massa do corpo rígido e sua orientação. A cada interação, a posição absoluta

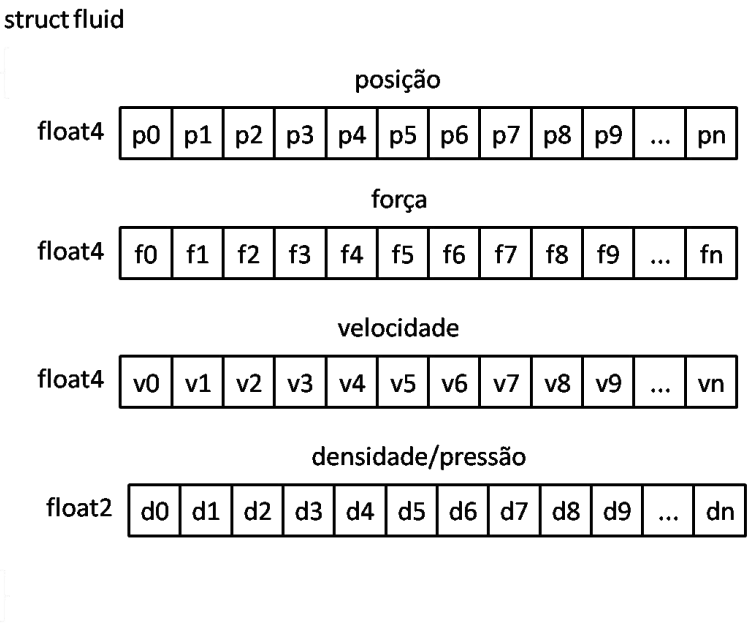

Figura 4.6: Organização do estado das partículas de fluido em GPU.

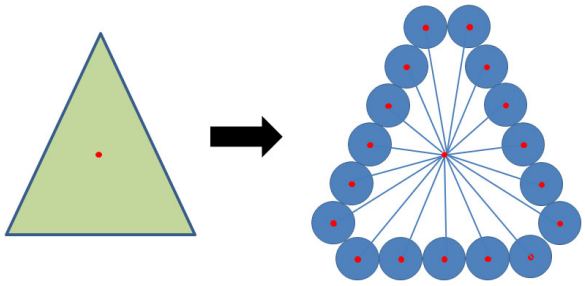

Figura 4.7: Relação entre as partículas e o centro de massa do corpo rígido.

de cada partícula é calculada, baseada na posição do centro de massa de seu respectivo corpo rígido.

Cada partícula p_x pertencente ao corpo rígido Y necessita de acesso às informações de orientação RY_{ori} e posição do centro de massa RY_{cm} do seu respectivo corpo rígido. Pode-se observar através da figura 4.7 que existe um relacionamento $N \rightarrow 1$ entre as partículas e seu respectivo corpo rígido. Além disso, um corpo rígido A é discretizado em um conjunto de P_A partículas, o que não é garantido ser o mesmo número para todos os corpos rígidos na simulação. Como essas partículas encontram-se na GPU, uma solução para esse problema seria a duplicação dessas informações para cada partícula, exigindo um maior consumo de memória. Porém, outra solução é adotada neste trabalho. A coordenada de posição w relativa de cada partícula armazena o índice do corpo rígido ao qual esta partícula pertence. Esse índice é utilizado para localizar as informações de posição de centro de massa e orientação de seu respectivo corpo rígido durante a transformação da

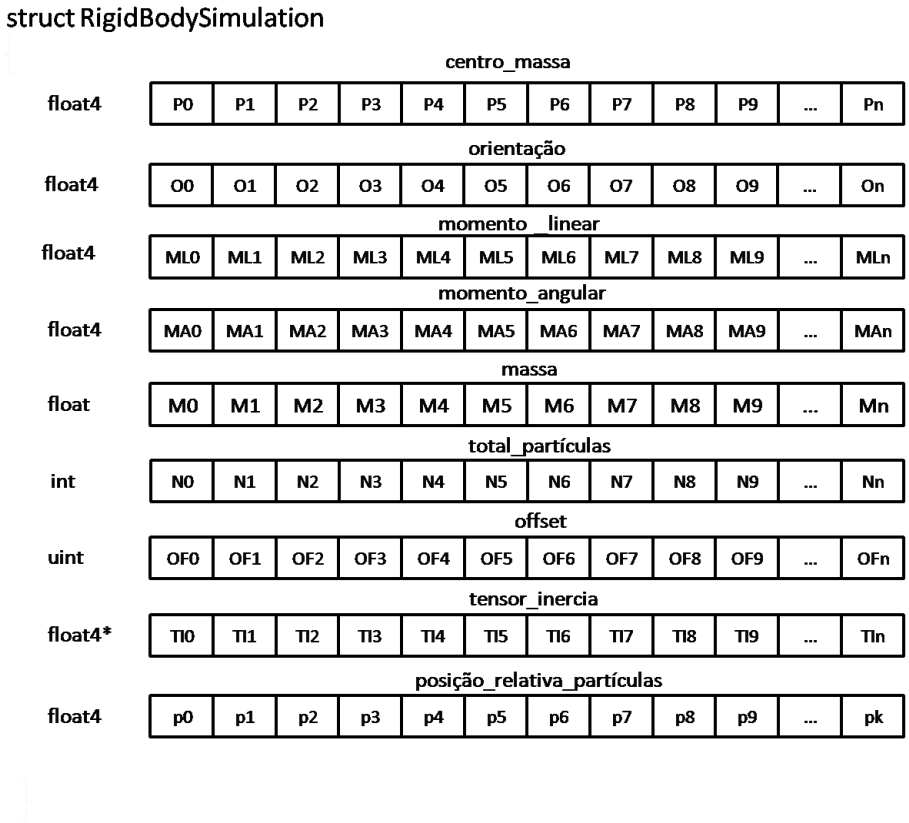

Figura 4.8: Estado dos corpos rígidos armazenados em GPU.

partícula para posição absoluta, o que evita duplicação de dados e, consequentemente, menor consumo de memória. A organização das informações de estado dos corpos rígidos em memória na GPU é apresentada na figura 4.8.

Capítulo 5

Ambiente heterogêneo (CPU-GPU)

Neste capítulo é apresentado um modelo de distribuição de tarefas para processamento de fluidos e corpos rígidos com dupla interação, utilizando ambiente heterogêneo de GPU em conjunto com vários núcleos de CPU. Para evitar a ociosidade entre os processadores, este modelo é desenvolvido utilizando baixo acoplamento entre tarefas de GPU e CPU. Para isso é necessário a identificação das tarefas e suas dependências (Seção 5.1). Após esta identificação, é apresentado o modelo desenvolvido (Seção 5.2), bem como sua distribuição entre os processadores durante a simulação e o detalhamento destas para simulação de fluidos e corpos rígidos (Seção 5.3).

5.1 Identificação das tarefas

Durante a simulação de fluidos e corpos rígidos, tarefas bem definidas podem ser observadas, bem como sua ordem de execução. Na utilização em simulações de fluidos e corpos rígidos onde não ocorre interação entre estes, o processamento de tarefas inerentes à simulação do escoamento do fluido poderia ser executado concorrentemente com o processamento de tarefas inerentes à simulação de corpos rígidos. Esse paralelismo poderia ser obtido em GPU utilizando uma arquitetura FERMI [1] e em CPU utilizando um processador com vários núcleos. Porém, o modelo desenvolvido neste trabalho considera a interação entre fluidos e corpos rígidos. Dessa forma, se pode identificar algumas dependências durante o processamento das tarefas de fluidos e corpos rígidos, conforme pode ser observado na figura 5.1, através das setas tracejadas.

[1] http://www.nvidia.com/object/fermi_architecture.html

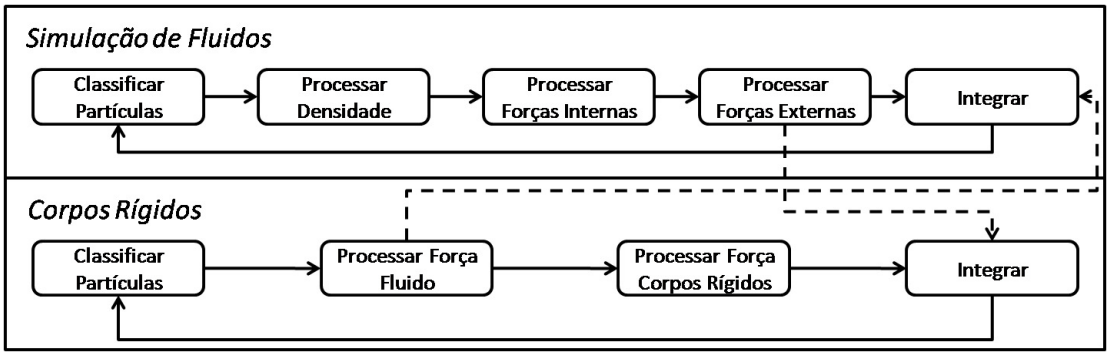

Figura 5.1: Dependência entre as tarefas, representada pelas setas tracejadas.

Utilizando uma arquitetura heterogênea de GPU e CPU, além da dependência de dados existente durante o processamento destas tarefas, também deve ser resolvida a distribuição das mesmas nestes processadores de forma a minimizar ao máximo qualquer ociosidade que possa ocorrer. Para efetuar este desacoplamento de tarefas, as informações utilizadas do fluido na simulação de corpos rígidos e dos corpos rígidos na simulação de fluidos são obtidas do *frame* anterior, técnica similar ao *double buffer* em computação gráfica. Nesse caso, durante o processamento da simulação de fluidos e corpos rígidos, informações tais como posição, velocidade e orientação são armazenadas em um espaço de memória diferente daquele acessado pela simulação de fluidos e de corpos rígidos.

5.2 Modelo de tarefas

A GPU possui uma arquitetura *SIMT*, ou seja, os dados de entrada são processados simultaneamente por várias threads que executam a mesma operação sobre esses dados. Com isso, as tarefas atribuidas à GPU devem seguir esse modelo de execução, a fim de obter melhor performance.

Por sua vez, o paralelismo da CPU é inferior se comparado ao da GPU, ou seja, o número de *threads* simultâneas que podem ser executadas em GPU é bem superior ao daquelas da CPU, devendo esse fato ser levado em consideração durante a atribuição de tarefas entre estes processadores.

As simulações de fluidos e corpos rígidos em trabalhos apresentados na literatura utilizando GPU [53, 52, 21, 14] efetuam o processamento de todas as tarefas identificadas na figura 5.1 a cada *frame*, independentemente destes estarem interagindo ou não durante seu processamento.

Porém, conforme podemos observar na figura 5.2, pode-se identificar três fatos importantes que permite simplificar ou até excluir a necessidade de processamento de algumas destas tarefas durante o processamento de cada *frame*: (1) existem corpos rígidos que não colidem com corpos rígidos; (2) existem corpos rígidos que não colidem com fluidos e (3) existem corpos rígidos que não colidem com fluidos e outros corpos rígidos. Com esta observação, um modelo de tarefas mais eficiente é apresentado na figura 5.3.

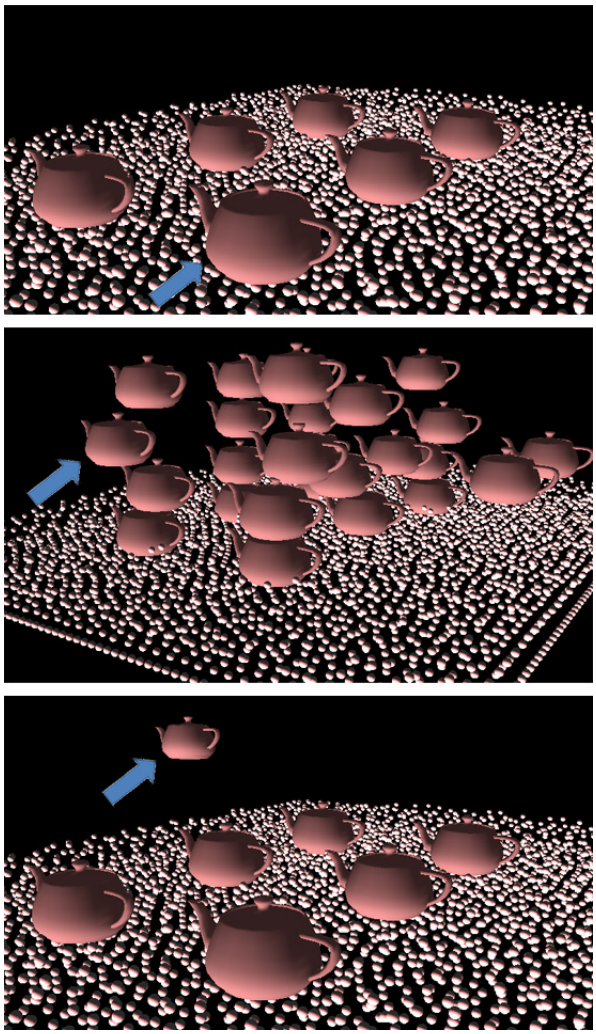

Figura 5.2: Comportamentos durante a simulação. De cima para baixo: corpo rígido sem interação com outro corpo rígido; corpo rígido sem interação com fluido e corpo rígido sem interação com fluido e outro corpo rígido.

No modelo apresentado na figura 5.3, em alguns casos, pode-se omitir o processamento de determinadas tarefas durante cada *frame* da simulação. Além disso, as interações entre corpos rígidos e fluidos ou entre corpos rígidos somente precisam ser processadas para aqueles que potencialmente colidiram.

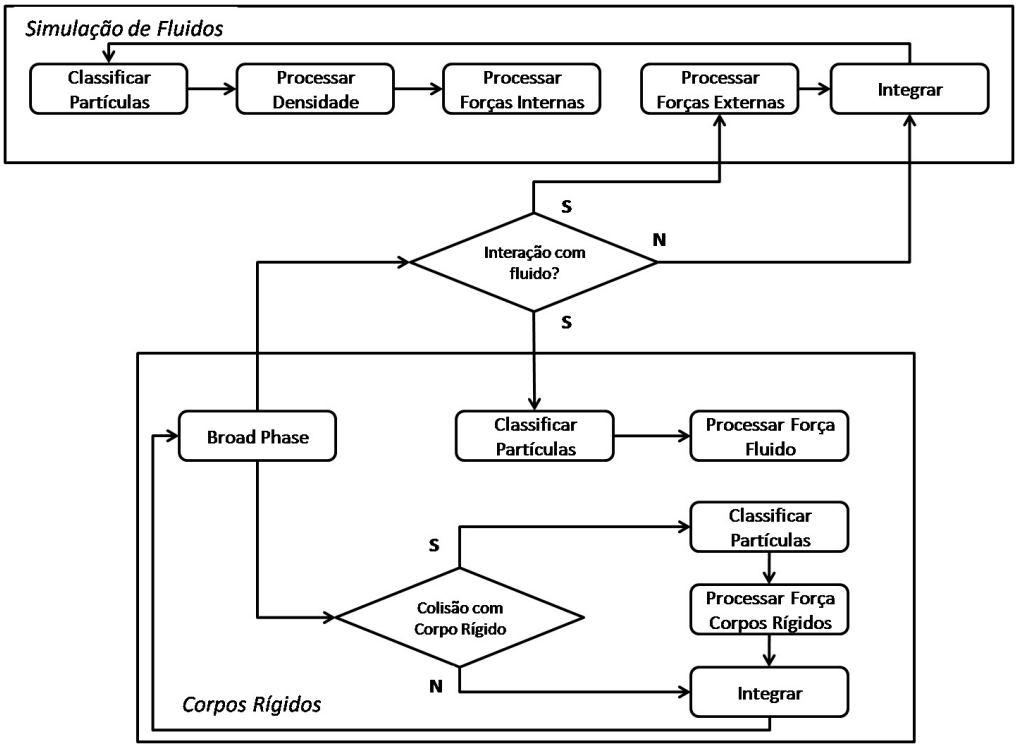

Figura 5.3: Modelo implementado para execução da simulação de corpos rígidos e fluidos com dupla interação.

Dessa forma, baseado no modelo apresentado, a figura 5.4 apresenta a distribuição de tarefas a serem processadas em GPU e CPU, cujo funcionamento será detalhado na próxima seção.

5.3 Distribuição de tarefas

Com o objetivo de manter o processamento contínuo em ambos os processadores, o processamento das tarefas está dividido em quatro blocos, exibidos na figura 5.4.

No modelo apresentado, o *bloco 1* inicia a classificação das partículas de fluido enquanto que no *bloco 3*, ao mesmo tempo, a CPU inicia a etapa de *broad phase*. Após esta verificação, em caso de possível colisão, o *bloco 2* é iniciado, classificando partículas que colidiram com o fluido e partículas que colidiram com corpos rígidos.

Após a etapa de *broad phase*, as tarefas do *bloco 4* são executadas de acordo com o resultado do teste de colisão executado. Somente partículas de corpos rígidos que potencialmente colidiram são avaliadas, ao contrário de vários trabalhos que utilizam discretização por partículas [1, 21, 14], os quais enviam todas as partículas presentes na cena para processamento de colisão em GPU.

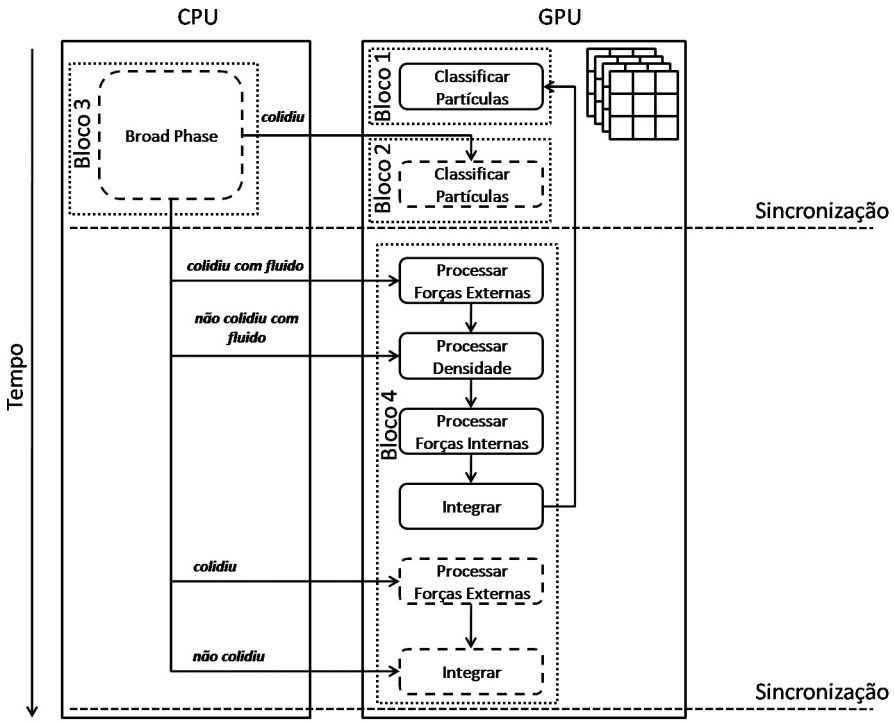

Figura 5.4: Diagrama de distribuição de tarefas entre GPU e CPU. No bloco da GPU, em linhas cheias, as tarefas inerentes à simulação de fluidos, enquanto em linhas tracejadas as tarefas inerentes à simulação de corpos rígidos.

Nas seções seguintes, estas tarefas serão apresentadas mais detalhadamente.

5.3.1 Classificação de partículas

A etapa de classificação de partículas é responsável pelo agrupamento das partículas em suas respectivas células, utilizando uma função de *hash* baseada em sua posição.

Para a simulação de fluidos, esta tarefa deve ser executada a cada *frame*, visto que métodos Lagrangeanos necessitam de informações das partículas vizinhas para a computação de variáveis tais como pressão e densidade.

Porém, para simulação de corpos rígidos, suas partículas somente são utilizadas para o cálculo de colisão, sendo necessário sua classificação somente nos casos onde ocorram colisão com fluidos ou outros corpos rígidos.

Para sua classificação, primeiramente é calculado o seu código de hash em GPU, conforme algoritmo 2, gerando um array de tamanho N, onde N é o número de partículas. Após o cálculo desse código para cada partícula, é utilizado a função de *sort* para ordenação deste array, sendo necessário também um array auxiliar indicando o índice da partícula. Durante essa ordenação, esse array auxiliar também é alterado, possibilitando

assim que a partícula seja corretamente localizada. Posteriormente, de posse deste array ordenado, a etapa final efetua a localização inicial e final do *offset* de cada célula de hash.

5.3.2 Calculo de densidade

Após a classificação de partículas, deve-se calcular a densidade, utilizando as informações processadas previamente para a localização das partículas vizinhas. Devido à dependência do campo de densidade para os cálculos posteriores, essa etapa deve ser executada em um *kernel* diferente durante o processamento do SPH. O processamento da interação com cada partícula se dá através da verificação com as partículas em sua célula e as partículas nas células adjacentes, através do algoritmo 4. Assim, substitui-se a função $Func$ neste algoritmo pela função exibida no algoritmo 5 para efetuar o cálculo de densidade para cada partícula.

Algoritmo 4: Processamento de células.

Entrada: float4* positions, float3 cellsize, int partIndex
Saída: void
para $z \leftarrow -1$ **to** 1 **faça**
  para $y \leftarrow -1$ **to** 1 **faça**
    para $x \leftarrow -1$ **to** 1 **faça**
      float3 offset $\leftarrow float3(cellsize.x * x, cellsize.y * y, cellsize.z * z)$;
      int hash $\leftarrow HashKey(positions[partIndex] + offset)$;
      $Func(partIndex, hash)$;
    fim
  fim
fim

Algoritmo 5: Processamento da densidade de uma partícula de fluido.

Entrada: float4* positions, int partIndex, int hashIndex, float2 massDensity, uint* startOffset, uint* endOffset
Saída: void
se *startOffset[hashIndex]* $\neq$ *MAX_UINT* **então**
  para $i \leftarrow$ *startOffset[hashIndex]* **to** *endOffset[hashIndex]* **faça**
    uint piIdx $\leftarrow$ startOffset[i];
    float4 posI $\leftarrow$ positions[piIdx];
    float density $\leftarrow$ Density(positions[partIndex], posI);
    massDensity[partIndex] $\leftarrow$ massDensity[partIndex] + density;
  fim
fim

A função *Density* efetua o cálculo da densidade entre duas partículas utilizando a equação (2.5).

5.3.3 Cálculo das forças internas

Nesta etapa do processo, são calculadas as forças internas do fluido, através do algoritmo 6, o qual é substituído na função *Func* no algoritmo 4. A função *CalcPressure* neste algoritmo é responsável pelo cálculo da força de pressão, utilizando a equação (2.11). A força de viscosidade é calculada na função *CalcViscosity*, utilizando a equação (2.14).

Algoritmo 6: Processamento das forças de pressão e viscosidade de uma partícula de fluido.

```
Entrada: float4* positions, int partIndex, int hashIndex, float2 massDensity,
         uint* startOffset, uint* endOffset, float4* velocity, float viscosity,
         float4* totalForces
Saída: void
se startOffset[hashIndex] ≠ MAX_UINT então
    para i ← startOffset[hashIndex] to endOffset[hashIndex] faça
        uint piIdx ← startOffset[i];
        float4 posI ← positions[piIdx];
        float dsI ← massDensity[piIdx].x;
        float ds ← massDensity[partIndex].x;
        float4 velI ← velocity[piIdx];
        float4 vel ← velocity[partIndex];
        float4 pressure ← CalcPressure(positions[partIndex], posI, ds, dsI);
        float4 visco ← CalcViscosity(positions[partIndex], vel, VelI, posI, dsI,
        viscosity);
        totalForces[partIndex] ← totalForces[partIndex] + pressure + visco;
    fim
fim
```

5.3.4 Broad phase

A simulação de fluidos e corpos rígidos com dupla interação requer acesso a espaços de memória distintos e independentes. Mesmo utilizando uma estrutura de aceleração para evitar a complexidade exponencial, esta interação ainda pode se apresentar custosa para processamento, visto o número de partículas de fluido e corpos rígidos que necessitam de processamento.

Durante a simulação de fluidos e corpos rígidos com dupla interação, os corpos rígidos são discretizados em partículas. Estas partículas são utilizadas somente para efeito de colisão, sendo desnecessário o seu processamento para corpos rígidos que não apresentem colisão com fluido e/ou outros corpos rígidos. Assim, nesta etapa, verifica-se corpos rígidos que colidiram com outros corpos rígidos e/ou fluidos utilizando a técnica de *bounding*

box. Essa verificação é executada em paralelo pelo número de núcleos disponível na CPU, utilizando a biblioteca *OpenMP* [2], e os *flags* de colisão armazenados em um array, apresentado na figura 5.5.

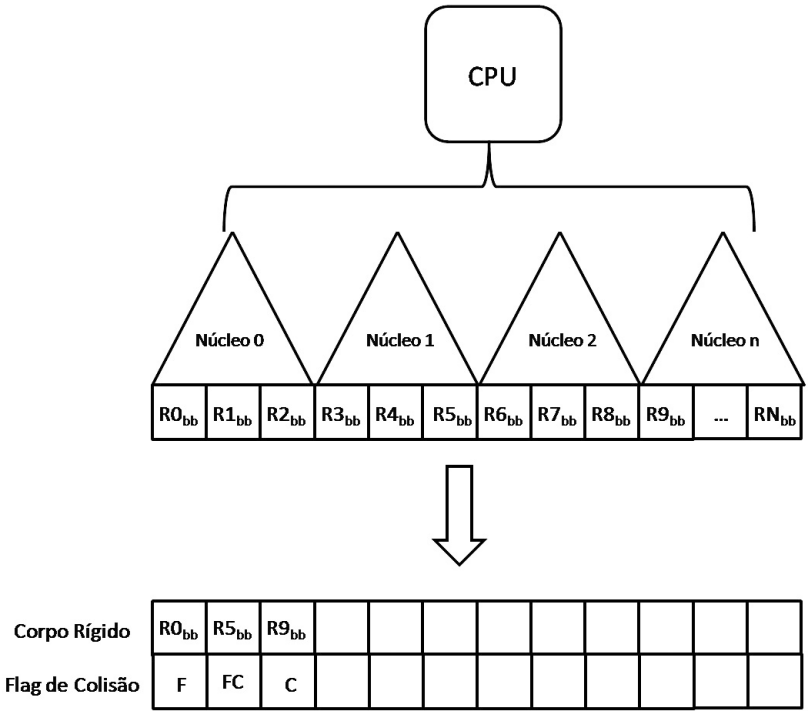

Figura 5.5: Execução da *broad phase* utilizando múltiplos núcleos em CPU. Rk_{bb} representa o *bounding box* do corpo rígido k. F representa colisão com fluido e C com corpo rígido.

Para este processamento, é utilizado o algoritmo 7, paralelamente em múltiplos núcleos da CPU.

Após este processamento, tem-se um array indicando os corpos rígidos que potencialmente colidiram assim como os *flags*, indicando se colidiram com outros corpos rígidos e/ou fluido. Essa identificação é importante, pois permite que o processamento do fluido prossiga caso não exista nenhuma interação com corpos rígidos. Neste caso, apenas as forças internas e gravitacional precisam ser processadas. Esta pré-seleção é bastante importante pois verifica-se que, em muitas simulações, o fluido ocupa uma pequena parte do espaço em comparação com os corpos rígidos. Além disso, pode-se ter uma uma pilha de corpos rígidos sobre um fluido, ocorrendo interação somente dos fluidos que encontram-se na base desta pilha, podendo-se observar este caso na segunda imagem da figura 5.2.

Utilizando este array de *flags*, são gerados dois outros arrays, caso necessário: um array contendo as posições absolutas das partículas de corpos rígidos que colidiram com

[2]Disponível em http://www.openmp.org

Algoritmo 7: Broad phase em multiplos núcleos de CPU.

```
Entrada: int numThreads, int numElements, int currentThread, float2* arrayflag,
         rigidbody* rb, boudingbox* fluid
Saída: void
int elementsPerThread = (int) (ceil((float)numElements / (float)numThreads));
int startElement ← currentThread * elementsPerThread;
int endElementz;
se currentThread == numThreads - 1 então
  | endElement ← numElements;
senão
  | endElement ← startElement + elementsPerThread;
fim
para i ← startElement to endElement faça
  | para j ← 0 to numElemens faça
  |   | se rb[i] colidiu rb[j] então
  |   |   | Setar arraryflag[i] para C;
  |   |   | Incrementar contador;
  |   | fim
  |   | se rb[i] colidiucom fluid então
  |   |   | Setar arraryflag[i] para F;
  |   |   | Incrementar contador;
  |   | fim
  |   | se rb[i] colidiu rb[j] e rb[i] colidiu fluid então
  |   |   | Setar arraryflag[i] para FC;
  |   |   | Incrementar contador;
  |   | fim
  | fim
fim
```

outros corpos rígidos e um outro array com as mesmas informações, porém, contendo partículas de corpos rígidos que colidiram com fluidos. Também são gerados um array com o número de partículas de cada corpo rígido e offset, responsável pela localização das partículas de cada corpo rígido no array de posição, conforme pode ser observado na figura 5.6.

Após esta etapa, somente as partículas de corpo rígido que potencialmente possam colidir são processadas, efetuando-se a classificação e dupla interação entre fluidos e corpos rígidos.

5.3.5 Cálculo das forças externas

Durante esta etapa, são calculadas as forças de fluidos nos corpos rígidos e destes nos fluidos somente para as partículas de corpos rígidos que foram selecionadas na etapa anterior.

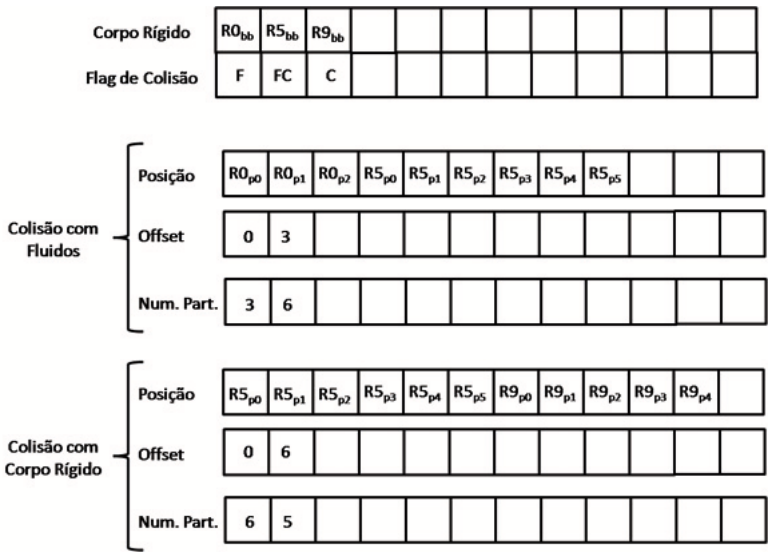

Figura 5.6: Informações geradas após o processamento da etapa de *broad phase*. Partículas que colidiram com corpos rígidos são armazenadas independentemente daqueles que colidiram com fluidos.

Além disso, a interação de fluidos com corpos rígidos será processada independentemente da interação entre corpos rígidos.

Para o cálculo das forças entre estas partículas, utiliza-se o método de elementos discretos (DEM), utilizado para simulação de materiais granulares [27] tal como areia. A força repulsiva f_{ij}, agindo na partícula i através da interação com uma partícula j é modelada utilizando uma força de elasticidade $f_{i,s}$

$$f_{i,s} = -k(t_{rad} - \|\mathbf{r_{ij}}\|)\frac{\mathbf{r_{ij}}}{\|\mathbf{r_{ij}}\|} \tag{5.1}$$

e uma força de amortecimento $f_{i,d}$

$$f_{i,d} = \eta \mathbf{v_{ij}}, \tag{5.2}$$

onde k é o coeficiente de elasticidade, η é o coeficiente de amortecimento, $\mathbf{r_{ij}}$, $\mathbf{v_{ij}}$ e t_{rad} representa a distância relativa, velocidade relativa, e soma dos raios das partículas i e j, respectivamente.

Durante a interação de partículas de fluido com partículas de corpos rígidos ou entre estes, as forças geradas devem obedecer a terceira lei de Newton, a qual afirma que para toda ação resulta uma reação de mesma intensidade, porém direções opostas. Dessa forma, o mesmo cálculo de forças apresentado é utilizado durante a interação de fluidos com corpos rígidos e entre corpos rígidos. Assim, cada partícula de corpo rígido armazena o total de forças acumulada para posterior integração.

A utilização de corpos rígidos discretizados em partículas permite o cálculo das forças aplicadas fora do centro de massa com maior facilidade, pois não existe a necessidade de cálculos para a localização do ponto exato da colisão, como acontece quando se utiliza triângulos.

Um aspecto importante a ser notado aqui é o conceito de pares de corpos rígidos que são gerados durante a colisão. Geralmente, durante o teste de colisão, quando o corpo rígido A colide com um corpo rígido B, não é necessário a verificação, posteriormente, de colisão entre o corpo rígido B e o corpo rigido A. Dessa forma, para evitar este processamento desnecessário, geralmente utiliza-se uma matriz triangular a fim de efetuar o armazenamento destes pares de colisão.

Nessa arquitetura isto não é necessário. Como uma partícula de um corpo rígido está localizada em uma célula, o teste de colisão é executado somente nesta célula e nas células adjacentes, não sendo necessário, portanto, o armazenamento de pares de colisão. O teste de colisão entre uma partícula pode ser visto na figura 5.7.

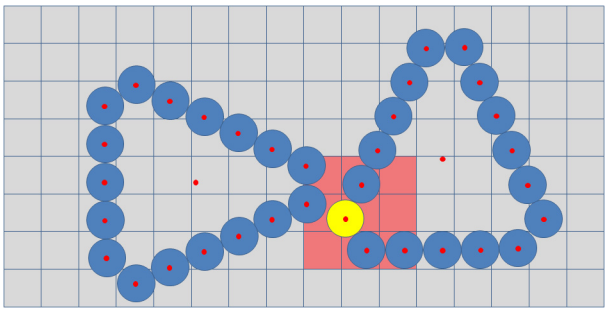

Figura 5.7: Verificação de colisão da partículas em amarelo com as células adjacentes.

5.3.6 Integração da equação de movimento

Finalmente, após a identificação de todas as forças que agem sobre a partícula i, $i < N$, onde N é o número de partículas, é realizado a integração dessas forças para o cálculo da aceleração e, posteriormente, velocidade e posição. Para essa integração, foi utilizado o método de Euler explícito com um passo de tempo constante. Nesta etapa também é efetuada a verificação de colisão entre um conjunto de superfícies implícitas, utilizadas neste trabalho para representar os bordos do contêiner do fluido. Para o seu tratamento optou-se pelo reposicionamento da partícula no interior do contânier e a alteração da componente normal da velocidade [30].

Na simulação de fluidos, o cálculo da velocidade final das partículas utiliza o XSPH, discutido na seção 2.4.3.

Para a simulação de corpos rígidos, a integração é feita diretamente na posição do centro de massa. Porém, para os casos onde ocorre colisão, é necessário que as forças aplicadas em suas partículas sejam transformadas em uma força resultante a ser aplicada diretamente no centro de massa. Dessa forma, deve-se efetuar uma operação de somatório dessas forças em paralelo, conhecida como *scan* [15].

Conforme apresentado na seção 5.3.5, existe a necessidade do armazenamento das forças resultantes durante a interação de cada partícula de corpos rígidos selecionados na etapa de *broad phase*. Essas forças são armazenadas em um array contíguo para todas as partículas de todos os corpos rígidos que potencialmente colidiram. Dessa forma, a aplicação do *scan* em GPU no array de forças resultaria em um somatório das forças de todas as partículas de todos os corpos rígidos dinâmicos que colidiram, comportamento este não desejado. A fim de contornar este problema, se deve aplicar um *scan segmentado* [42], onde utiliza-se um array auxiliar, indicando com o valor *1* o início de cada segmento. Assim, aplicando um scan segmentado do tipo *backwarded inclusive scan* no array de forças utilizando um array auxilar para indicar início de cada corpo rígido, o somatório desta operação para cada corpo rígido será armazenado na mesma posição onde encontra-se o valor *1* no array de *flags*.

Para essa operação, foi utilizado a biblioteca *CUDPP* [3], que implementa o *scan* segmentado e o não segmentado. Um dos problemas encontrados nesta biblioteca é a implementação destas operações somente para os tipos de dados primitivos, como ponto flutuante e inteiro, não sendo possível utilizar vetores, como é o caso do tipo de dado utilizado no array de armazenamento das forças aplicadas nas partículas. Dessa forma, uma alternativa seria a segmentação desse array por componentes *x, y e z* e execução desta operação três vezes, uma para cada componente. Infelizmente essa opção não se apresentou favorável visto a sua complexidade de desenvolvimento em se trabalhar com os componentes em separado. Com o objetivo de contornar este problema, efetuou-se uma modificação no código da biblioteca através da modificação dos núcleos de *kernels* da biblioteca, a fim de permitir a operação com vetores.

[3]Disponível em http://gpgpu.org/developer/cudpp

Capítulo 6

Resultados

Neste capítulo são apresentados os resultados obtidos com o trabalho desenvolvido. Primeiramente serão apresentados os resultado obtidos com a estrutura de aceleração desenvolvida (Seção 6.1). Posteriormente são apresentados os resultados da simulação de fluidos e corpos rígidos utilizando a arquitetura híbrida de GPU e múltiplos núcleos de CPU (Seção 6.2).

Para a realização dos testes foi utilizado um PC com processador Intel Core 2 Quad Q6600 com 4GB de memória ram e placa de vídeo NVidia 9600 GT com 512 MB de memória de vídeo.

Para a renderização da cena é utilizado o *framework* guff (games uff) [50], desenvolvido como um projeto de pesquisa anterior utilizando a linguagem C++ com a incorporação de algumas outras bibliotecas para manipulação de recursos.

6.1 Estrutura de aceleração

Devido à utilização de partículas de fluidos e corpos rígidos com raios e objetivos diferentes, este trabalho utiliza três tabelas de hash, variando o tamanho de sua célula durante a simulação.

Dessa forma, para permitir a dupla interação entre fluidos e corpos rígidos estáticos e dinâmicos, é necessários o compartilhamento de informações entre essas tabelas localizadas em posições de memória independentes.

Para a realização dos testes, foi utilizada uma cena contendo 50 corpos rígidos, totalizando 3800 partículas, variando o número de partículas do fluido. Para comparação, foi

utilizado grade com tamanho de célula capaz de armazenar o maior objeto da cena, assim como o método desenvolvido, utilizando várias grades.

A tabela 6.1 apresenta o resultado do tempo de acesso de uma partícula a todas as suas células adjacentes utilizando apenas uma grade e o método desenvolvido, utilizando duas grades, sendo uma para fluido e outra para corpos rígidos dinâmicos. Além disso, a tabela também apresenta o *speedup*, definido como o tempo de acesso da utilização de uma grade sobre o tempo de acesso utilizado pela estrutura desenvolvida.

Tabela 6.1: Tempo de acesso, em segundos, utilizando uma e duas grades regulares.

Partículas	4098	8196	16392	32784
Duas grades	0,053	0,101	0,199	0,775
Uma grade	0,062	0,126	0,288	1,062
Speedup	1,169	1,247	1,447	1,370

Na tabela 6.2 é apresentado o resultado do tempo de acesso de uma partícula de fluido utilizando apenas uma grade e o tempo de acesso desta mesma partícula utilizando três grades regulares, sendo uma para fluido, corpos rígidos estáticos e dinâmicos.

Tabela 6.2: Tempo de acesso, em segundos, utilizando uma e três grades regulares.

Partículas	4098	8196	16392	32784
Três grades	0,101	0,187	0,372	1,381
Uma grade	0,154	0,298	0,661	2,013
Speedup	1,524	1,593	1,776	1,457

Conforme se pode observar no gráfico apresentado na figura 6.1, a utilização do método desenvolvido oferece um ganho de performance superior aquele utilizando apenas uma grade.

6.2 Simulação de fluidos e corpos rígidos

Nesta seção são apresentados os resultados obtidos na simulação de fluidos e corpos rígidos utilizando várias configuraçõees de execução para análise.

Primeiramente é apresentado na tabela 6.3 a simulação de fluidos individualmente na GPU e CPU utilizando cenas com diferentes números de partículas. O campo da tabela *IPS* representa o *número de interações por segundo* executados pela simulação, sem a contabilização do tempo necessário para renderização. Além disso, também é exibido na tabela o *Speedup* da simulação, definido como a relação de X^1 sobre Y^2. Devido ao foco do trabalho na simulação de fluidos e corpos rígidos, otimizações não foram feitas na etapa

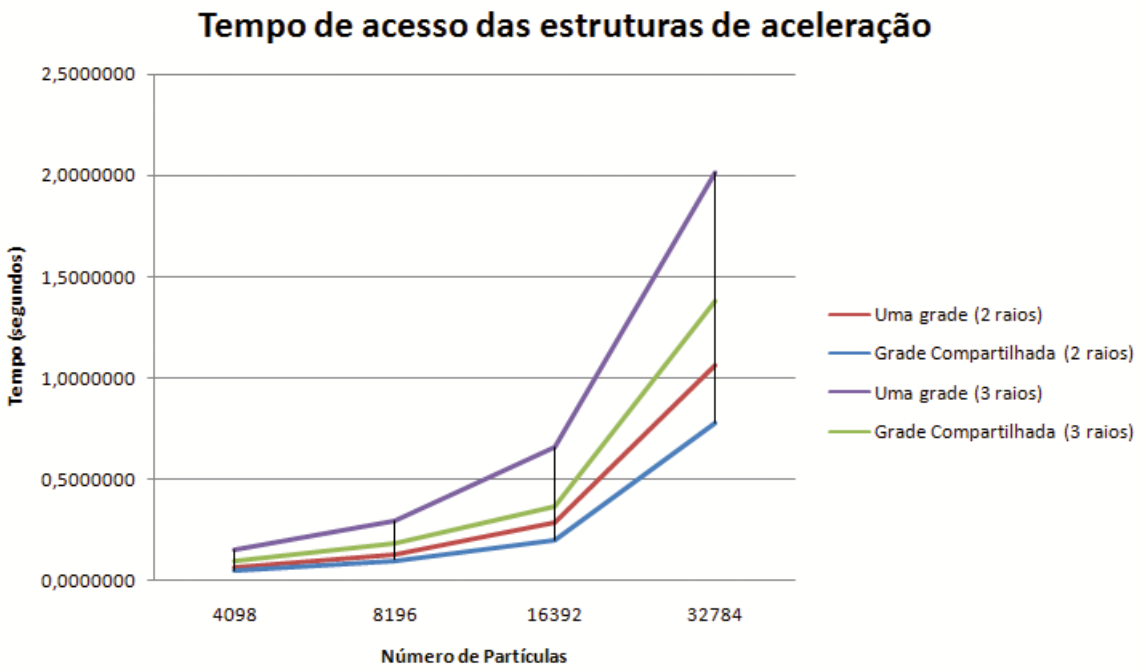

Figura 6.1: Gráfico comparativo da utilização da estrutura de aceleração desenvolvida.

de renderização. Porém, para efeito de comparação, a tabela também exibe o *fps* (frames por segundo) da simulação. A figura 6.2 apresenta o gráfico dos resultados desta tabela.

Tabela 6.3: Resultado da simulação de fluidos em GPU e CPU.

	GPU		CPU		
Partículas	FPS	IPS[2]	FPS	IPS[1]	Speedup
2048	242,3	410,5	28,8	28,9	14,17
4096	96,8	167,0	9,7	9,8	16,99
8192	34,0	43,5	3,0	3,0	14,27
16394	13,9	15,1	1,4	1,4	10,62
32768	4,5	4,7	0,5	0,5	9,00

Na tabela 6.4 é apresentado a simulação de corpos rígidos utilizando diferentes cenários em GPU e CPU, assim como seu gráfico comparativo na figura 6.3. Para este resultado, foram utilizados um conjunto de objetos *teapot* discretizados utilizando 76 partículas com raio de *0,5*, observado na figura 6.4, onde esta cena é simulada.

Os resultados da simulação de fluidos e corpos rígidos, simultaneamente, são executados em duas etapas. Primeiramente a tabela 6.5 apresenta os resultados da simulação de fluidos e corpos rígidos simultaneamente em GPU e CPU, porém, sem considerar interação entre estes. A figura 6.5 apresenta o gráfico desta simulação.

Finalmente são apresentados nas tabelas 6.6, 6.7 e 6.8 o resultado da simulação de fluidos e corpos rígidos com dupla interação em GPU, CPU e modo heterogêneo (CPU e

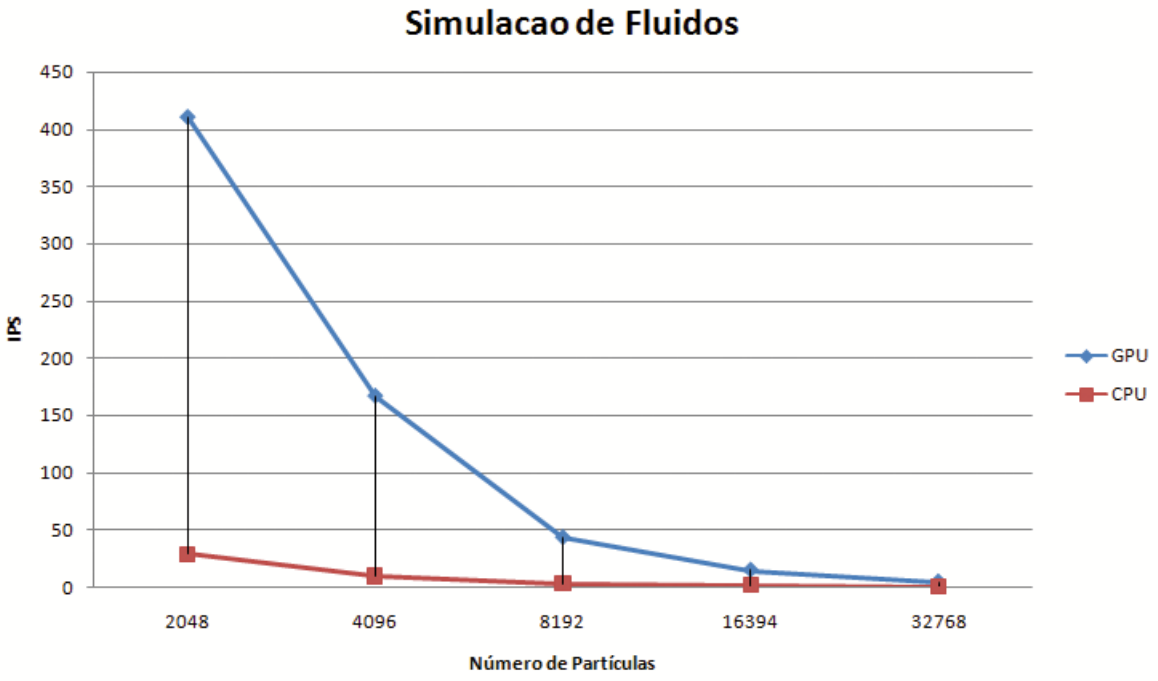

Figura 6.2: Gráfico comparativo da simulação de fluidos em GPU e CPU.

Tabela 6.4: Resultado da simulação de corpos rígidos em GPU e CPU. CR: corpos rígidos.

		GPU		CPU		
CR	Partículas	FPS	IPS[1]	FPS	IPS[2]	Speedup
26	1976	220,7	227,5	19,0	19,0	11,92
53	4028	210,8	223,6	8,6	8,6	25,89
107	8132	159,9	174,7	4,1	4,1	42,48
215	16340	102,2	114,6	2,0	2,0	57,41
430	32680	63,0	72,6	0,9	0,9	74,09

GPU), respectivamente. A figura 6.6 apresenta o gráfico deste resultado. Na figura 6.7, pode-se observar o fluido exercendo forças nos objetos, fazendo os mesmos flutuarem sobre o fluido, através de um empuxo, o qual é baseado no peso do corpo rígido. Na figura 6.8 pode-se verificar a força exercida pelo corpo rígido no fluido, causando seu deslocamento.

A tabela 6.9 apresenta o *speedup* alcançando, comparando-se o ambiente heterogêneo de múltiplos núcleos de CPU e GPU daquele utilizando apenas a GPU.

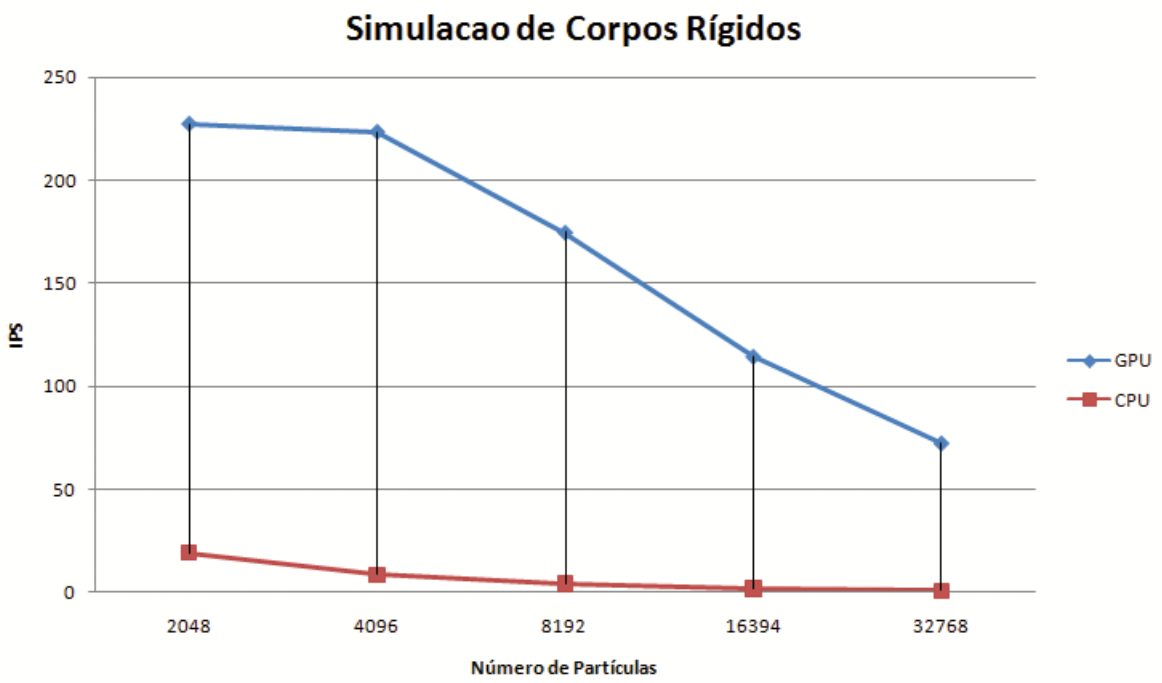

Figura 6.3: Gráfico comparativo da simulação de corpos rígidos em GPU e CPU.

Tabela 6.5: Resultado da simulação de fluidos e corpos rígidos em GPU e CPU sem interação. CR: corpos rígidos; CRP: numero de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema.

				GPU		CPU		
CF	CR	CRP	TPS	FPS	IPS[1]	FPS	IPS[2]	Speedup
4096	200	15200	19296	49,5	57,4	1,6	1,6	34,59
4096	400	30400	34496	43,3	50,2	0,8	0,9	54,97
8192	200	15200	23392	25,8	27,7	1,2	1,2	22,30
8192	400	30400	38592	23,1	25,1	0,7	0,7	33,10

Tabela 6.6: Resultado da simulação de fluidos e corpos rígidos em GPU com dupla interação. CR: corpos rígidos; CRP: numero de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema.

				GPU			
CF	CR	CRP	TPS	FPS	Tempo	IPS	Tempo
4096	200	15200	19296	27,7	0,0361	29,5	0,0339
4096	400	30400	34496	20,6	0,0485	22,0	0,0455
8192	200	15200	23392	18,7	0,0534	19,7	0,0508
8192	400	30400	38592	14,0	0,0711	14,9	0,0672

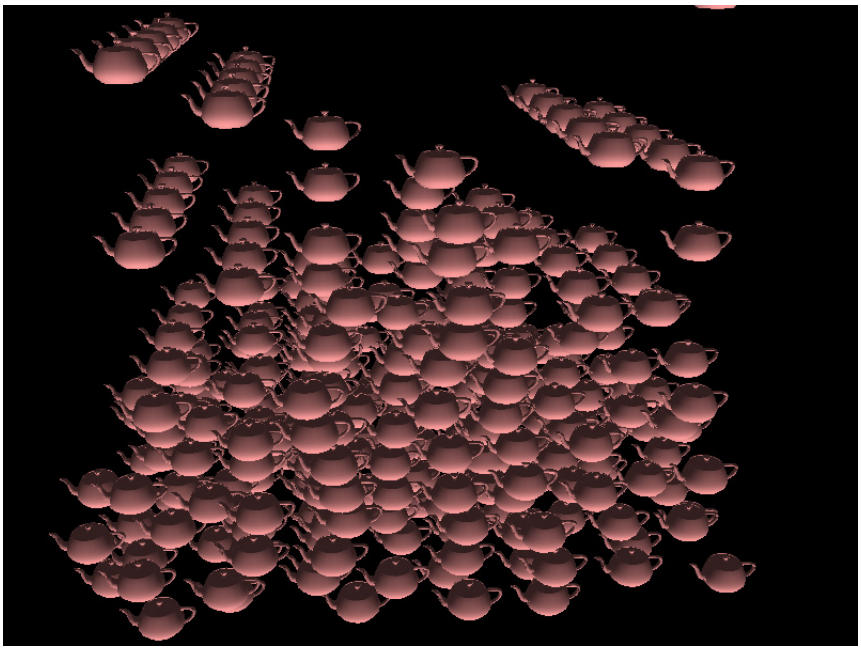

Figura 6.4: Renderização de uma cena da simulação de 430 corpos rígidos totalizando 32680 partículas.

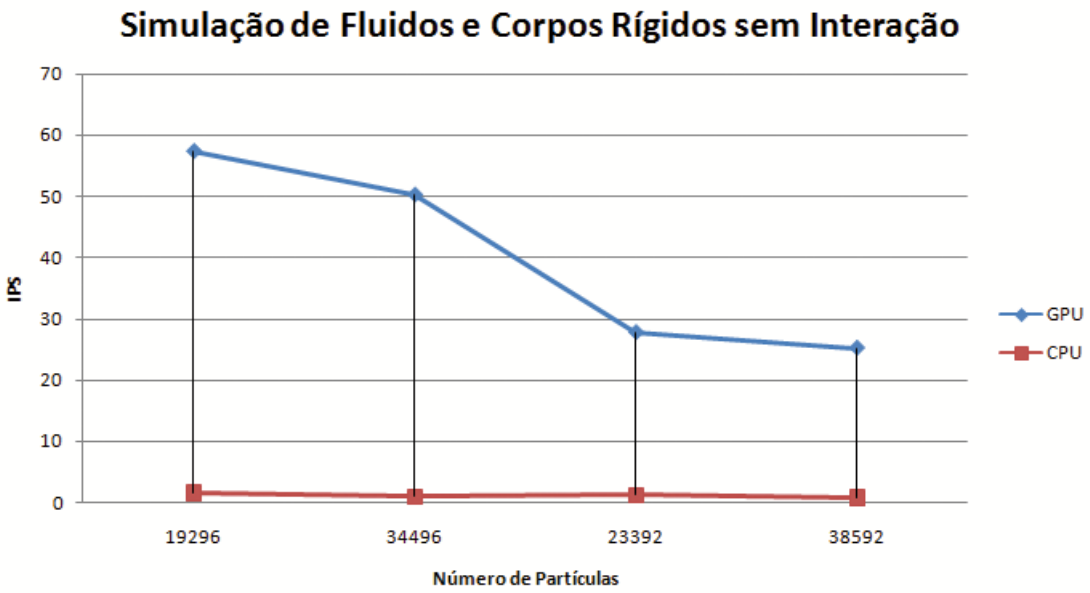

Figura 6.5: Gráfico comparativo da simulação de fluidos e corpos rígidos em GPU e CPU sem interação.

Tabela 6.7: Resultado da simulação de fluidos e corpos rígidos em CPU com dupla interação. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema.

				CPU			
CF	CR	CRP	TPS	FPS	Tempo	IPS	Tempo
4096	200	15200	19296	0,6	1,6077	0,6	1,5924
4096	400	30400	34496	0,3	2,9674	0,3	2,9240
8192	200	15200	23392	0,5	2,0202	0,5	2,0080
8192	400	30400	38592	0,3	3,4130	0,3	3,3784

Tabela 6.8: Resultado da simulação de fluidos e corpos rígidos em modo heterogêneo de CPU e GPU com dupla interação. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema.

				Heterogêneo (CPU + GPU)			
CF	CR	CRP	TPS	FPS	Tempo	IPS	Tempo
4096	200	15200	19296	27,9	0,0359	53,4	0,0187
4096	400	30400	34496	18,8	0,0593	52,6	0,0190
8192	200	15200	23392	18,3	0,0545	28,2	0,0355
8192	400	30400	38592	12,2	0,0817	27,6	0,0363

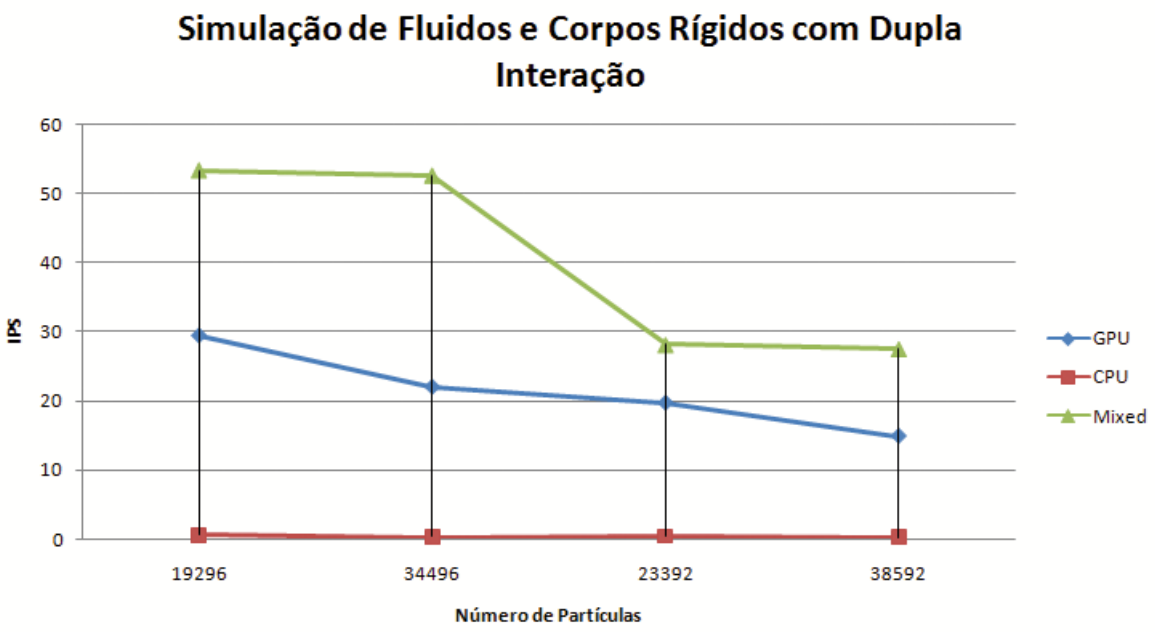

Figura 6.6: Gráfico comparativo da simulação de fluidos e corpos rígidos em GPU, CPU e modo heterogêneo(GPU e CPU) com dupla interação.

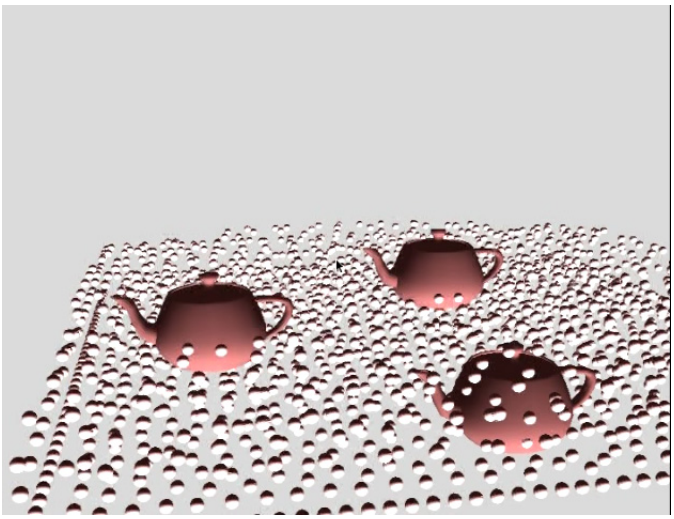

Figura 6.7: Renderização de uma cena onde verifica-se o comportamento do corpo rígido através da aplicação de forças pelo fluido.

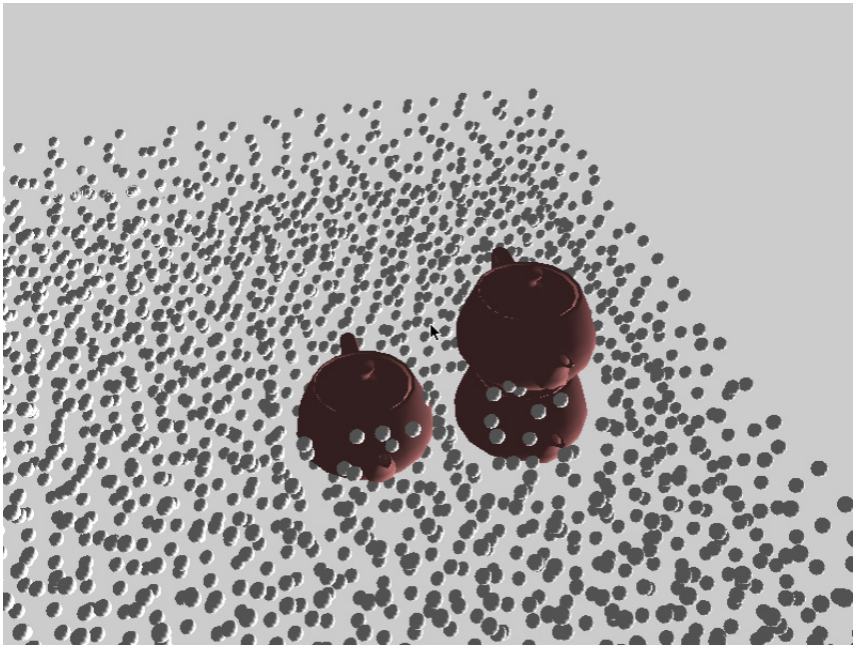

Figura 6.8: Renderização de uma cena onde verifica-se o comportamento do fluido através da aplicação de forças pelo corpo rígido.

Tabela 6.9: Comparação da arquitetura heterogênea de CPU e GPU e da arquitetura de GPU. CR: corpos rígidos; CRP: número de partículas de corpos rígidos; CF: número de partículas de fluido; TPS: total de partículas no sistema.

				gpu		Heterogêneo (cpu + gpu)		
CF	CR	CRP	TPS	IPS[2]	Tempo	IPS[1]	Tempo	Speedup
4096	200	15200	19296	29,5	0,0339	53,4	0,0187	1,81
4096	400	30400	34496	22,0	0,0455	52,6	0,0190	2,39
8192	200	15200	23392	19,7	0,0508	28,2	0,0355	1,43
8192	400	30400	38592	14,9	0,0672	27,6	0,0363	1,85

Capítulo 7

Conclusão e trabalhos futuros

A proposta do desenvolvimento de uma arquitetura híbrida de múltiplos núcleos de CPU e GPU com processamento em paralelo foi obtida, conforme os resultados apresentados.

Para ser possível este processamento, uma estrutura de aceleração foi desenvolvida e apresentou desempenho satisfatório, observada pelos resultados do tempo de acesso apresentados. Esta estrutura foi necessária para permitir o compartilhamento de informações entre tabelas de hash, localizadas em endereços de memória independentes.

Dessa forma, este trabalho mostrou que o tempo necessário para o acesso a espaços de memória diferentes é melhor realizado utilizando a estrutura de aceleração desenvolvida em relação a uma única estrutura de dados capaz de armazenar todos os objetos em uma única célula, utilizando, neste caso, um tamanho de célula capaz de armazenar o maior dos objetos da cena.

A simulação de corpos rígidos apresentou um comportamento diferente daquele observado na simulação de fluidos, conforme se dobrava o número de partículas. Neste caso, o aumento do número de partículas de fluido causava um impacto maior no tempo necessário para o processamento da simulação, em comparação com o aumento do número de partículas de corpos rígidos.

A nova arquitetura heterogênea de CPU e GPU, apesar de não modificar este comportamento, apresentou um *speedup* próximo de três, comparando-se com a mesma simulação executada somente em GPU.

Com o resultado apresentado, a investigação da utilização de múltiplos núcleos de CPU e GPU, simultaneamente, mostrou-se válida, instigando o uso de CPUs com um maior número de núcleos para processamento da simulação.

Como trabalhos futuros, alguns pontos na simulação e arquitetura desenvolvida podem ser aperfeiçoados e estendidos. O primeiro deles seria o tratamento da colisão entre corpos rígidos. No tipo de tratamento desenvolvido existe a necessidade do ajuste fino de parâmetros de elasticidade e amortecimento na geração de forças, além de sua modificação para diversos tipos de simulações, a fim de evitar possíveis interpenetrações entre essas partículas, tornando-o um método muito sensível a parâmetros, com grande dificuldade de configuração.

Além disso, com a nova arquitetura FERMI de GPUs que permite a execução de múltiplos *kernels*, o trabalho desenvolvido pode ser estendido para tirar proveito dessa nova arquitetura, com o objetivo de permitir um maior número de corpos rígidos e partículas de fluidos na simulação.

Devido à utilização de partículas para detecção e tratamento de colisão de corpos rígidos assim como a utilização de múltiplas estruturas de tabelas de hash, um sistema de LOD de colisão pode ser desenvolvido, onde corpos rígidos fora do foco da simulação utilizam discretizações com menor número de partículas, enquanto aquelas dentro do foco de visão são processadas mais detalhadamente através de um número maior de partículas.

Com a utilização de um sistema de partículas para representação de corpos rígidos, além da utilização do método SPH para simulação de fluidos, a arquitetura pode ser estendida para simulação de fusão e solidificação sem muito esforço. Nesse caso, partículas de corpos rígidos, durante sua transição, podem se aglomerar com as partículas de fluido e serem simuladas como tais.

Referências

[1] Nathan Bell, Yizhou Yu, and Peter J. Mucha. Particle-based simulation of granular materials. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 77–86, New York, NY, USA, 2005. ACM.

[2] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 377–384, New York, NY, USA, 2004. ACM.

[3] A.J. Cuadros-Vargas, L.G. Nonato, R. Minghim, and T. Etiene. Imesh: An image based quality mesh generation technique. In *IEEE Proceedings SIBGRAPI*, pages 341–348, 2005.

[4] P. A. Cundall. Distinct element models of rock and soil structure. In *Brown (Ed.) Analytical and Computational Methods in Engineering Rock Mechanics*, pages 129–163, 1987.

[5] Mathieu Desbrun and Marie paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *In Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation*, pages 61–76. Springer-Verlag, 1996.

[6] Christer Erleben. *Real-time collision detection.* Elsevier, San Francisco, CA, USA, 2005.

[7] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann. *Physics based animation.* Charles River Media, San Diego, CA, USA, 2005.

[8] Cass Everitt. Interactive order-independent transparency. Technical report, NVidia Corporation, 2001.

[9] Bryan E. Feldman, James F. O'Brien, Bryan M. Klingner, and Tolga G. Goktekin. Fluids in deforming meshes. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 255–259, New York, NY, USA, 2005. ACM.

[10] T.P. Fires and H. G. Matthies. Classification and overview of meshfree methods. Technical report, Technical University Braunschweig, Brunswick, Alemanha, 2004.

[11] Armando de Oliveira Fortuna. *Técnicas computacionais para dinâmica dos fluidos.* Editora da Universidade de São Paulo, São Paulo, 2000.

[12] Amos Gilat and Vish Subramaniam. *Métodos numéricos para engenheiros e cientistas.* Bookman, Porto Alegre, 2008.

[13] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. In *Royal Astronomical Society, Monthly Notices, vol. 181*, pages 375–389, 1977.

[14] T. Harada. Real-time rigid body simulation on gpus. In Hubert Nguyen, editor, *GPU Gems 3*, pages 611–631. Addison-Wesley, 2007.

[15] Mark Harris, Shubhabrata Sengupta, and John D. Owens. Parallel prefix sum (scan) with cuda. In Hubert Nguyen, editor, *GPU Gems 3*. Addison Wesley, August 2007.

[16] Mark Joselli, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, Regina Leal-Toledo, Aura Conci, Paulo Pagliosa, Luis Valente, and Bruno Feijó. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Comput. Entertain.*, 7(4):1–15, 2009.

[17] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM.

[18] Micky Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics. Master's thesis, 2006.

[19] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *GI '06: Proceedings of Graphics Interface 2006*, pages 41–48, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[20] S. Koshizuka, A. Nobe, and Y. Oka. Numerical analysis of breaking waves using the moving particle semi-implicit method. *International Journal for Numerical Methods in Fluids.*, 26:751–769, 1998.

[21] Sho Kurose and Shigeo Takahashi. Constraint-based simulation of interactions between fluids and unconstrained rigid bodies. In *Proceedings of Spring Conference on Computer Graphics*, pages 197–204, 2009.

[22] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 579–588, New York, NY, USA, 2006. ACM.

[23] Baoquan Liu, Li-Yi Wei, and Ying-Qing Xu. Multi-layer depth peeling via fragment sort. Technical report, Microsoft Research, 2006.

[24] G R Liu and M B Liu. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method; electronic version.* World Scientific, Singapore, 2003.

[25] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977.

[26] L. G. Margolin. Introduction to "an arbitrary lagrangian-eulerian computing method for all flow speeds". *J. Comput. Phys.*, 135(2):198–202, 1997.

[27] B. K. Mishra. A review of computer simulation of tumbling mills by dem part i - contact mechanics. In *International Journal of Mineral Processing, Vol. 71(1-4)*, pages 73–93, 2003.

[28] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Reviews Astronomy and Astrophysics*, 30:543–573, 1992.

[29] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animationr3. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1988. ACM.

[30] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[31] Matthias Müller, Simon Schirm, Matthias Teschner, Bruno Heidelberger, and Markus Gross. Interaction of fluids with deformable solids. *Comput. Animat. Virtual Worlds*, 15(3-4):159–171, 2004.

[32] Matthias Müller, Barbara Solenthaler, Richard Keiser, and Markus Gross. Particle-based fluid-fluid interaction. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 237–244, New York, NY, USA, 2005. ACM.

[33] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element methods: diffuse approximation and diffuse elements. *Computational Mechanics*, (10):307–318, 1992.

[34] NVIDIA. Cuda zone. [Online; Acessado em 13-ago-2010].

[35] NVIDIA. Getting started. nvidia cuda c. installation and verification on microssoft windows xp, windows vista and windows 7, 2009. [Online; Acessado em 13-ago-2010].

[36] NVIDIA. Nvidia cuda. best practices guide. version 3.0, 2010. [Online; Acessado em 13-ago-2010].

[37] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *CA '95: Proceedings of the Computer Animation*, page 198, Washington, DC, USA, 1995. IEEE Computer Society.

[38] A. PAIVA, F. PETRONETTO, T. LEWINER, and G. TAVARES. Simulação de fluidos sem malha: Uma introdução ao método sph. In *27º Colóquio Brasileiro de Matemática*, page 198, Rio de Janeiro, RJ, Brasil, 2009. IMPA.

[39] Claude Puech and Hussein Yahia. Quadtrees, octrees, hyperoctrees: a unified analytical approach to tree data structures used in graphics, geometric modeling and image processing. In *SCG '85: Proceedings of the first annual symposium on Computational geometry*, pages 272–280, New York, NY, USA, 1985. ACM.

[40] Tiago Etiene Queiroz. Animação computacional de escoamento de fluidos utilizando o método sph. Master's thesis, 2008.

[41] Hanan Samet. *The design and analysis of spatial data structures.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[42] Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens. Scan primitives for gpu computing. In *Graphics Hardware 2007*, pages 97–106. ACM, August 2007.

[43] Jonathan Richard Shewchuk. *Delaunay refinement mesh generation*. PhD thesis, Carnegie Mellon University, 1997.

[44] Karl Sims. Particle animation and rendering using data parallel computation. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 405–413, New York, NY, USA, 1990. ACM.

[45] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[46] Jos Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*, 2003.

[47] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 203–210, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[48] Matthias Teschner, Bruno Heidelberger, Matthias Mueller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. In *In Proceedings of VMV'03*, pages 47–54, 2003.

[49] Matthias Trapp and Jürgen Döllner. Real-time volumetric tests using layered depth images. In *Eurographics 2008*, pages 235–238. The Eurographics Association, 2008.

[50] L. Valente. Guff: um framework para desenvolvimento de jogos. Master's thesis, Universidade Federal Fluminense, 2005.

[51] Gary D. Yngve, James F. O'Brien, and Jessica K. Hodgins. Animating explosions. In *Proceedings of ACM SIGGRAPH 2000*, pages 29–36. ACM, 2000.

[52] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. Gpu accelerated sph particle simulation and rendering. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, page 9, New York, NY, USA, 2007. ACM.

[53] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. Adaptive sampling and rendering of fluids on the gpu. In *Eurographics/IEEE VGTC Symposium on Point-Based Graphics*, pages 137–146, New York, NY, USA, 2008.