Natasha Sayuri Dias Nakashima

Paralelização do modelo BioCrowds para simulação de multidões em GPU e inclusão do efeito de pressão

Campinas
2013

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Natasha Sayuri Dias Nakashima

Paralelização do modelo BioCrowds para simulação de multidões em GPU e inclusão do efeito de pressão

Dissertação de mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para a obtenção do título de Mestra em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Léo Pini Magalhães

Este exemplar corresponde à versão final da dissertação defendida pela aluna, e orientada pelo Prof. Dr. Léo Pini Magalhães

Campinas
2013

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

N145p	Nakashima, Natasha Sayuri Dias, 1988- Paralelização do modelo BioCrowds para simulação de multidões em GPU e inclusão do efeito de pressão / Natasha Sayuri Dias Nakashima. --Campinas, SP: [s.n.], 2013. Orientador: Léo Pini Magalhães. Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. 1. Multidão. 2. Comportamento de massa. 3. Processamento paralelo (Computadores). I. Magalhães, Léo Pini, 1952-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Parallelization of BioCrowds algorithm for crowd simulation and inclusion of pushing effect
Palavras-chave em Inglês: Crowd, Collective behavior, Parallel processing (Electronic computers)
Área de concentração: Engenharia de Computação
Titulação: Mestra em Engenharia Elétrica
Banca examinadora: Alessandro de Lima Bicho, José Mario De Martino
Data da defesa: 31-01-2013
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidata: Natasha Sayuri Dias Nakashima

Data da Defesa: 31 de janeiro de 2013

Título da Tese: "Paralelização do Modelo Biocrowds para Simulação de Multidões em GPU e Inclusão do Efeito de Pressão"

Prof. Dr. Léo Pini Magalhães (Presidente): _____

Prof. Dr. Alessandro de Lima Bicho: _____

Prof. Dr. José Mario De Martino: _____

Aos meus pais, Jorge e Miuk,
e meu irmão Tihiro.

Agradecimentos

Ao meu orientador, Prof. Léo Pini Magalhães, pela oportunidade concedida, ensinamentos, paciência, dedicação e orientação que tanto me ajudaram a transpor as barreiras encontradas e atingir meu objetivo.

Aos membros da banca examinadora pelo tempo dedicado e valiosas contribuições para finalização deste trabalho.

Ao grande amigo Victor Ishizuca, que foi meu "veterano" aqui em Campinas, ajudando-me em todos os momentos dessa longa caminhada.

Aos meus pais, Jorge e Miuk, e meu irmão, Tihiro, que mesmo distantes transmitiram muito amor, carinho e incentivos para a conclusão de mais uma etapa em minha vida.

Aos amigos do Laboratório de Computação e Automação (LCA) que foram verdadeiros companheiros de trabalho e de "café" durante todo o desenvolvimento deste trabalho.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio financeiro concedido durante o período do mestrado.

"Keep your dreams alive. Understand to achieve anything requires faith and belief in yourself, vision, hard work, determination, and dedication. Remember all things are possible for those who believe."
Gail Devers

Resumo

O BioCrowds é um modelo para simulação de multidões virtuais, proposto por Alessandro Bicho (2009), baseado no algoritmo de colonização do espaço, o qual foi originalmente concebido para geração de padrões de nervuras em folhas vegetais e de ramificações em árvores. Em sua implementação sequencial, o BioCrowds apresenta uma diminuição na taxa de quadros por segundo em simulações com grande número de indivíduos. O presente trabalho implementa uma proposta para a simulação de multidões virtuais utilizando o BioCrowds através de técnicas de computação paralela para unidade de processamento gráfico (GPU). Aborda-se também a inclusão no modelo do efeito de pressão (*pushing*), o qual está relacionado a cenários onde há grande densidade de indivíduos. Para a paralelização do algoritmo utilizou-se a plataforma OpenCL juntamente com a plataforma CUDA, presente nas placas NVIDIA. Nas simulações realizadas com o algoritmo BioCrowds paralelo, obteve-se um aumento na taxa de quadros por segundo mantendo a qualidade da simulação dos comportamentos emergentes em multidões reais.

Palavras-chave: Simulação de Multidões, Processamento Paralelo, Unidade de Processamento Gráfico (GPU).

Abstract

BioCrowds is a method for crowd simulation proposed by Alessandro Bicho (2009) based on the biologically-motivated space colonization algorithm. This algorithm was originally introduced to model leaf venation patterns and the branching architecture of trees. However, the increase on the number of individuals corresponds to a decrease on the framerate of the simulation. This work implements a solution to simulate virtual crowds using the BioCrowds and parallel computing. Additionally it approaches an extension of BioCrowds to treat simulation scenarios where there is a narrowing in the route. This effect is named pushing. The proposed parallel algorithm was implemented using the OpenCL and NVIDIA CUDA framework. The simulations with this algorithm resulted in a increase on the framerate, maintaining the reproduction of emergent behaviors on real crowds.

Keywords: Crowd Simulation, Parallel Processing, Graphic Processing Unit (GPU).

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ALU	Arithmetic Logic Unit (unidade lógica aritmética)
API	Application Programming Interface (interface de programação de aplicação)
CC	Compute Capability
CPU	Central Processing Unit (unidade central de processamento)
CUDA	Compute Unified Device Architecture
DRAM	Dynamic Random Access Memory (memória dinâmica de acesso randômico)
DSP	Digital Signal Processor
FPS	Frames per Second (quadros por segundo)
GPGPU	General-Purpose Computing on Graphics Processing Unit (computação de propósito genérico em unidade de processamento gráfico)
GPU	Graphic Processing Unit (unidade de processamento gráfico)
OpenCL	Open Computing Language
SFU	Special Function Unit
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SM	Streaming Multiprocessor
SP	Streaming Processor

Sumário

Capítulo 1

Introdução

Atualmente estima-se que a população mundial seja de cerca de seis bilhões e novecentos milhões de habitantes (6.900.000.000) sendo, em sua maioria, encontrados nos grandes centros urbanos e industriais. Nestes locais, é comum haver a movimentação de grande quantidade de indivíduos em horários de pico, no deslocamento para o trabalho, ou em atividades de lazer, como em cinemas, praças, partidas de futebol, etc.

De acordo com Bon (1905), apud Bicho (2009), um grande número de indivíduos em um mesmo ambiente físico compartilhando um objetivo em comum é definido como multidão, sendo que estes indivíduos podem apresentar comportamentos diferentes do que quando estão sozinhos. Estes comportamentos coletivos são chamados de comportamentos emergentes.

Portanto, entender como uma multidão age em diferentes situações se tornou um fator importante para o planejamento e a melhoria dos locais públicos onde estas se concentram, facilitando e agilizando o deslocamento de pessoas ou mesmo no planejamento de segurança onde haja condições de perigo.

Com o objetivo de estudar este comportamento em diferentes situações, vários modelos para simulação de multidões foram propostos nas últimas décadas, tornando-se tema de interesse de diversas áreas do conhecimento como, por exemplo, na engenharia de segurança, onde as simulações podem ser utilizadas para o estudo da desocupação de construções, na indústria de entretenimento, onde podem ser utilizadas na produção de animações, jogos e filmes, e no planejamento e desenvolvimento de áreas urbanas, edifícios e eventos onde haja grande fluxo de pessoas. Em todas estas áreas citadas, a necessidade de simular multidões adviu de situações que podem ocorrer no mundo real e não podem ser obtidos com uma multidão real, por trazer riscos físicos ou mesmo por ser inviável submeter grupos de pessoas a determinadas situações.

Existem, na literatura, diversas propostas para simular multidões. Muitas destas apresentam abordagens referentes a uma aplicação específica como o modelo proposto por Song et al. (2013) para simulação de situações de pânico em ataques bioterroristas. Alguns dos modelos propostos na literatura são pré-programados para simular determinados comportamentos ou possuem métodos complexos que necessitam de cuidado na escolha dos parâmetros para a obtenção de resultados realísticos. Um estudo sobre os diversos modelos existentes pode ser encontrado no livro de Thalmann e Musse (2012).

O modelo proposto por Bicho, denominado BioCrowds (Bicho, 2009), baseia-se no modelo de

colonização do espaço proposto por Runions et al. (2005) e apresenta a ideia de que indivíduos competem pelo espaço no qual se movem. Este espaço é representado através da existência ou ausência de marcadores e o movimento de cada indivíduo simulado é afetado apenas pela disponibilidade destes marcadores.

Em sua tese, Bicho apresenta uma análise qualitativa e quantitativa de seu modelo em relação a outros modelos de simulação de multidões como os propostos por Treuille (2006), Pelechano et al. (2007), Berg et al. (2008) e Helbing et al. (2000).

Ao analisar uma multidão de forma qualitativa, Bicho comprovou o aparecimento de vários comportamentos de multidões reais, como o surgimento de vias de pedestres, ao simular grupos se deslocando em sentidos opostos. Entretanto, o modelo BioCrowds não simulava o efeito causado pela pressão sofrida pelos indivíduos em ambientes muito populosos. Pela análise quantitativa, pode-se perceber que o modelo BioCrowds apresenta uma taxa de quadros por segundos condizente com a quantidade de agentes simulados, mostrando que o seu desempenho é comparável a outros modelos e simuladores existentes. Entretanto, ao aumentar o número de indivíduos simulados e de marcadores no cenário, percebe-se uma diminuição considerável na taxa de quadros por segundo da simulação. Assim, a utilização de computação paralela para a implementação de algoritmos para simulação de multidões apresenta-se como uma possível alternativa para atingir um melhor desempenho.

1.1 Objetivos

O objetivo deste trabalho é o estudo e a implementação do paradigma de computação paralela aplicado ao modelo para simulação de multidões BioCrowds. Serão utilizadas técnicas de programação genérica através da unidade de processamento gráfico (GPUs) para alcançar um desempenho computacional superior ao algoritmo sequencial deste modelo. Além disto, este trabalho propõe um método para a inclusão do efeito de pressão em indivíduos simulados causado pelo contato entre os mesmos em ambientes com grande densidade populacional.

1.2 Justificativa

A unidade de processamento gráfico (GPU) é um dispositivo computacional especializado para processamento e geração de imagens. Estes dispositivos são compostos por um ou mais conjuntos de núcleos de processamento e utilizam o paralelismo de dados para aumentar sua taxa de processamento, como será tratado no Capítulo 3.

Estes dispositivos eram inicialmente utilizados apenas para processamento e renderização de imagens em computadores pessoais. Entretanto, devido a seu grande potencial para processamento paralelo de dados, sua utilização para programação de propósito geral tem se tornado uma área de pesquisa promissora e continuamente ganha atenção de pesquisadores e indústrias do ramo. Adota-se o termo GPGPU (*General Purpose computation on GPU*) para designar este novo ramo de pesquisa.

Os primeiros aplicativos que utilizaram a GPU para este fim precisaram adaptar as interfaces de programação de aplicativos – APIs (*Application Programming Interface*) para este novo

conceito, apresentando uma curva de aprendizado muito lenta para os desenvolvedores. Por este motivo, foram desenvolvidas novas plataformas para facilitar a utilização da GPU para processamento genérico, como o CUDA (NVIDIA, 2012) e OpenCL (Tsuchiyama et al., 2009).

Com o surgimento da arquitetura de GPUs, os trabalhos relacionados a simulação de multidões também tem se beneficiado deste novo paradigma de programação para obter resultados mais realistas em tempo real. Exemplo desta utilização é o trabalho desenvolvido já em 2008 pelos pesquisadores da empresa AMD (Shopf et al., 2008), com o objetivo de aumentar o número de entidades que interagem entre si ao serem simuladas. Neste caso, pode-se pensar em diversas aplicações para a sua utilização, como em simulações de situações de pânico, criando a necessidade de evacuação de um grande número de pessoas, tráfego de pedestres em ruas, entre outras.

Nesta dissertação pretende-se utilizar estas plataformas para programação paralela em GPU para simulação de comportamentos de multidões reais utilizando o modelo BioCrowds proposto por Bicho (2009) em sua tese de doutorado - ver também (Bicho et al., 2012).

1.3 Organização do trabalho

Este trabalho está organizado da seguinte maneira:

- Capítulo 2: apresenta uma fundamentação teórica sobre multidões, a qual servirá de suporte para a compreensão dos demais capítulos. São apresentados conceitos relacionados à multidão, as principais classificações dos sistemas de simulação de multidões e alguns trabalhos consolidados na área.

- Capítulo 3: apresenta uma fundamentação teórica sobre as plataformas CUDA e OpenCL para programação genérica para GPUs.

- Capítulo 4: apresenta o algoritmo para simulação BioCrowds paralelo. O capítulo inicia com uma breve descrição do modelo original, proposto por Bicho (2009). Em seguida as principais características e implementação do algoritmo paralelo são apresentadas.

- Capítulo 5: apresenta os resultados experimentais de estudos de caso do algoritmo paralelo, analisando-o quantitativamente e qualitativamente em relação ao algoritmo original.

- Capítulo 6: são abordadas as contrituições deste trabalho e trabalhos futuros no tema.

- Anexo A: são apresentadas as especificações da placa de vídeo utilizada nas simulações realizadas com o algoritmo BioCrowds paralelo.

- Anexo B: é apresentado como é definida a estrutura de blocos e *threads* a serem executadas na GPU.

- Anexo C e CD: acompanha esta dissertação um CD contendo o código fonte do simulador desenvolvido e vídeos das simulações realizadas neste trabalho.

Capítulo 2

Revisão Bibliográfica

O objetivo do modelo BioCrowds é proporcionar a simulação da movimentação de indivíduos, concentrando-se no realismo dos aspectos comportamentais. Assim, existe a necessidade de compreender a forma como as pessoas se comportam de forma coletiva e quais características naturais estes agrupamentos apresentam em várias situações.

Este capítulo apresenta conceitos relacionados ao comportamento coletivo de pessoas, abordando aspectos psicológicos e sociais, bem como comportamentos característicos das multidões. É importante destacar que não serão diferenciados os termos multidão e grupos de pessoas, pois aborda-se neste trabalho apenas agrupamentos de pessoas sem relação entre si. Com este entendimento, estes termos serão utilizados alternativamente.

São apresentados modelos para simulação de multidões que constam na literatura especializada, tendo sido dado ênfase àqueles que ilustram os comportamentos do próprio ser humano e comportamentos que surgem devido a auto organização de uma multidão.

2.1 Multidões

Há muito, o estudo do comportamento coletivo é alvo de estudo. Já em 1905, o psicólogo social Gustave Le Bon publicou a obra "A psicologia das multidões" (*La psychologie des foules*) (Bon, 1905 apud Bicho, 2009), a qual deu importante contribuição ao estudo de multidões.

De acordo com Le Bon, multidão, em seu sentido ordinário, representa um aglomerado de indivíduos, não importando nacionalidade, profissão, sexo ou motivos que os aproximam. Em seu sentido psicológico, entende-se como uma concentracão de indivíduos com características comportamentais distintas daquelas que apresentariam, caso estivessem isolados. Estes padrões comportamentais são decorrentes das interações e das influências que o indivíduo estabelece com o meio e com os outros indivíduos.

Diversos autores observaram e identificaram algumas características comportamentais individuais em multidões reais (Henderson, 1971; Helbing & Molnar, 1997; Still, 2000), as quais estão relacionadas a fatores inerentes ao ser humano, resultantes de seus desejos, crenças e emoções, e também a padrões que emergem da multidão quando esta é observada como um todo (Bicho, 2009). Os principais comportamentos individuais inerentes a uma multidão observados em situações normais são:

- deslocar-se ao destino (*goal seeking*): os indivíduos movimentam-se no ambiente com o objetivo de atingir seus destinos;

- deslocar-se evitando colisão (*collision avoidance*): ao se deslocar, o indivíduo mantém uma distância de outros indivíduos e de obstáculos, chamada de distância interpessoal, evitando assim, possíveis colisões. Esta distância pode diminuir com o aumento da densidade ao redor do indivíduo;

- estratégia do mínimo esforço (*least effort strategy*): indivíduos escolhem caminhos que permitem chegar mais rápido ao seu destino, demandando menos esforço para um deslocamento. Caso haja mais de uma trajetória possível, é conhecido que o indivíduo escolherá aquela que permitir seu deslocamento com mínima variação de sua velocidade e orientação.

Os comportamentos emergentes são observados ao se analisar a multidão como um todo e resultam de sua auto-organização, ou seja, estes comportamentos não são explicitamente planejados ou organizados. Os principais padrões observados são:

- formação de vias de indivíduos (*lanes formation*): quando há uma grande densidade populacional em um determinado ambiente, surgem fluxos de indivíduos se deslocando na mesma direção, em sentidos opostos (Helbing and Molnar, 1997; Still, 2000) ou no mesmo sentido. Esta auto-organização ocorre pois o indivíduo procura minimizar seu esforço, seguindo o indivíduo imediatamente a sua frente, mantendo uma velocidade igual ou menor a velocidade do mesmo. A Figura 2.1 mostra a formação destas vias de indivíduos em uma multidão real.

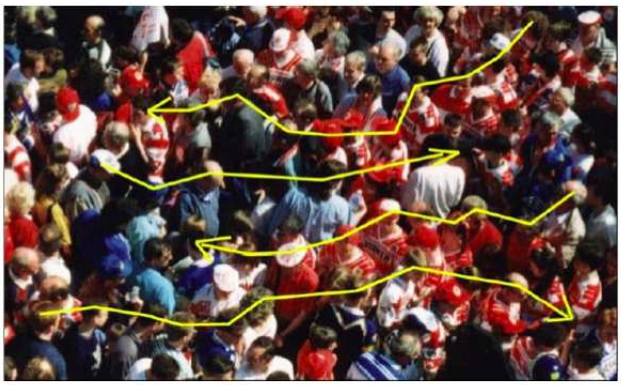


Figura 2.1: A grande densidade local causa o aparecimento de vias de indivíduos para minimizar o deslocamento dos mesmos (Still, 2000).

- prévia organização (*organization prior*): neste efeito, os indivíduos posicionam-se antecipadamente em vias para evitar possíveis colisões quando há grupos se deslocando em sentidos opostos com um determinado espaço entre eles;

- formação de arco (*arc formation*): quando indivíduos desejam passar por um acesso de saída em um ambiente fechado, há a formação de um arco a medida que a densidade

populacional aumenta. Isto ocorre devido à diminuição de velocidade dos indivíduos e a necessidade de se manter próximo à via que os leva ao seu destino;

- efeito de gargalo (*bottleneck effect*): este padrão pode ser observado, por exemplo, em um corredor que apresenta um estreitamento ao longo de seu caminho. Nele, há um aumento da densidade populacional nas regiões anteriores ao estreitamento e uma diminuição da mesma na região após este estreitamento.

- efeito de pressão (*pushing effect*): este efeito ocorre em situações nas quais há o contato físico entre os indivíduos da multidão, como em ambientes onde a densidade de indivíduos é muito grande, sendo comum as pessoas se empurrarem para buscar um caminho para seu deslocamento.

2.2 Modelagem e simulação de multidões

A modelagem e a simulação de multidões tem se tornado tema de interesse de diversas áreas do conhecimento como, por exemplo, na engenharia de segurança, na indústria de entretenimento e no planejamento de áreas urbanas. Devido a esta grande abrangência do tema, Ulicny et al. (2006) propuseram separar estes modelos em duas grandes ênfases: ênfase no realismo dos aspectos comportamentais da multidão, o qual é o foco deste trabalho, e ênfase na visualização de alta qualidade.

No primeiro grupo, o foco está em validar quantitativamente a correspondência do comportamento representado na simulação com os resultados observados em uma multidão real. Neste caso, as simulações apresentam uma visualização simples em 2D e os indivíduos são representados através de formas simples como pontos coloridos, pois esta representação esquemática destaca informações importantes e auxilia na compreensão dos resultados obtidos.

A segunda ênfase, em contrapartida, direciona-se para a utilização pelas indústrias de entretenimento, na produção de filmes e jogos de computador, pois o principal objetivo deste grupo é a apresentação de um resultado visual o mais parecido com a realidade, geralmente não priorizando o realismo no comportamento dos indivíduos. A multidão virtual simulada deve apresentar modelos tridimensionais texturizados e animados e a iluminação do ambiente deve ser adequada para apresentar um ambiente rico visualmente.

A convergência destas ênfases vem sendo facilitada pelo surgimento de novos suportes à simulação e renderização através das tecnologias *multi-core* e GPU. Sistemas orientados ao comportamento estão incorporando uma melhor visualização da simulação enquanto que sistemas onde a ênfase é a vizualização estão inserindo melhores modelos comportamentais, facilitando a criação de animações convincentes.

Diversos modelos e aplicações para simulação de multidões podem ser encontrados na literatura. O presente trabalho adapta uma classificação, já utilizada por Bicho (2009) em sua tese de doutorado, definindo três grandes grupos para classificação de modelos. Na primeira abordagem são congregados os modelos cuja individualidade não é prioritária, ou seja, a interação entre os indivíduos é definida de maneira coletiva. Na segunda, o comportamento coletivo é característica emergente da atuação dos individuos simulados. Na terceira abordagem, chamada

de híbrida, há características tanto referentes aos modelos com foco no coletivo como também aos modelos com foco no indivíduo.

A seguir, serão apresentados trabalhos relacionados a simulação de multidões que utilizam estas abordagens.

2.2.1 Modelos com foco no coletivo

Como foi dito anteriormente, os modelos que compreendem esta categoria abstraem o comportamento individual em favor do fluxo da multidão. Um dos primeiros trabalhos que pode ser classificado neste grupo foi o proposto por Henderson (1971), o qual associou o comportamento de multidões ao movimento de fluidos, demonstrando que é possível modelar o fluxo de indivíduos utilizando as equações de Navier-Stokes [1] para dinâmica de fluidos Newtonianos [2].

Bouvier et al. (1997) utilizaram a combinação de conceitos relacionados a sistemas de partículas, forças newtonianas e grafos para a vizualização de espaços urbanos. Neste modelo, foram definidas duas camadas de abstração. Na primeira, forças de atração e repulsão permitem que os indivíduos, representados por grupos de partículas que interagem entre si, se movam no ambiente. Assim, objetivos geram forças atrativas enquanto que obstáculos geram forças repulsivas. Na segunda, as decisões dos indivíduos simulados são modeladas através de "cargas de decisão" e de "campos de decisão", que podem ser comparados com carga e campo elétrico, respectivamente. Desta forma, assim como uma carga elétrica é influenciada por um campo elétrico, um indivíduo com uma "carga de decisão" sofre influência de um "campo de decisão". Neste modelo, os autores simularam um total de 45.000 indivíduos.

Goldenstein et al. (1999) desenvolveram uma abordagem baseada em sistemas dinâmicos para simulação de indivíduos em uma multidão. Dois sistemas foram criados com este propósito, o primeiro controla o movimento básico dos indivíduos através de sua velocidade angular, enquanto que o segundo sistema modela a geometria do ambiente como alvos e obstáculos. Em um trabalho posterior (Goldenstein et al., 2001) foi adicionado um sistema para localização eficiente de obstáculos, o qual integrava um modelo dinâmico para deslocamento sem colisões, uma estrutura de dados cinéticos e um planejador de caminhos, permitindo a navegação em terrenos complexos e a cooperação entre indivíduos.

Burstedde et al. (2001) utilizaram o conceito de autômato celular bidimensional associado a uma ideia similar a quimiotaxia [3]. No caso da simulação de multidões, ao invés da locomoção ser orientada por um gradiente químico, os pedestres sofrem a influência de um campo de superfície, denominado *flood field*, semelhante a um gradiente, que permite a movimentação do indivíduo para as células vizinhas. Através da introdução deste campo, foi relatado ser possível obter efeitos coletivos e comportamentos auto-organizáveis nas simulações realizadas.

[1] As equações de Navier-Stokes descrevem o movimento de fluidos e foram definidas por Claude-Louis Navier e George Gabriel Stokes.

[2] Um fluido newtoniano é um fluido em que cada componente da tensão de cisalhamento é proporcional ao gradiente da velocidade na direção normal a esta componente.

[3] Quimitaxia é o nome dado ao processo de locomoção de células em direção a um gradiente químico. A quimiotaxia pode ser negativa, fazendo as células irem em sentido oposto a uma substância, ou positiva, fazendo com que estas células caminhem em direção a uma determinada substância.

Hughes (2003) partiu de observações minuciosas do comportamento coletivo de indivíduos para descrever uma multidão como um "fluido pensante" e derivou equações de movimento que regem o fluxo bidimensional de indivíduos para expressar a mecânica do movimento de multidões. A equação básica que governa o fluxo de um único indivíduo andando em uma superfície foi derivada seguindo três hipóteses:

1. a velocidade com a qual o indivíduo caminha é determinada pela densidade dos indivíduos ao redor do mesmo, pela característica comportamental dos mesmos e pelo solo no qual se deslocam;

2. os indivíduos possuem um senso comum da tarefa que devem realizar para chegar ao seu destino comum;

3. indivíduos procuram minimizar o tempo até seu objetivo, entretanto, procuram evitar locais com grande densidade de indivíduos.

Outro trabalho inspirado na mecânica de fluidos foi apresentado por Treuille et al. (2006). Neste modelo, um campo potencial dinâmico integra a navegação global de indivíduos com obstáculos móveis, os quais podem ser outros indivíduos, sem a necessidade de um tratamento explicito para evitar colisão. Para isto, foram criadas quatro hipóteses para a equação do movimento da multidão:

1. cada indivíduo está buscando alcançar seu objetivo;

2. indivíduos se movem com a máxima velocidade possível;

3. há áreas pelas quais é preferível que o indivíduo se locomova;

4. os indivíduos escolhem caminhos de forma a minimizar a combinação linear de três termos: a distância a ser percorrida, o tempo necessário para alcançar o destino e o desconforto sentido ao longo do caminho.

Karamouzas et al. (2009) desenvolveram um método genérico para o planejamento realista de caminhos, denominado IRM (*Indicative Route Method*). Neste método, uma rota inicial, chamada pelo autor de rota indicativa (*Indicative Route*), é definida da posição inicial do indivíduo até seu objetivo. A partir deste caminho inicial, define-se um corredor ao seu redor, formando um conjunto de caminhos livres de colisão que permitem o cálculo de caminhos suaves para movimentação do indivíduo. O caminho final é planejado através da abordagem de campos de força. Quatro forças são calculadas: força para manter o agente dentro dos limites do corredor calculado, força de direção, que guia o indivíduo ao seu objetivo, uma força é adicionado para gerar variações aleatórias na rota e a força de repulsão, responsável por evitar colisões.

Bicho (2009) propôs em sua tese de doutorado o modelo chamado BioCrowds, o qual é um modelo para simulação de multidões desenvolvido baseado no algoritmo de colonização do espaço, apresentado por Runions et al. (2005) para simular o crescimento de nervuras em folhas, galhos e ramos de árvores. Neste trabalho, o espaço disponível para o crescimento da lâmina

da folha é identificado pela presença de um hormônio chamado auxina e o crescimento acontece devido a competição por estes espaços pelas nervuras da folha.

Na abordagem proposta por Bicho, o crescimento das nervuras pode ser interpretado como o deslocamento de um agente em um determinado cenário preenchido por marcadores, similarmente às auxinas. Assim, o algoritmo pode ser dividido em quatro etapas: inicialização do modelo, cálculo do agente mais próximo de cada marcador, cálculo do vetor de movimento instantâneo de cada agente e atualização de sua posição. Por este algoritmo ser objeto de estudo deste trabalho, o mesmo será detalhado no Capítulo 4.

Jiang et al. (2010) propuseram uma abordagem para simulação de multidões baseada no modelo de Treuille et al. (2006). Os autores desenvolveram um método para discretização de ambientes através de uma grade, onde cada elemento do cenário é disposto em células adjascentes aos outros elementos que compõe o ambiente. Além desta abordagem, o modelo incluiu o que os autores chamaram de "campo de desconforto" ao redor de obstáculos do ambiente, gerando assim trajetórias mais suaves para os indivíduos simulados. Este método gerou multidões e seus comportamentos, entretanto, algumas limitações foram observadas nas simulações como a falta de heterogeneidade entre os indivíduos e o elevado custo computacional para simualação de ambientes complexos.

Através desta seção, pode-se constatar a grande variedade de modelos que tratam a multidão como um coletivo, onde o indivíduo não é o foco da simulação, mas sim os comportamentos emergentes de sua interação. Na próxima seção serão abordados modelos que tratam estes comportamentos como consequência de atuação individual.

2.2.2 Modelos com foco no indivíduo

Nesta linha de estudo, pode-se classificar como trabalho precursor o modelo proposto por Reynolds (1987), baseado em regras comportamentais. Através da aplicação das mesmas a cada personagem virtual, definido como *boid*, consegue-se simular o movimento de bandos de pássaros, rebanhos de animais e cardumes de peixes.

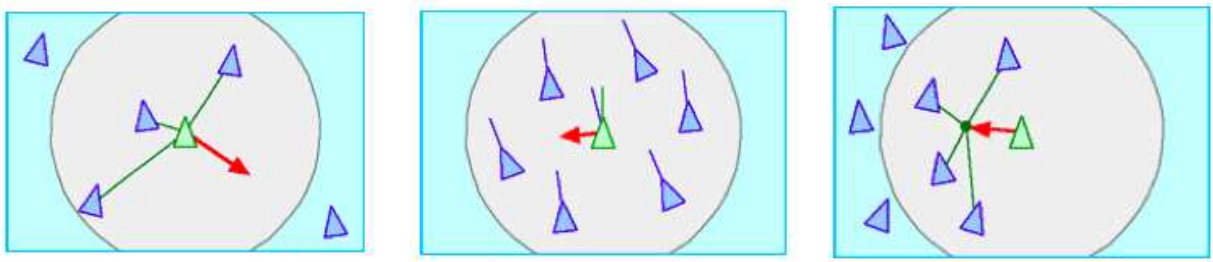
O movimento dos personagens autônomos virtuais (*self-animated characters*) dentro do grupo é definido através da utilização de três regras, as quais estão organizadas de acordo com seu nível de prioridade (Figura 2.2):

1. separação: com maior prioridade, esta regra permite manter uma distância mínima em relação aos *boids* vizinhos e aos obstáculos do ambiente, evitando assim colisões;

2. alinhamento: através de ajustes no módulo e posição do seu vetor velocidade, o *boid* deve manter uma trajetória coerente aos *boids* vizinhos;

3. coesão: o *boid* deve manter sua posição próxima ao centroide das posições dos *boids* vizinhos.

Cada indivíduo tem acesso direto a descrição geométrica de toda a cena, entretanto, reage apenas a indivíduos dentro de uma pequena vizinhança ao seu redor, caracterizada por uma distância, medida do centro do *boid*, e um ângulo, medido a partir da direção do deslocamento



(a) (b) (c)

Figura 2.2: Regras propostas por Reynolds para definição do comportamento coletivo dos *boids*: (a) separação, (b) alinhamento e (c) coesão (Reynolds, 1987).

do mesmo. Além desta interação entre *boids* vizinhos, não há contato direto entre eles, fazendo com que o modelo, por exemplo, não simule o fenômeno de pressão (*pushing*) entre os mesmos.

Tu and Terzopoulos (1994), aprimoraram o modelo proposto por Reynolds e criaram um ambiente para simulação de ambientes marinhos com mínima intervenção do usuário. Utilizando uma máquina de estados finitos baseada em regras, conseguiu-se aumentar o realismo comportamental das simulações adicionando visão sintética e percepção do ambiente a cadurmes de peixes. Assim, foram utilizados três sistemas responsáveis pelo controle e fluxo de informação no peixe artificial:

1. sistema motor: compreende o modelo dinâmico do peixe, os atuadores e um conjunto de controladores de movimento (MC – *motor controllers*). Eles possuem procedimentos parametrizados, cada um dedicado à realização de uma função motora específica, como ”nadar” ou ”virar à esquerda”;

2. sistema de percepção: utiliza um conjunto de sensores virtuais responsáveis pela captação de informações referentes à dinâmica do ambiente simulado. Ele inclui um mecanismo de atenção perceptiva, o qual permite que o peixe artificial treine seu sensor virtual para filtrar informações de acordo com suas necessidades comportamentais;

3. sistema de comportamento: serve como um mediador entre o sistema de percepção e o sistema motor. O animador determina as características inatas do peixe artificial, como se ele gosta ou não de escuridão ou se é femea ou macho, através de um conjunto de parâmetros de hábitos. Um gerador de intenção combina estes hábitos com o fluxo de entrada de informação sensorial para gerar metas dinâmicas para o peixe e computar o parâmetro apropriado para o controlador motor levar o peixe ao seu destino.

Dentro deste grupo de modelos com foco no indivíduo, encontra-se também os modelos de forças sociais, sendo o precursor desta idéia os trabalhos de Helbing and Molnar (1997) e Helbing et al. (2000). Sua base consiste em definir forças psico-sociais e físicas que influenciam na dinâmica de indivíduos, os quais movimentam-se em um espaço bidimensional contínuo. Os autores sugeriram que a descrição do movimento de um indivíduo está sujeita a forças sociais, as quais são medidas através da motivacão interna de cada indivíduo para realizar certas ações

ou movimentos. Assim, foram definidas duas forças essenciais, as quais são responsáveis pela variação da velocidade do indivíduo no tempo:

1. repulsão: existe uma força repulsiva a outros indivíduos, obstáculos e cantos;
2. atração: os indivíduos podem ser atraídos por outros indivíduos ou por outros objetos.

Juntando-se estas duas forças e combinando-as com um termo de flutuação de comportamento, o qual descreve variações comportamentais decorrentes de desvios acidentais ou deliberados às regras habituais de movimento, Helbing e Molnár (1997) produziram uma equação que define a mudança temporal da velocidade v_α de um pedestre α.

$$m_\alpha \frac{d\vec{v_\alpha}}{dt} = \vec{f_\alpha}(t) + F \tag{2.1}$$

onde, f_α representa as diferentes influências no comportamento de um indivíduo α e F são flutuações que podem ser adicionadas ao sistema. Utilizando estas forças, o modelo conseguiu simular interações entre pessoas e obstáculos, comportamento realístico de *pushing* e a variação no fluxo dos indivíduos no ambiente.

Musse (2000) apresentou uma metodologia hierárquica para melhor descrever indivíduos, grupos e multidões, chamada KSI, a qual é composta por três tipos de informação: *Knowlege* (conhecimento) – é o estado interno, referente à percepção e à memória das entidades, *Status* (status) – refere-se aos atributos das entidades e *Intentions* (intenções) – referente aos objetivos a serem alcançados. Musse e Thalmann (2001) propuseram um modelo hierárquico baseado em grupos para simulação de multidões em tempo real. Denominado de ViCrowd, ele permite definir comportamentos de indivíduos e de grupos em três níveis de autonomia: programados, autônomos e guiados, os quais estão relacionados à independência dos mesmos em relação ao usuário do sistema e à informação necessária para simular a multidão.

Zhang et al. (2010) desenvolveram um sistema para simulação de multidões baseado no modelo de redes sociais, o qual é definido como o relacionamento social entre os indivíduos. Neste sistema, três forças sociais são calculadas: a força de direção, a força entre os pedestres e a força entre o pedestre e o limite do espaço determinado pelo seu objetivo e sua percepção do ambiente. As forças calculadas neste modelo podem ser expressas pela seguinte equação:

$$m_i \frac{d\vec{v_i}}{dt} = \vec{f_\alpha} + \sum_\beta \vec{f_{\alpha\beta}} + \sum_b \vec{f_{\alpha b}} + \sum_i \vec{f_{ai}} + \sum_\beta \vec{f_{\alpha\beta}^{att}} + \xi \tag{2.2}$$

onde, $\vec{f_\alpha}$ indica a força de direção; $\vec{f_{\alpha\beta}}$ representa a força repulsiva entre o indivíduo α e o indivíduo β; $\vec{f_{\alpha b}}$ indica a força de repulsão entre o indivíduo α e o obstáculo mais próximo; $\vec{f_{ai}}$ indica a força de atração entre o indivíduo α e o ponto de interesse i; $\vec{f_{\alpha\beta}^{att}}$ representa a força de atração entre o indivíduo α e o indivíduo β e ξ representa uma força aleatória de flutuação.

Ondřej et al. (2010) desenvolveram um modelo baseado em visão sintética para evitar colisões em simulações interativas de multidões. Este modelo foi aplicado em situações complexas de interação entre indivíduos, tendo como resultado padrões adequados de auto-organização da multidão. De acordo com Cutting et al. (1995), apud Ondřej et al. (2010), humanos respondem

a duas perguntas durante a interação com obstáculos estáticos e dinâmicos: uma colisão irá ocorrer? Quando a colisão irá ocorrer? As respostas a estas perguntas podem ser extraídas de dois indicadores do fluxo óptico percebido:

1. humanos percebem visualmente obstáculos dentro de um determinado ângulo, denominado pelos autores de ângulo de rolamento (*bearing angle*). Uma colisão é prevista quando a derivada no tempo do ângulo é zero ou próxima de zero;

2. humanos percebem obstáculos com um determinado tamanho. A taxa de crescimento destes no tempo permite a percepção de obstáculos que estejam se aproximando. Além disto, quanto maior a taxa de crescimento, mais rápido o obstáculo está se aproximando. Assim, pode-se avaliar o tempo para colisão *ttc* (*time-to-collision*).

Para determinar a trajetória livre de colisões dos indivíduos simulados, os autores descreveram os obstáculos estáticos e dinâmicos como um conjunto de pontos $P = \{p_i\}$ resultantes da visão sintética de cada indivíduo. Em seguida, para cada ponto percebido p_i, calcula-se o ângulo chamado pelos autores de *bearing angle*, que é o ângulo de percepção de obstáculos de cada indivíduo, a derivada no tempo deste ângulo e o tempo tti_i (*time to interaction*) restante para a interação relativa do indivíduo. O risco de uma possível colisão é determinado pela derivada do *bearing angle* e o grau de periculosidade da situação pelo tempo tti_i. A reação do indivíduo é determinada através de duas estratégias: para evitar futuras colisões, adapta-se sua orientação com antecedência; em caso de colisão iminente, desacelera-se o indivíduo até ele ficar completamente parado ou a colisão ser resolvida.

Kim et al. (2012)propuseram uma nova técnica para simular padrões comportamentais em indivíduos de uma multidão baseado na teoria da psicologia chamada de Síndrome de Adaptação Geral (SAG) (Selye, 1956, apud Kim et al., 2012). Na idéia proposta, o comportamento de cada indivíduo simulado pode ser alterado no decorrer do tempo em resposta a estímulos externos como mudanças de situações ou no ambiente ao qual ele está inserido. O sistema criado com este modelo apresenta três componentes principais, o primeiro gera um fator chamado de fator de estresse em resposta a um estímulo que o indivíduo esteja recebendo do ambiente, em seguida um segundo componente acumula este fator utilizando o modelo de SAG e, por fim, altera-se o comportamento do indivíduo simulado através do aumento de sua agressividade e impulsividade. Os resultados obtidos por Kim et al. (2012) visando a simulação de multidões em situações de evacuação e movimentação de grupos de indivíduos em sentidos opostos apresentaram o surgimento de comportamentos observados em multidões reais quando submetidas a situações de estresse.

Assim como apresentado na seção anterior, também se constata a grande variedade de modelos representados nesta segunda classificação, denominada por Bicho (2009) “agentes”, a qual tem como objetivo a modelagem individual dos elementos que constituem a multidão. Na próxima seção, são apresentados modelos que podem ser classificados tanto na abordagem coletiva quanto na abordagem de indivíduos, chamados de híbridos.

2.2.3 Modelos híbridos

Existem trabalhos que utilizam características destas duas abordagens, que podem ser chamados de híbridos, permitindo combinar os aspectos psicológicos de cada indivíduo e aspectos globais da multidão.

Pelechano et al. (2007) propuseram um framework para simulação de multidões em tempo real, composto por três módulos: o HALCA (*Hardware Acceleration Character Animation Library*), HiDAC (*High-Density Autonomous Crowds*), o qual é um sistema para simulação de multidão e o APM (*Animation Planning Mediator*). O HiDAC, através da utilização da combinação de regras geométricas e psicológicas com o modelo de forças físicas e sociais, simula a movimentação local de indivíduos em multidões e calcula caminhos globais para os mesmos em um ambiente virtual alterável. De um modo geral, através do HiDAC calcula-se para cada quadro a posição p, a velocidade v e a orientação desejada para cada agente. Em seguida, estas informações são passadas ao APM para a escolha dos parâmetros a serem enviados ao módulo de animação, o HALCA. O HiDAC é baseado em uma combinação de informações geométricas, tais como área de influência e ângulo de orientação entre agentes, e regras psicológicas, tais como pânico e impaciência, associadas a um modelo de forças físicas para gerar comportamentos realistas. Além disto, estas regras psicológicas permitem que o modelo simule o comportamento de *pushing* entre os indivíduos simulados, pois em situações de pânico, os mesmos chegarão perto o suficiente dos demais indivíduos até empurrá-los para conseguir um caminho livre para locomoção em ambientes com grande densidade de indivíduos.

Qiu e Hu (2010) propuseram um sistema híbrido para simulação de diferentes aspectos da estrutura de grupo em uma multidão. Neste método, a multidão é entendida como um conjunto de grupos de diferentes indivíduos, sendo caso especial um grupo composto por um único indivíduo e uma multidão composta por um único grupo. Nos casos em que há vários grupos pertencentes à multidão, os autores definiram a existência de interações entre cada membro do grupo (*intra-group structure*), através das quais cada indivíduo influencia os demais pertencentes ao mesmo grupo, e interações entre os diferentes grupos que compõem a multidão (*inter-group structure*). Estas informações são armazenadas na forma de matriz e definem a estrutura dos grupos da multidão. Por fim, a modelagem do comportamento de cada indivíduo da multidão é inspirada no trabalho de Reynolds (1987) no qual um conjunto de vetores é utilizado para representar as forças aplicadas a cada individuo. Estes são:

- vetor para comportamentos referentes a prevenção de colisões;
- vetor para componentes aleatórias do movimento do agente;
- dois vetores para manter o comportamento de grupo, o primeiro para a movimentação do indivíduo para o centro do respectivo grupo e o segundo vetor para movimentar o individuo conforme a movimentação do grupo como um todo.

Chen et al. (2012) também propuseram um método híbrido para simulação de multidões. Neste abordagem, os autores desenvolveram três modelos para caracterizar diferentes aspectos de uma multidão. O primeiro foca na caracterização individual de cada membro da multidão, o segundo modelo tem como objetivo descrever a dinâmica global da multidão e o último modelo

une os modelos anteriores, permitindo, desta forma, o estudo do comportamento da multidão tanto em escala individual como global. Para a realização das simulações utilizando este modelo híbrido, os autores montaram uma arquitetura computacional em forma de grade. Nela, tem-se vários computadores interconectados a uma rede de alto desempenho para troca de informações e dados entre os mesmos.

Outro método para simulação de multidões que utiliza uma abordagem híbrida é o modelo de Xiong et al. (2010). Os autores combinaram aspectos coletivos e individuais de uma multidão, particionando o ambiente a ser simulado de acordo com estas características. Durante a execução da simulação, cada método trabalha simultaneamente em suas correspondentes partes do ambiente simulado e a comunicação entre estas é feita através de operações que calculam as informações de velocidade e densidade da multidão obtidas através do modelo com foco individual, inserindo estes valores no modelo coletivo, e outras operações responsáveis pela criação de agentes para o modelo com foco no indivíduo através da coleta de informações da multidão simulada utilizando o modelo com foco no coletivo.

A próxima seção enfoca os modelos propostos para simulação de multidões que utilizam conceitos de programação paralela e foram implementados utilizando a unidade de processamento gráfico (GPU - *Graphic Processing Unit*) do computador para alcançar níveis superiores de desempenho daqueles implementados para a CPU. É importante ressaltar que os modelos propostos a seguir também podem ser classificados de acordo com as Seções 2.2.1 e 2.2.2, entretanto serão apresentados em uma nova seção para enfatizar a utilização da GPU como hardware para processamento.

2.3 Simulação de multidões com suporte computacional à unidade de processamento gráfico - GPU

Em sua tese, Bicho (2009) apresentou uma análise qualitativa de modelos para simulação de multidões, mostrando o desempenho computacional destes modelos em simulações com grande número de indivíduos. A Tabela 2.1 mostra uma comparação feita entre eles.

Autores/Modelos	**Hardware**	**Desempenho Computacional**
Treuille et al. (2006)	Pentium 3,4 Ghz	2 a 5 FPS (10000 indivíduos)
Pelechano et al. (2007)	Xeon 2,99 Ghz, 2 GB RAM	25 FPS (1800 indivíduos)
Berg et al. (2008)	Xeon 2,93 Ghz, 8 GB RAM	20 FPS (2500 indivíduos) a 2 FPS (20000 indivíduos)
BioCrowds (Bicho, 2009)	Core 2 Duo 2,2 Ghz, 3 GB RAM	30 FPS (800 indivíduos) a 6 FPS (12800 indivíduos)

Tabela 2.1: Custo computacional dos modelos para simulação de multidões

Percebe-se, como esperado, que com o aumento do número de indivíduos simulados tem-se

uma diminuição considerável na taxa de processamento, devido ao aumento do tempo computacional. Além disto, para simulações em tempo real, nas quais as taxas de processamento e renderização devem ser elevadas, não se consegue simular, em tempo real, multidões com elevado número de indivíduos.

Para solucionar este gargalo computacional, cada vez mais são utilizadas placas de processamento gráfico (GPUs) como ferramentas para processamento paralelo de dados. Os trabalhos relacionados a simulação de multidões também tem se beneficiado deste novo paradigma de programação para obter resultados mais realistas em tempo real.

Exemplo desta utilização é o trabalho desenvolvido pelos pesquisadores da empresa AMD (Shopf et al., 2008), com o objetivo de aumentar o número de entidades que interagem entre si ao serem simuladas. Eles implementaram um ambiente que utiliza a GPU para o planejamento de rotas para multidões em grande escala. Eles utilizaram uma abordagem contínua, como a proposta por Treuille et al. (2006), juntamente com um modelo de grão-fino [4] para a navegação local sem colisões. Este sistema conseguiu simular 65.000 indivíduos utilizando uma única placa gráfica ATI RadeonTM HD 4870 a uma taxa de frames por segundo superior a 20.

Passos et al. (2010) propuseram uma nova arquitetura para armazenar, organizar e simular grandes quantidades de indivíduos autônomos utilizando GPU. Esta estrutura de dados pode ser utilizada por qualquer simulador baseado em autômatos celulares com vizinhança de Moore variável. Utilizando a placa gráfica NVIDIA 8800 GTS, conseguiu-se simular um total de 1048567 *boids* a uma taxa de quadros por segundo igual a 25,45.

O modelo de Ondřej et al. (2010), comentado na Seção 2.2.2, foi implementado utilizando OpenGL, programação de Shaders e CUDA, a arquitetura utilizada pelas placas de processamento gráfico da NVIDIA. Para teste, os autores utilizaram a GPU Quadro FX 3600M e obtiveram uma taxa de 25 quadros por segundo ao simular 200 pedestres se locomovendo. Neste último caso, pode-se pensar em diversas aplicações para a sua utilização, como em simulações de situações de pânico, criando a necessidade de evacuação de um grande número de pessoas, tráfego de pedestres em ruas, entre outros.

2.4 Considerações finais

Os trabalhos apresentados nesta seção foram de fundamental importância tanto para o entendimento de multidões reais, quanto para a compreensão das abordagens utilizadas para modelar e simular o comportamento coletivo. Os trabalhos foram divididos em três grupos de acordo com o enfoque dado, que pode ser nos aspectos individuais, no coletivo dos indivíduos ou em ambos os enfoques. Esta taxonomia é utilizada neste trabalho para facilitar a comparação do modelo aqui utilizado e dos modelos presentes na literatura e foi adaptada da classificação apresentada por Bicho (2009) em sua tese de doutorado.

É importante destacar que esta classificação não é a única utilizada para os modelos de multidões. Pelechano, em seu livro, utiliza uma classificação que distingue modelos baseados em grandezas macroscópicas, como média de velocidade, densidade, fluxo e pressão, e basea-

[4] A granulariade de um sistema ou modelo é definida pela avaliação de seu detalhamento. Quanto mais bem detalhado e subdividido é o modelo do sistema, mais fina é sua granularidade.

dos em grandezas microscópicas, como a exata posição e velocidade de cada indivíduo em um determinado momento.

Finalmente, foram comentados os trabalhos referentes à simulação de multidões que incorporaram conceitos de computação paralela em sua implementação. Estes trabalhos utilizam a placa gráfica do computador, GPU, para a realização dos cálculos necessários para a simulação, mostrando-se como uma grande tendência para o desenvolvimento de aplicações que necessitam de alto poder computacional.

Neste trabalho, o modelo para simulação de multidões escolhido para estudo foi o BioCrowds desenvolvido por Bicho (2009). Sua escolha foi baseada na simplicidade para definição dos parâmetros de entrada para definição da multidão. Além disto, neste modelo, ao trabalhar com a representação de agentes pontuais, não ocorre colisão entre os mesmos. No Capítulo 4 desta dissertação este modelo será discutido detalhadamente.

Capítulo 3

Plataformas para programação genérica utilizando GPUs

Um dos objetivos desta dissertação é analisar o uso de unidades de processamento gráfico GPUs como hardware para processamento do algoritmo BioCrowds. Desta forma, este capítulo apresenta a arquitetura das placas de vídeo NVIDIA, bem como as plataformas para a sua utilização para processamento genérico, CUDA e OpenCL, as quais servirão de suporte para compreensão dos capítulos seguintes deste trabalho.

3.1 Computação paralela utilizando GPU

Desde 2003, a indústria de semicondutores concentrou-se nas arquiteturas para microprocessadores: *multicore* e *many-core* (Kirk and Hwu, 2010).

Os microprocessadores *multicore* utilizam um sofisticado sistema de controle lógico para manter a velocidade de execução enquanto programas são processados pelos diversos núcleos de processamento existentes no chip. Estes microprocessadores possuem grande quantidade de memória *cache*, a qual reduz o tempo de espera para acesso a instruções e aos dados em aplicações. Um exemplo desta arquitetura é o processador Intel Core i7®, no qual cada um dos núcleos pode executar independentemente suas instruções.

O foco dos processadores *many-core* está na quantidade de dados processados durante a execução de um programa (*throughput*). Esta arquitetura tira vantagem da grande quantidade de núcleos disponíveis para cálculo em ponto flutuante para diminuir o tempo de execução de aplicações, tornando-se uma alternativa atraente para sistemas de computação de alto desempenho. Um exemplo é a GPU NVIDIA Geforce GTS 250, que possui 128 unidades de processamento. Esta GPU é *multithread* [1] e seus núcleos compartilham a unidade de controle e o cache de instrução.

Dois exemplos destas arquiteturas são mostrados na Figura 3.1, onde a CPU representa uma arquitetura *multicore*, tipicamente com poucas unidades lógicas e aritméticas, uma unidade de controle sofisticada e grande quantidade de memória cache, e a GPU, uma arquitetura *many-*

[1] *Multithread* refere-se a capacidade do *hardware* de executar eficientemente múltiplas *threads*, que são a menor unidade de processamento que pode ser tratada pelo sistema operacional.

core, que por sua vez possui grande quantidade de unidades de processamento, pouca memória cache e uma simplificada unidade de controle de execução.

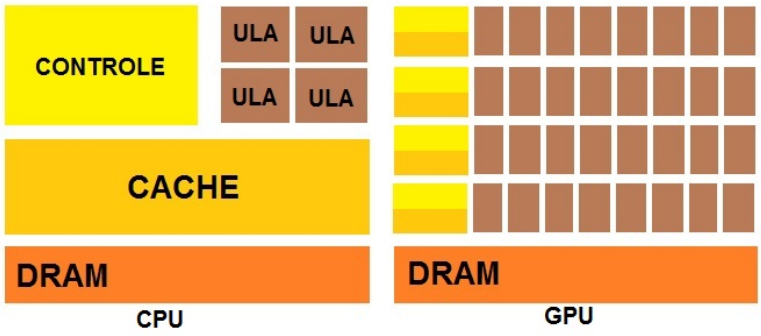


Figura 3.1: Diferença entre as arquiteturas GPU e CPU. Figura adaptada de (Kirk and Hwu, 2010).

Embora tenham uma grande capacidade de processamento, as GPUs não são apropriadas para resolver todos os tipos de problemas computacionais eficientemente. Certas características devem ser verificadas na aplicação para garantir a eficiência do código, tais como:

1. paralelismo: devido à arquitetura da GPU, na qual as *threads* são executadas em paralelo e de modo independente, ao se mapear um problema para a GPU, os dados da aplicação devem ser independentes, garantindo, assim, o uso dos núcleos de processamento da placa gráfica eficientemente;

2. grande volume de dados: GPUs manipulam usualmente bilhões de dados (*pixels*) para renderização de imagens por segundo, submetendo-os a diversas operações aritméticas. Portanto, o ganho do desempenho de uma aplicação na GPU se deve principalmente ao grande volume de dados processados.

Os primeiros aplicativos que utilizaram a GPU com a finalidade de programação genérica precisaram adaptar sua API para este novo conceito, apresentando uma curva de aprendizado muito lenta para os desenvolvedores. Por este motivo, foram desenvolvidas novas arquiteturas para facilitar a utilização da GPU para processamento genérico. A NVIDIA introduziu em 2007 a plataforma CUDA (Kirk and Hwu, 2010), a qual foi desenvolvida para permitir a utilização conjunta da CPU, para execução sequencial, e GPU, para cálculos numéricos intensos na execução de uma aplicação. Entretanto, esta arquitetura apresentada em 2007 permitia que apenas as placas gráficas da NVIDIA fossem utilizadas com o propósito genérico. Assim, em 2008, um consórcio entre diversas empresas do ramo criou uma plataforma genérica para processamento paralelo, chamada de OpenCL (Tsuchiyama et al., 2009). Esta permite que uma aplicação paralela utilize diferentes *hardware* com capacidade de paralelismo para execução. As duas seções seguintes apresentam as plataformas CUDA e OpenCL, sendo esta última a adotada neste trabalho.

3.2 A plataforma CUDA

A plataforma de programação paralela CUDA, disponível para placas de vídeo produzidas pela NVIDIA a partir da série Geforce 8, aproveita a grande capacidade de processamento aritmético das GPUs para a execução de programas de propósito genérico cuja principal característica seja a natureza paralela de processamento de seus dados. Foram desenvolvidos *drivers* e bibliotecas com suporte ao sistema operacional e uma linguagem similar a linguagem de programação C, com a inclusão de extensões para chamada da API da GPU, facilitando seu aprendizado pelos programadores (Kirk and Hwu, 2010).

As GPUs possuem multiprocessadores denominados *Streaming Multiprocessors* (SMs) com vários núcleos de processamento, chamados de *Streaming Processors* (SPs) responsáveis pelo processamento de *threads*, cada um vinculado a um dado da aplicação. Estes SPs compartilham recursos do SM como a unidade de controle lógico, o *cache* de instruções e a memória, como pode ser visto na Figura 3.2. Também nesta figura, uma outra unidade de processamento é apresentada, a SFU (Unidade de funções especiais), responsável pelo processamento de funções, como senos e cossenos.

Figura 3.2: Organização simplificada de um SM (Kirk and Hwu, 2010).

Esta organização permite a execução de uma mesma instrução em diferentes dados da aplicação, caracterizando o dispositivo como SIMD [2]. Para GPU, o termo SIMT (*Single Instruction, Multiple Thread*) (Tsuchiyama et al., 2009) também é utilizado, pois cada multiprocessador executa centenas de *threads* concorrentemente.

Um programa que utiliza a plataforma CUDA é composto por duas ou mais fases, as quais intercalam a utilização da CPU chamada também de *host*, para configuração de dispositivos, inicialização e configuração de parâmetros, e da GPU, também chamada de *device* ou dispositivo,

[2]SIMD (*Single Instruction, Multiple Data*) faz parte da classificação de sistemas proposta por Michael J. Flynn em 1966. Arquiteturas SIMD são capazes de aplicar instruções em múltiplos conjuntos de dados.

para execução das partes do código que possuam dados a serem processados paralelamente. A parte do programa que irá executar no dispositivo (ou GPU) é denominada de *kernel*, que ao ser invocada, é executada como uma grade de *threads* paralelas. As *threads* em uma grade são organizadas em dois níveis hierárquicos, como mostra a Figura 3.3. Em um primeiro nível, a grade consiste de um ou mais blocos (*thread blocks*), os quais, em um segundo nível, são divididos como *arrays* de *threads*.

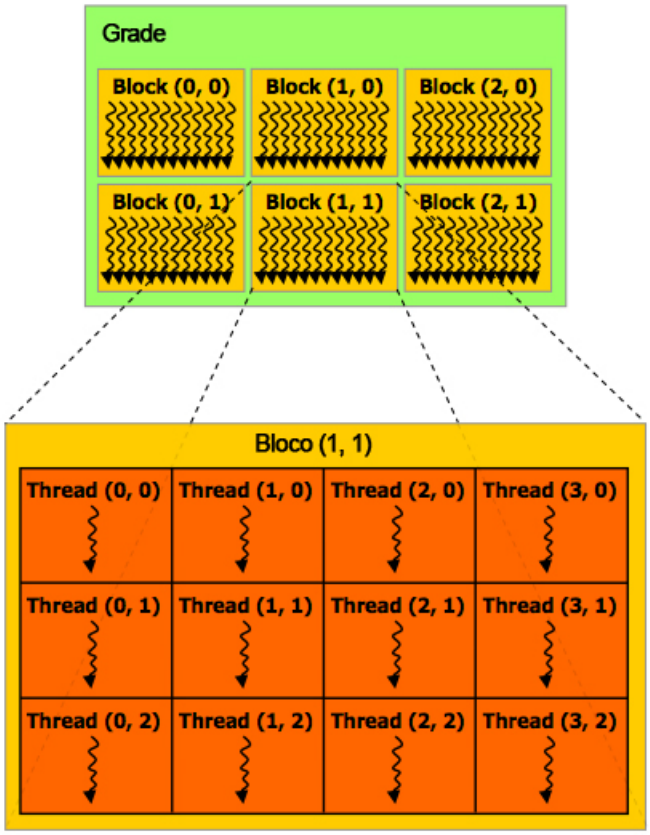


Figura 3.3: Organização das *threads* na GPU. Figura adaptada de (Kirk and Hwu, 2010).

Ao ser atribuida a um SM para execução, além da divisão em blocos, a grade é dividida em grupos chamados de *warps*, os quais são compostos por 32 *threads* que começam sua execução no mesmo endereço do programa. Caso alguma destas *threads* de um *warp* tenha caminhos de execução diferente das outras [3], causado por estruturas de controle de fluxo (*if, for, while, etc*), sua execução passa a ser sequencial, diminuindo consideravelmente o *throughput* de instruções e aumentando o tempo de execução do *kernel* (NVIDIA, 2010).

É importante destacar que cada arquitetura de GPU da NVIDIA possui características técnicas próprias como, por exemplo, o número de *streaming processors* (SPs) em cada *streaming multiprocessor* (SMs), número de blocos e threads máximo por SM e quantidade de memória disponível. Estas especificações são chamadas de *Compute Capability* (CC). Neste trabalho,

[3]Na Seção 4.3.2 este problema será denominado de divergêngia de execução de *threads*.

utilizou-se a placa de video Geforce GTS 250 da NVIDIA, cuja CC é denominada 1.1 e encontra-se descrita no Anexo A.

Nesta seção foram resumidos os principais conceitos referentes à arquitetura e plataforma CUDA, introduzida pela NVIDIA em suas placas gráficas. O entendimento dos termos utilizados é de fundamental importância, pois neste trabalho, foram utilizados apenas dispositivos NVIDIA. Para o desenvolvimento do simulador BioCrowds para GPU, utilizou-se a plataforma de desenvolvimento paralelo OpenCL, pois esta pode ser utilizada independente do dispositivo que se tenha para execução concorrente, visto que esta portabilidade facilita a utlização deste trabalho em futuros trabalhos com o simulador.

3.3 A plataforma OpenCL

A plataforma OpenCL (Tsuchiyama et al., 2009) foi desenvolvida com a finalidade de permitir a criação de programas paralelos que executam em plataformas heterogêneas como unidades lógicas de processamento (CPUs) e unidades de processamento gráfico (GPUs). Ela inclui uma extensão da linguagem de programação C, um compilador e um ambiente para o controle e execução do código escrito nesta linguagem de programação. Este padrão foi desenvolvido pelo consórcio Khronos Group, do qual fazem parte grandes empresas como AMD, Apple, IBM Intel, NVIDIA, Texas Instruments, Sony e Toshiba. O objetivo da criação de um padrão unificado é proporcionar a capacidade de programar utilizando apenas uma linguagem diversos processadores de diferentes fabricantes, como CPU, GPU, Cell/B.E., DSP.

O modelo da plataforma OpenCL consiste de um *host* (CPU) conectado a um ou mais dispositivos (GPU) que suportam a utilização da mesma. Estes dispositivos são divididos em unidades computacionais (*Compute Units*), as quais são compostas por uma ou mais unidades de processamento (*Processing Elements*), como pode ser visto na Figura 3.4.

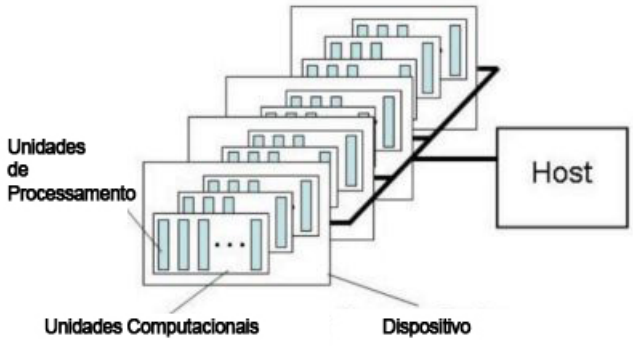


Figura 3.4: Modelo da plataforma OpenCL. Figura adaptada de (Tsuchiyama et al., 2009).

Nesta plataforma, assim como em CUDA, o programa responsável pelo processamento paralelo na GPU [4] é chamado de *kernel*, e a criação de um contexto para a sua execução e

[4] A partir deste momento, irá se restringir o dispositivo com suporte à plataforma OpenCL apenas à GPU, pois este foi o dispositivo utilizado neste trabalho.

gerenciamento é realizado através de uma parte do programa que executa na CPU. Ao se instanciar um *kernel*, este é composto por itens de trabalho (*work-items*), os quais são a unidade mínima de dado a ser processado e que executam concorrentemente em grupos de trabalho (*work-groups*), os quais são organizados em uma grade, chamada de *NDRange*, como mostra a Figura 3.5. Cada item de trabalho possui um posicionamento local, referente ao bloco ao qual pertence e um posicionamento global obtido a partir do tamanho do grupo de trabalho (Sx, Sy) e seu posicionamento em relação ao *NDRange* (wx, wy).

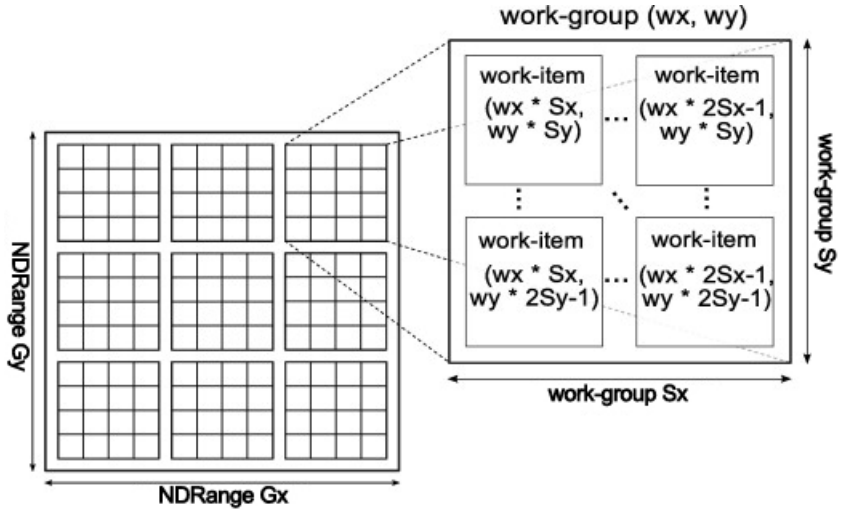


Figura 3.5: Modelo de execução da plataforma OpenCL. Figura adaptada de (Tsuchiyama et al., 2009).

Pode-se perceber a semelhança entre as plataformas OpenCL e CUDA (ver Tabela 3.1), tanto em relação à organização das unidades de processamento como no modo como o programa é organizado hierarquicamente para execução no dispositivo (GPU). Isto se deve ao fato de que os processadores gráficos produzidos pela NVIDIA também podem ser utilizados para programação genérica com a plataforma OpenCL, e esta, por sua vez, deve apresentar uma estrutura genérica para suportar as diferentes plataformas de diferentes empresas que se propõem a utilizá-la.

3.4 Considerações Finais

Neste capítulo foram apresentadas as plataformas CUDA e OpenCL para desenvolvimento de aplicações genéricas utilizando dispositivos gráficos, os quais eram anteriormente utilizados apenas para processamento gráfico. Estes dispositivos SIMD são eficientes quando associados a solução de problemas com grande quantidade de dados independentes a serem processados.

Com a crescente demanda pela utilização de GPUs com propósito de processamento genérico, a NVIDIA criou a plataforma CUDA e uma plataforma para desenvolvimento com o mesmo nome para facilitar a utilização das GPUs com esta finalidade. Entretanto, a plataforma CUDA pode ser apenas utilizada nas placas gráficas produzidas pela empresa. Por esta razão, foi desenvolvida a plataforma OpenCL, a qual tem como principal vantagem em relação à plataforma

CUDA o fato de ser heterogênea, ou seja, pode ser utilizada independente do *hardware* sobre o qual irá ser feito o processamento.

Para este trabalho, utilizou-se as placas de vídeo da NVIDIA, e escolheu-se a plataforma OpenCL para o desenvolvimento do simulador de multidões paralelo deste trabalho, pela portabilidade que ele proporciona. Utiliza-se no decorrer deste trabalho as terminologias referentes à plataforma CUDA, por serem mais intuitivas e comuns. A Tabela 3.1 faz um mapeamento das terminologias de ambas as plataformas CUDA e OpenCL.

CUDA	**OpenCL**
Grid	*NDRange*
Thread block	*Work-group*
Thread	*Work-item*

Tabela 3.1: Mapeamento entre alguns dos termos utilizados em CUDA e OpenCL.

Capítulo 4

BioCrowds paralelo

Neste capítulo, será abordado o desenvolvimento do trabalho proposto nesta dissertação, apresentando os passos tomados para a paralelização do modelo Biocrowds e a inclusão do efeito de pressão (*pushing*). A partir deste momento, os indivíduos da simulação serão chamados de agentes, denominação utilizada por Bicho (2009) em sua tese de doutorado.

4.1 O modelo BioCrowds

O BioCrowds, proposto por Bicho (2009), é um modelo para simulação de multidões baseado no algoritmo de colonização de espaço apresentado por Runions et al. (2005) para simular o crescimento de nervuras em folhas, galhos e ramos de árvores. O espaço disponível para o crescimento da lâmina da folha é identificado pela presença de um hormônio chamado auxina, responsável por guiar o crescimento da folha devido à competição por espaço pelas nervuras da folha.

Na abordagem proposta por Bicho, o crescimento das nervuras pode ser interpretado como a movimentação de um agente em um cenário virtual, sendo os espaços livres para deslocamento identificados pela existência de marcadores. Estes marcadores são dispostos aleatoriamente nas regiões do cenário virtual através da utilização do algoritmo de lançamento de dardos [1], o qual manteve-se neste trabalho. Os espaços onde há obstáculos a serem evitados são determinados pela ausência de marcadores na região. Cada agente compete pela posse de marcadores em cada iteração do simulador e como resultado obtém-se o seu deslocamento. A Figura 4.1 ilustra quatro agentes em um cenário e seus respectivos campos de percepção de marcadores.

Assim, o algoritmo BioCrowds pode ser dividido em quatro etapas: inicialização do modelo, cálculo do agente mais próximo de cada marcador, cálculo do vetor movimento instantâneo de cada agente e atualização da posição do agente.

Para a inicialização do modelo, define-se para cada agente I_i os seguintes parâmetros individuais: sua posição atual $\vec{x_i}(t)$, seu objetivo atual $\vec{g_i}(t)$, o campo de percepção, definido como uma região circular de raio R_i, responsável pela determinação da máxima distância em que

[1] O algoritmo de lançamento de dardo proposto por Cook (1986), apud Bicho (2009), gera pontos distribuidos de acordo com uma distribuição de Poisson. Pontos são rejeitados caso estes não satisfaçam a distância mínima dos pontos anteriormente gerados pelo algoritmo (Lagae, 2009).

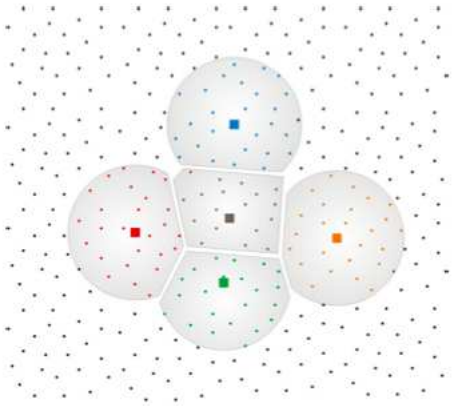


Figura 4.1: O espaço pessoal (regiões hachuradas) e os marcadores (pontos) associados a cinco agentes (pequenos quadrados). Os marcadores são mostrados na mesma cor do agente que os contêm em seu espaço pessoal (Bicho, 2009).

um agente pode perceber um marcador e o deslocamento máximo do agente por segundo da simulação s_{max} [2]. Para definir o valor do deslocamento máximo do agente em cada iteração, s'_{max}, divide-se o valor de s_{max} pela taxa de quadros por segundo (FPS) desejada. Além destes parâmetros referentes aos agentes, define-se a área da simulação da seguinte forma:

- cenário no qual os agentes se deslocarão, definindo obstáculos e área livres para movimentação;
- número de agentes e suas posições iniciais;
- destinos definidos para indivíduos, dependendo da aplicação;
- densidade de marcadores μ que serão adicionados à área da simulação.

Na segunda etapa, cria-se o conjunto A_i que contém todos os marcadores que estejam dentro do campo de percepção do agente I_i, mais próximos a ele do que de qualquer outro agente no cenário. Este conjunto é representado por $A_i = \{\vec{a_1}, \vec{a_2}, \ldots, \vec{a_N}\}$, onde N é o número total de marcadores relacionados ao agente I_i, $\vec{a_k}$ a posição do marcador k e $\vec{x_i}$ a posição do agente i. A'_i é o conjunto de vetores de orientação definido a seguir:

$$A'_i = \{\vec{d_1}, \vec{d_2}, \ldots, \vec{d_n}\}, \quad onde \quad \vec{d_k} = \vec{a_k} - \vec{x_i}. \tag{4.1}$$

Na terceira etapa, para calcular o vetor movimento instantâneo do agente, define-se primeiramente um vetor de tentativa de movimento $\vec{m}$, o qual permite que o vetor objetivo do agente seja levado em consideração para o cálculo de seu deslocamento. Assim, o vetor $\vec{m}$ é computado da seguinte forma:

[2] Para facilitar a leitura do texto, o parâmetro de tempo t será omitido a partir deste ponto.

$$\vec{m} = \sum_{k=1}^{N} w_k \vec{d_k}. \tag{4.2}$$

Nesta equação são utilizados coeficientes w_k para cada vetor de orientação $\vec{d_k}$ de acordo com o grau de alinhamento deste com o objetivo $\vec{g_i}$ do agente. Este peso é determinado empiricamente pela seguinte equação:

$$w_k = \frac{f(\vec{g}, \vec{d_k})}{\sum_{l=1}^{N} f(\vec{g}, \vec{d_l})}. \tag{4.3}$$

Para determinar uma função f para o modelo, em sua tese Bicho (2009) assumiu inicialmente que todos os marcadores $\vec{a_k}$ que influenciam o agente I_i estão a uma mesma distância $||\vec{d_k}||$ deste agente. Assim, a função f deve satisfazer as seguintes proposições:

1. atingir seu valor máximo quando o ângulo θ entre o vetor objetivo $\vec{g_i}$ e o vetor $\vec{d_k}$ for igual a 0°;
2. atingir seu valor mínimo quando $\theta = 180°$;
3. decrescer monotonicamente a medida que θ cresce de 0° a 180°;
4. apresentar valores maiores ou iguais a zero.

No caso em que as distâncias $||\vec{d_k}||$ sejam diferentes, os marcadores mais distantes do agente devem influenciar menos o seu movimento, portanto ter pesos relativamente menores. Assim, f foi definida como:

$$f(\vec{g}, \vec{d_k}) = \begin{cases} \frac{1+\cos\theta}{1+||\vec{d_k}||} = \frac{1}{1+||\vec{d_k}||}\left(1 + \frac{<\vec{g}, \vec{d_k}>}{||\vec{g}||||\vec{d_k}||}\right) & \text{se } ||\vec{d_k}|| > 0, \\ 0 & \text{se } ||\vec{d_k}|| = 0. \end{cases} \tag{4.4}$$

Para calcular um deslocamento instantâneo $\vec{h_i}$ do agente, é utilizado o vetor $\vec{m}$ de tentativa de movimento e o deslocamento máximo do agente s'_{max} por quadro. Assim, a terceira etapa do algoritmo é obtida através da seguinte equação:

$$\vec{h_i} = s\frac{\vec{m}}{||\vec{m}||}, \quad onde \quad s = \min\{||\vec{m}||, s'_{max}\}. \tag{4.5}$$

Portanto, na quarta etapa do algoritmo, a posição do agente é atualizada através de

$$\vec{x}(t+1) = \vec{x}(t) + \vec{h_i}. \tag{4.6}$$

Se considerado que os agentes possuem tamanho infinitesimal, a abordagem proposta por Bicho para a simulação dinâmica de multidões é livre de colisões, independentemente do número de agentes, da densidade populacional da multidão ou de marcadores, conforme discutido em sua tese e em Bicho et al. (2012). Entretanto, é comum em simulações considerar que cada agente

I_i ocupe um espaço finito, representado por um círculo de raio r_i. Neste caso, o centro de dois agentes vizinhos nunca colidirão, mas os seus círculos podem interseccionar. Para tratar esta questão, Bicho utilizou o algoritmo de *Convex Hull* (O'Rourke, 1998), que calcula o polígono convexo que circunscreve um conjunto de pontos (Bicho, 2009). Desta forma, o autor obteve uma segunda equação para o vetor de tentativa de movimento agora denominado $\vec{m'}$:

$$\vec{m'} = \beta\vec{m}, \tag{4.7}$$

onde

$$\beta = \begin{cases} \vec{d_{ch}}/||\vec{m}|| & \text{, se } ||\vec{m}|| > \vec{d_{ch}} \\ 1 & \text{, demais casos.} \end{cases} \tag{4.8}$$

Na Equação 4.7, β ajustará o módulo do vetor de movimento $\vec{m}$ ao máximo deslocamento possível $\vec{d_{ch}}$ do agente I_i considerando seu *Convex Hull*.

Finalmente, $\vec{h_i}$ é calculado similarmente a Equação (4.5), ou seja,

$$\vec{h_i} = s\frac{\vec{m'}}{||\vec{m'}||}, \quad onde \quad s = \min\{||\vec{m'}||, s'_{max}\}. \tag{4.9}$$

O Algoritmo 1 apresenta as principais etapas do cálculo do deslocamento dos agentes no modelo para simulação de multidões proposto por Bicho.

Pode-se perceber que a posição de cada agente é atualizada a cada iteração do programa, dependendo para este cálculo apenas dos marcadores associados.

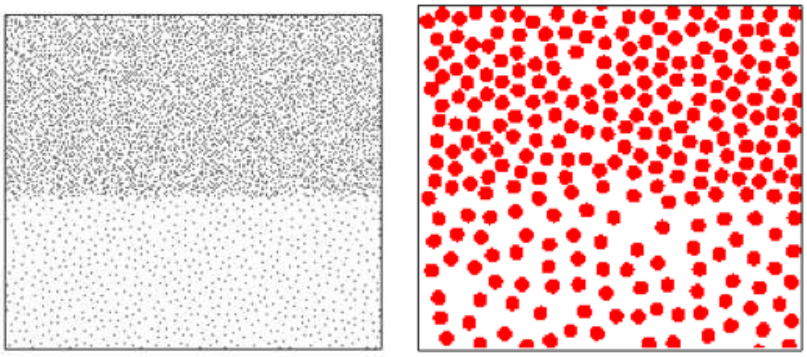
Ainda neste modelo, dependendo da densidade, da disponibilidade de marcadores e da importância dada ao objetivo na função f de ponderação dos marcadores (Equação 4.4), é possível que os agentes aproximem-se entre si, mas não cheguem a colidir. Quando indivíduos em uma multidão real ficam muito próximos entre si, surge o comportamento emergente de pressão entre os agentes, também chamado de *pushing*. Segundo Helbing et al. (2000) e Pelechano et al. (2007) o *pushing* ocorre quando há o contato físico entre os indivíduos da multidão, exigindo que pressões sejam simuladas entre os mesmos.

Em sua tese, Bicho (2009) discute a hipótese da inclusão do comportamento emergente *pushing* ou efeito de pressão no BioCrowds, comportamento este que não é simulado pelo modelo. Uma hipótese levantada por Bicho foi a adoção de uma função f alternativa à apresentada na Equação 4.4 para ponderação dos marcadores de cada agente. Nesta alteração deveria ocorrer a minimização da diferença angular entre o vetor objetivo g e o vetor d calculado para cada marcador do agente. Desta forma, o comportamento *pushing* poderia ser reproduzido no modelo BioCrowds, mesmo que este não simule pressões entre os agentes próximos.

Este problema de inclusão do comportamento de *pushing* no BioCrowds foi também abordado por Pinheiro (2010). Em seu trabalho, Pinheiro propôs a alteração da densidade local dos marcadores do cenário de forma interativa onde houvesse acúmulo de agentes. Esta maior densidade permite que a distância entre os indivíduos simulados seja menor, implicando em uma maior utilização do espaço disponível para locomoção. A Figura 4.2 ilustra o aumento da proximidade entre os agentes com o aumento da densidade de marcadores.

Algoritmo 1: Algoritmo para simulação de multidões - BioCrowds

Entrada: *FPS, agentes, S,* $s_{max} \geqslant 0$, $R \geqslant 0$, $r \geqslant 0$
Saída: *agentes*
$s_{max}^{'} = s_{max}/FPS$
repita
  para *cada agente* i **faça**
    $A_i = \emptyset$
  fim para
  para *cada marcador* $\vec{a} \in A$ **faça**
    $i = agenteMaisProximo(\vec{a})$
    se *distância*$(\vec{x_i}, \vec{a}) \leq R_i$ **então**
      $A_i = A_i \cup \{\vec{a}\}$
    fim se
  fim para
  para *cada agente* i **faça**
    se $A_i \neq \emptyset$ **então**
      $A_i^{'} = \{\vec{d} \quad | \quad \vec{a} \in A_i, \vec{d} = \vec{a} - \vec{x_i}\}$
      $\vec{m_i} = \sum_{\vec{d} \in A_i^{'}} w\vec{d}$, *onde* $w = f(\vec{g_i}, \vec{d}) / \sum_{\vec{d} \in A_i^{'}} f(\vec{g_i}, \vec{d})$
      se *representação do agente for finita* **então**
        $\vec{m_i}^{'} = calculaConvexHull(r_i, A_i, \vec{m_i})$
        $\vec{h_i} = (\vec{m_i}^{'}/||\vec{m_i}^{'}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}^{'}||, s_{max}^{'})$
      senão
        $\vec{h_i} = (\vec{m_i}/||\vec{m_i}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}||, s_{max}^{'})$
      fim se
      $\vec{x_i} = \vec{x_i} + \vec{h_i}$
    fim se
  fim para
até *encerrar simulação*;



(a) diferentes densidades de marcadores.

(b) Agentes simulados.

Figura 4.2: (a) Representação de diferentes densidades em um cenário e (b) a diferença na disposição dos agentes.

Entretanto, esta alteração na proposta de Pinheiro (2010) ocorre de forma premeditada, com a definição antecipada das regiões do cenário factíveis para ocorrerência do comportamento *pushing* e o cálculo da posição dos novos marcadores a serem adicionados é realizada em uma etapa de pré-processamento.

O trabalho atual apresenta uma solução alternativa para a inclusão do comportamento emergente de *pushing*, presente em multidões reais. Sua implementação será detalhada na próxima seção.

4.2 Modelagem e implementação do efeito de pressão

O presente trabalho apresenta uma solução genérica para a inclusão do comportamento *pushing* no modelo BioCrowds através da alteração de forma não premeditada dos marcadores de uma região do cenário de acordo com a necessidade no decorrer da simulação.

Nesta solução, faz-se o monitoramento constante das regiões do cenário onde há agentes estagnados durante a simulação de uma multidão no modelo BioCrowds. Um agente é considerado estagnado quando sua posição permanece a mesma durante 30 iterações seguidas do algoritmo. Ao se observar que o um número de agentes estagnados utrapassou o limite de 5% do total de agentes na simulação em qualquer região do cenário, criam-se novos marcadores para o local, aumentando, assim, a densidade local e a disponibilidade de marcadores.

Para determinar a região onde deve ocorrer a mudança na densidade local de marcadores, utiliza-se o algoritmo que calcula o polígono convexo que circuscreve um conjunto de pontos, denominado *Convex Hull* (O'Rourke, 1998). A Figura 4.3 ilustra o *Convex Hull* de um conjunto de pontos em um espaço bidimensional.

No caso do modelo BioCrowds, estes pontos são os agentes estagnados em uma determinada região do cenário. Esta situação pode ocorrer devido à existência de obstáculos não móveis, como paredes, representadas no modelo BioCrowds através da ausência de marcadores, ou obstáculos móveis, como outros agentes da simulação. A Figura 4.4 mostra o *Convex Hull* calculado para um conjunto de agentes parados devido a uma parede presente no cenário simulado.

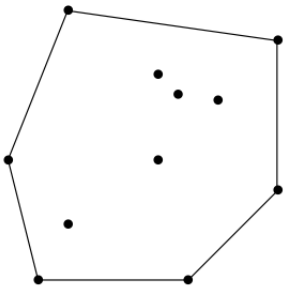


Figura 4.3: *Convex Hull* calculado para um conjunto de pontos em um espaço 2D.

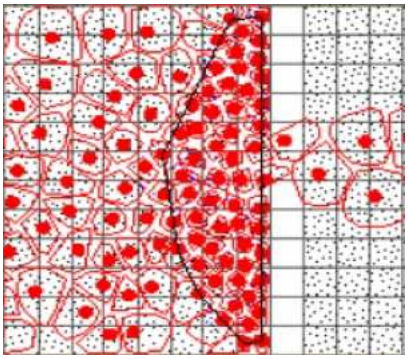


Figura 4.4: *Convex Hull* calculado para os agentes estagnados em um cenário composto apenas por uma saída estreita.

Definida a área do *Convex Hull*, geram-se novos marcadores para a mesma, utilizando o algoritmo de "lançamento do dardo" (*dart-throwing algorithm*) (Cook, 1986; Mitchell, 1987 *apud* Bicho, 2009). Com este algoritmo, a posição dos novos marcadores são geradas randomicamente, com uma distribuição uniforme, apenas sobre a área definida anteriormente pelo *Convex Hull*. Uma nova posição é rejeitada se a sua distância em relação aos demais marcadores já existentes no cenário for menor do que um limite previamente estabelecido. Satisfeita esta restrição, o novo marcador é adicionado ao cenário simulado.

Este método permite que os agentes se concentrem muito próximos entre si em áreas específicas do cenário sem a prévia definição das mesmas, simulando o comportamento de *pushing* no modelo BioCrowds conforme apresentado no Capítulo 5. O Algoritmo 2 apresenta as etapas propostas nesta tese. Nele, o conjunto B representa os agentes estagnados na simulação e o *Convex Hull* calculado a partir dos mesmos é definido como o conjunto C.

4.3 O algoritmo BioCrowds paralelo para simulação de multidões

Tendo em vista a mudança do paradigma computacional ao se executar programas paralelos, o modelo para simulação de multidões, BioCrowds, desenvolvido por Bicho (2009) em seu

Algoritmo 2: Algoritmo para inclusão do efeito de pressão no modelo BioCrowds

Entrada: *agentes*
Saída: *novos marcadores*
$B = \emptyset$
$MaxAgentesEstagnados = 0,05 * TotalAgentes$
para *cada agente* i **faça**
  $n_i = NumeroQuadrosEstagnado(i)$
  se $n_i = 30$ **então**
    $B = B \bigcup i$
  fim se
fim para
se $B \neq \emptyset \quad e \quad B \geq MaxAgentesEstagnados$ **então**
  $C = calculaConvexHull(B)$
  $adicionaNovosMarcadores(C)$
fim se

doutorado, foi adaptado para sua utilização com processadores com arquitetura SIMD, como as GPUs. Para isto, foram definidas duas etapas de desenvolvimento. A primeira, refere-se a identificação de partes do algoritmo que poderiam ser implementadas de forma paralela. Na segunda etapa, o algoritmo foi implementado para processamento utilizando GPUs.

4.3.1 Definição dos *kernels* do algoritmo

O algoritmo BioCrowds foi desenvolvido para ser executado sequencialmente, utilizando a CPU como *hardware* de processamento. Para sua execução em dispositivos SIMD como as GPUs, o modelo para simulação de multidões precisou ser adaptado para este novo paradigma de processamento.

A GPU, como visto no Capítulo 3 é composta por diversas unidades lógicas e aritméticas (ULAs) responsáveis pelo processamento da aplicação. Esta arquitetura permite que vários dados sejam processados simultaneamente nos diversos núcleos do dispositivo. Portanto, o paralelismo associado a este hardware é o paralelismo de dados.

O paralelismo de dados consiste na execução de um mesmo programa por diferentes processadores, em diferentes conjuntos de dados. Este método é eficiente apenas quando não há dependência entre os dados processados, permitindo que estes executem paralelamente. Desta forma, o número de processadores influencia diretamente no ganho de desempenho adquirido ao se utilizar este tipo de arquitetura. A Figura 4.5 exemplifica uma soma de vetores onde cada elemento está sendo processado por um único processador do dispositivo.

Da mesma forma, para realizar o mapeamento do algoritmo BioCrowds sequencial para paralelo, foi preciso identificar o cálculo a ser realizado pelas unidades de processamento paralelo, ou também chamada de *thread* (Kirk and Hwu, 2010). Assim, analisou-se o algoritmo BioCrowds utilizando dois critérios referentes as características específicas do hardware, como discutido na Seção 3.1, que são em quais trechos do algoritmo há processamento de grande volume de dados e se estes são independentes entre si.

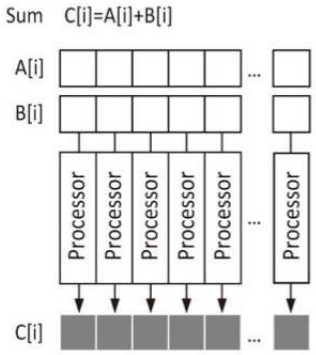


Figura 4.5: Soma de dois vetores A e B, onde cada operação é independente e processada paralelamente em diferentes elementos e em diferentes processadores (Tsuchiyama et al., 2009).

Desta forma, foram identificadas duas partes do código possíveis de serem implementados de forma paralela [3], que são a definição dos marcadores mais próximos de cada agente e o cálculo de uma nova posição para o agente, os quais no Algoritmo 1 são respectivamente a criação do conjunto A e o cálculo do deslocamento $\vec{h_i}$ do agente.

No primeiro trecho, para cada marcador $\vec{a}$ existente no cenário, acha-se o agente i mais próximo do marcador e que esteja a uma distância menor ou igual a R_i pré-determinada do mesmo, chamada de campo de percepção. No segundo trecho, o cálculo de uma nova posição é realizado para cada agente simulado. Em ambos, os cálculos necessários são realizados em um grande volume de dados.

Neste primeiro *kernel*, cada *thread* criada para processamento de uma unidade de dado é associada a um marcador, e o resultado obtido é salvo em uma estrutura definida a partir do número de agentes na simulação. Com a paralelização do algoritmo, diferentes *threads* poderão acessar ou modificar o mesmo dado da aplicação ao mesmo tempo, como é mostrado na Figura 4.6, o que não garante a integridado do resultado gerado.

Por este motivo, este trecho do algoritmo BioCrowds original precisou ser dividido em duas etapas de processamento, cada uma relacionada a qual dado será mapeado para as *threads* do problema: um primeiro *kernel* define um único agente mais próximo de cada marcador que esteja dentro de seu campo de percepção R, relacionando cada *thread* a um marcador $\vec{a}$, criando assim, uma dupla "marcador-agente" $D_{ai} = (\vec{a_k}, I_i)$.

Tendo esta estrutura montada, as *threads* são associadas a cada agente i em um segundo *kernel*. Desta forma, cria-se o conjunto de marcadores A_i de cada agente i da simulação sem que haja acesso simultâneo de memória pelas *thread* em execução. A Figura 4.7 ilustra os passos para a definição dos marcadores de cada agente no algoritmo proposto.

Portanto, o mapeamento do algoritmo BioCrowds para processamento paralelo em GPU resultou na criação dos seguintes *kernels*:

- *kernel* 1 (paralelização em relação aos marcadores): identificação do agente I_i mais próximo de cada marcador $\vec{a_k}$. O agente deve estar na área de percepção do marcador, a qual

[3]Estes trechos de código serão chamados de kernel, nomenclatura utilizada pelo OpenCL.

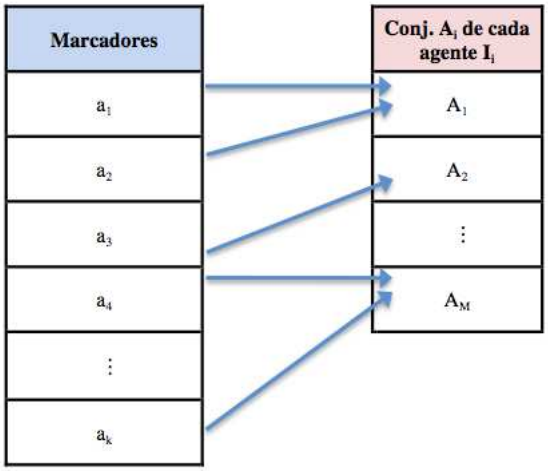


Figura 4.6: Cada *thread* associada a um marcador do cenário é responsável pela definição de um único agente mais próximo do mesmo, adicionando o marcador ao conjunto A_i do agente i no algoritmo BioCrowds sequencial. Neste caso, duas ou mais *threads* poderão acessar ao mesmo tempo o endereço de memória associado ao conjunto A_i, podendo gerar resultados diferentes do esperado.

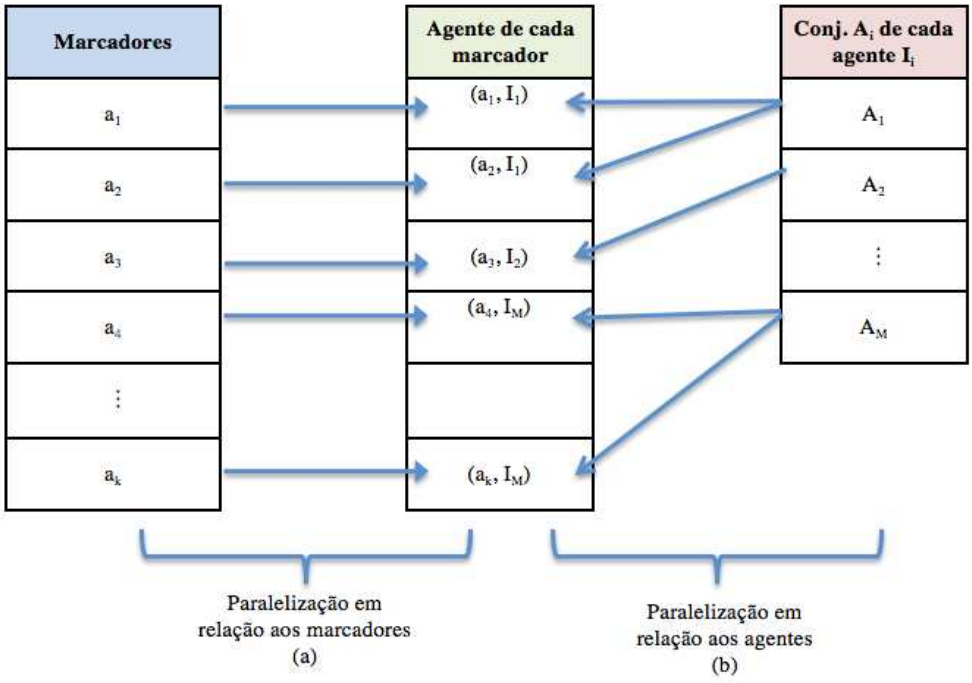


Figura 4.7: (a) Definição dos marcadores de cada agente no algoritmo BioCrowds paralelo e criação da dupla "marcador-agente". (b) Associação de cada marcador ao conjunto de marcadores de cada agente. Em cada etapa, as *threads* são associadas a dados diferentes garantindo, desta forma, que apenas uma *thread* manipule os endereços de memória onde os resultados são armazenados.

é igual ao raio de percepção R do agente (Algoritmo 3).

- *kernel* 2 (paralelização em relação aos agentes): definição de todos os marcadores $\vec{a}$ mais próximos do agente. Em seguida, calcula-se a próxima posição do agente conforme o algoritmo BioCrowds original (Algoritmo 4).

Algoritmo 3: *Kernel* 1 para seleção do agente mais próximo de cada marcador e que esteja dentro de seu raio de percepção na simulação

Entrada: *agentes, marcadores*
Saída: D_{ai}
para *cada marcador* $\vec{a} \in A$ **faça**
  menorDistancia $\leftarrow -\infty$
  para *cada agente* i **faça**
    $d_i =$ distância$(\vec{x_i}, \vec{a})$
    se $d_i \leq R_i$ *e* $d_i \leq menorDistancia$ **então**
      $menorDistancia = d_i$
      $D_{ai} = (\vec{a}, i)$
    fim se
  fim para
fim para

Algoritmo 4: *Kernel* 2 para seleção do conjunto de marcadores de cada agente e cálculo de seu próximo passo

Entrada: *FPS, agentes, S,* $s_{max} \geqslant 0$, $R \geqslant 0$, $r \geqslant 0$, D_{ai}
Saída: *agentes*
para *cada agente* i **faça**
  para *cada marcador* $\vec{a} \in A$ **faça**
    $I_a = recuperarAgenteDoMarcador(\vec{a}, D_{ai})$
    se $I_a = i$ **então**
      $A_i = A_i \cup \{\vec{a}\}$
    fim se
  fim para
  se $A_i \neq \emptyset$ **então**
    $A_i^{'} = \{\vec{d} \quad | \quad \vec{a} \in A_i, \vec{d} = \vec{a} - \vec{x_i}\}$
    $\vec{m_i} = \sum_{\vec{d} \in A_i^{'}} w\vec{d}$, *onde* $w = f(\vec{g_i}, \vec{d}) / \sum_{\vec{d} \in A_i^{'}} f(\vec{g_i}, \vec{d})$
    se *representação do agente for finita* **então**
      $\vec{m_i}^{'} = calculaConvexHull(r_i, A_i, \vec{m_i})$
      $\vec{h_i} = (\vec{m_i}^{'} / ||\vec{m_i}^{'}||) s_{min}$, *onde* $s_{min} = min(||\vec{m_i}^{'}||, s_{max}^{'})$
    senão
      $\vec{h_i} = (\vec{m_i} / ||\vec{m_i}||) s_{min}$, *onde* $s_{min} = min(||\vec{m_i}||, s_{max}^{'})$
    fim se
    $\vec{x_i} = \vec{x_i} + \vec{h_i}$
  fim se
fim para

4.3.2 Implementação do algoritmo paralelo

A implementação de um algoritmo para processamento em GPU deve ser discutida especificamente para o hardware que será utilizado (Kirk and Hwu, 2010). Neste trabalho, a placa de vídeo utilizada foi a NVIDIA GTS 250, cujas especificações computacionais (*compute capability - CC*) são referentes à CC 1.1 e se encontram no Anexo A.

De acordo com a *Compute Capability* 1.1, esta placa de vídeo tem um total de 16 *streaming multiprocessor* (SMs), sendo cada um composto por 8 *streaming processors* (SPs), totalizando 128 unidades de processamento, chamadas no contexto das placas NVIDIA de *cuda-cores*. Através deste documento, sabe-se também que o número máximo de *threads* que pode residir em um SM simultaneamente é igual a 768 *threads* organizadas na forma de 3 blocos de 256 *threads* cada, 6 blocos de 128 *threads* cada, ou outras combinações.

Ao se atribuir um bloco para processamento em um SM, o mesmo é dividido em grupos de 32 *threads*, chamados de *warps* (ver Seção 3.2), que são as unidades mínimas para escalonamento em um SM. Quando uma instrução executada pelas *threads* de um *warp* necessitam esperar o resultado de uma operação com latência muito grande, como é o caso do acesso à memória global, o *warp* não é selecionado para execução. Neste caso, outro *warp* residente no SM é selecionado.

Desta forma, a GPU consegue executar eficientemente programas cuja latência das operações seja muito grande. Isto implica que um número grande de *warps* residentes em um SM deve ser utilizado.

Sabendo que a placa de vídeo utilizada na implementação do algoritmo BioCrowds paralelo tem um limite de 32 *warps* e 768 *threads* em um SM, utilizou-se neste trabalho um total de 128 *threads* por bloco, garantindo-se a utilização de um número grande blocos sem exceder o limite em um SM. A Tabela 4.1 mostra um comparativo entre o número de *threads* em um bloco para um SM. O cálculo realizado para obtenção destes valores é explicado no Anexo B deste trabalho.

Bloco	Num. blocos	*warps*/bloco	Total de *warps*	OBS
512	2	16	32	excede o número de *warps* por SM
256	3	8	24	-
128	6	4	24	-
64	12	2	24	excede o número de blocos por SM
32	24	1	24	excede o número de blocos por SM

Tabela 4.1: Para cada tamanho de bloco, acha-se o número de *warps* residentes em um SM. Nos casos em que o número de blocos é excedido, apenas 8 blocos são utilizados, implicando em um menor número de *threads* sendo executado, ocorrendo a subutilização do dispositivo.

É importante ressaltar que a divergência na execução de instruções pelas *threads* em um *warp* (ver Capítulo 3) também acarreta um aumento na latência das operações. Esta divergência é causada pela utilização de estruturas de controle de fluxo como *if-else, switch, do, for e while*, presentes em diversas linguagens de programação. Ao utilizá-las, as *threads* pertencentes a um *warp* podem seguir fluxos de execução diferentes, causando a serialização das instruções que estiverem sendo executadas, afetando significativamente o tempo de processamento.

Para evitar que este problema ocorresse no simulador BioCrowds paralelo, estas estruturas foram substituídas por operações matemáticas, como multiplicações e comparações, sempre que possível. A Figura 4.8 apresenta um exemplo de alteração no algoritmo BioCrowds. Nesta mudança a estrutura *if-else* é substituída por uma operação ternária presente na linguagem de programação C [4]. Desta forma, *threads* de um mesmo *warp* podem apresentar um resultado diferente para a variável em questão, entretanto, estarão executando a mesma instrução, sem divergência de fluxo de execução.

se *representação finita dos agentes* **então**
  $\vec{m_i}' = calculaConvexHull(r_i, A_i, \vec{m_i})$
  $\vec{h_i} = (\vec{m_i}'/||\vec{m_i}'||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}'||, s'_{max})$
senão
  $\vec{h_i} = (\vec{m}/||\vec{m_i}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}||, s'_{max})$
fim se

(a)

$\vec{m_i}' = (representação\ finita\ de\ agentes)\ ?\ calculaConvexHull(r_i, A_i, \vec{m_i}) : \vec{m_i}'$
$\vec{v_i} = (\vec{m}/||\vec{m_i}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}||, s'_{max})$

(b)

Figura 4.8: A Figura (a) apresenta um trecho do algoritmo BioCrowds sequencial enquanto que a Figura (b) apresenta o mapeamento deste mesmo trecho para execução na GPU.

O algoritmo BioCrowds paralelo final pode ser visto no Algoritmo 5. É importante ressaltar que as estruturas de controle de fluxo foram substituidas em sua implementação, entretanto, preferiu-se mantê-las no algoritmo para melhor entendimento do mesmo.

4.4 Considerações Finais

Este capítulo apresentou as principais contribuições do trabalho, que são o desenvolvimento e implementação do modelo para simulação de multidões BioCrowds para processamento em hardwares SIMD, como as GPUs, bem como a inclusão de um comportamento emergente em multidões reais, chamado de *pushing*, o qual não era simulado no modelo inicial proposto por Bicho (2009). Em sua implementação, o *hardware* utilizado foi levado em consideração para conseguir o aproveitamento máximo das características de processamento da GPU. Para isto, fez-se uma comparação entre as possíveis estruturas de blocos e *threads*, achando a melhor combinação de blocos com 128 *threads*. O modelo final para simulação é apresentado no Algoritmo 5.

[4] A operação ternária é utilizada para simplificação de estruturas condicionais compostas, apresentando a seguinte forma: variavel = <condição> ? <resultado, se verdadeiro> : <resultado, se falso>.

Algoritmo 5: Algoritmo BioCrowds paralelo para simulação de multidões.

Entrada: *FPS, agentes, marcadores, S,* $s_{max} \geqslant 0$, $R \geqslant 0$, $r \geqslant 0$
Saída: *agentes*
$s_{max}^{'} = s_{max}/FPS$
$B = \emptyset$
$MaxAgentesEstagnados = 0,05 * TotalAgentes$
repita
  para *cada marcador* $\vec{a} \in A$ **faça**
    menorDistancia $\leftarrow -\infty$
    para *cada agente* i **faça**
      $d_i =$ distância$(\vec{x_i}, \vec{a})$
      se $d_i \leq R_i$ *e* $d_i \leq menorDistancia$ **então**
        $menorDistancia = d_i$
        $D_{ai} = (\vec{a}, i)$
      fim se
    fim para
  fim para
  para *cada agente* i **faça**
    para *cada marcador* $\vec{a} \in A$ **faça**
      $I_a = recuperarAgenteDoMarcador(\vec{a}, D_{ai})$
      se $I_a = i$ **então**
        $A_i = A_i \cup \{\vec{a}\}$
      fim se
    fim para
    se $A_i \neq \emptyset$ **então**
      $A_i^{'} = \{\vec{d} \quad | \quad \vec{a} \in A_i, \vec{d} = \vec{a} - \vec{x_i}\}$
      $\vec{m_i} = \sum_{\vec{d} \in A_i^{'}} w\vec{d}$, *onde* $w = f(\vec{g_i}, \vec{d}) / \sum_{\vec{d} \in A_i^{'}} f(\vec{g_i}, \vec{d})$
      se *representação do agente for finita* **então**
        $\vec{m_i}^{'} = calculaConvexHull(r_i, A_i, \vec{m_i})$
        $\vec{h_i} = (\vec{m_i}^{'}/||\vec{m_i}^{'}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}^{'}||, s_{max}^{'})$
      senão
        $\vec{h_i} = (\vec{m_i}/||\vec{m_i}||)s_{min}$, *onde* $s_{min} = min(||\vec{m_i}||, s_{max}^{'})$
      fim se
      $\vec{x_i} = \vec{x_i} + \vec{h_i}$
      $n_i = NumeroQuadrosEstagnado(i)$
      se $n_i = 30$ **então**
        $B = B \bigcup i$
      fim se
    fim se
  fim para
  se $B \neq \emptyset \quad e \quad B \geq MaxAgentesEstagnados$ **então**
    $C = calculaConvexHull(B)$
    $adicionaNovosMarcadores(C)$
  fim se
até *encerrar simulação*;

Capítulo 5

Resultados Obtidos

Este capítulo apresenta os resultados experimentais obtidos ao simular multidões utilizando o algoritmo BioCrowds paralelo proposto na dissertação. Estes resultados permitem analisar o algoritmo quantitativamente em relação ao algoritmo proposto por Bicho (2009) em sua tese de doutorado, avaliando o ganho no desempenho computacional com a utilização de GPUs como hardware de processamento genérico. Os parâmetros utilizados para validação do algoritmo paralelo são a quantidade de agentes que não chegaram ao seu destino, a densidade populacional, a velocidade, a variação angular da orientação dos agentes e a taxa de quadros por segundo da simulação. Por fim, a inclusão do efeito de pressão (*pushing*) no algoritmo foi testada.

5.1 Comparação entre os algoritmos BioCrowds paralelo e sequencial

Nesta seção são apresentados os resultados e a análise dos experimentos realizados para validação do algoritmo BioCrowds paralelo. Esta validação foi realizada através da comparação entre os resultados obtidos no algoritmo proposto nesta dissertação com o algoritmo sequencial desenvolvido por Bicho (2009) em sua tese de doutorado.

Foram realizados dois experimentos com o algoritmo desenvolvido neste trabalho: o primeiro analisa se os principais parâmetros do BioCrowds original são mantidos no algoritmo paralelo proposto, enquanto que o segundo experimento avalia se o comportamento emergente de formação de vias de pedestres, como descrito no Capítulo 2, é simulado. Ambos os experimentos utilizam um cenário composto por um corredor de 40 m de comprimento por 12 m de largura onde um grupo de agentes se move de uma extremidade a outra, na mesma direção, mas em sentidos opostos. A Figura 6.1 apresenta o cenário simulado, onde as setas indicam o sentido de deslocamento dos grupos de agentes.

Definido o cenário das simulações, os seguintes critérios foram utilizados para avaliação do BioCrowds paralelo [1]:

- velocidade média dos agentes, denominada $\bar{v}$ (m/s);

[1] Tanto o cenário utilizado quanto as variáveis de análise para validação do algoritmo BioCrowds paralelo são os mesmos utilizados por Bicho em seu trabalho.

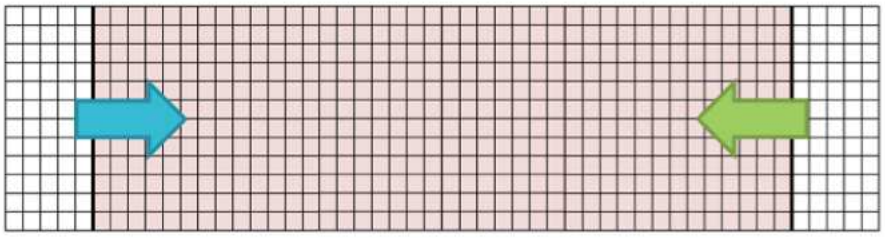


Figura 5.1: Cenário utilizado para comparação entre os algoritmos paralelo e sequencial do BioCrowds. As linhas em destaque na grade indicam a área analisada e as setas indicam o sentido de deslocamento dos grupos de agentes (Bicho, 2009).

- variação média angular dos agentes, denominada $\bar{\gamma}$ $(grau)$;
- densidade populacional global referente à toda a região no cenário, denominadas de $\bar{dp_g}$ (ags/m^2).

Além disto, utilizou-se o deslocamento máximo desejado por segundo s_{max} de todos os agentes na simulação igual à 1,2 m e uma taxa de quadros por segundos igual a 30.

Em sua tese de doutorado, Bicho (2009) faz uma avaliação do simulador sequencial alterando a densidade de marcadores no cenário e o raio do espaço pessoal do agente para um total de 400 agentes simulados (dois grupos de 200 agentes se deslocando em sentidos opostos) com sua representação infinitesimal, ou seja, as simulações realizadas são livres de colisão, como comentado no Capítulo 4. As Tabelas 5.1 e 5.2 apresentam os resultados das medidas utilizadas como critério de avaliação do simulador paralelo obtidos para densidades de 15 $marcadores/m^2$ e 60 $marcadores/m^2$, respectivamente, e para raio pessoal de cada agente igual a 0,6 m e 1,2 m. É importante destacar que, para cada experimento, foram realizadas 20 simulações variando a posições iniciais dos agentes e a distribuição dos marcadores do cenário, obtendo assim uma média e um desvio padrão dos valores das medidas analisadas.

Algoritmo	R (m)	$\bar{v}$ (m/s)	σ	$\bar{\gamma}$ (grau)	σ	$\bar{dp_g}$ (ags/m^2)	σ
Sequencial	1,2	1,2	0,01	10,18	0,56	1,04	0
Paralelo	1,2	1,15	0	14,6	0,22	1,12	0,06
Sequencial	0,6	1,2	0	9,21	0,14	1,05	0,01
Paralelo	0,6	1,15	0,02	14,15	0,17	1,45	0

Tabela 5.1: Média e desvio padrão dos valores de velocidade, da variação angular da orientação e da densidade populacional para o BioCrowds original e o algoritmo paralelo utilizando uma densidade de marcadores igual a 15 $marcadores/m^2$ e diferentes raios R para o espaço pessoal dos agentes. Os desvios padrão apresentados referem-se à dispersão das médias obtidas nos experimentos para uma mesma medida calculada.

Os resultados obtidos para o algoritmo paralelo devem ser comparados considerando que ao implementar o algoritmo para GPU, o mesmo precisou ser adaptado às características da

Algoritmo	R (m)	$\bar{v}$ (m/s)	σ	$\bar{\gamma}$ (degree)	σ	$\bar{dp_g}$ (ags/m^2)	σ
Sequencial	1,2	1,2	0	7,96	0,19	1,05	0
Paralelo	1,2	1,15	0,01	12,23	0,2	1,13	0
Sequencial	0,6	1,2	0	5,41	0,23	1,07	0,01
Paralelo	0,6	1,15	0	10,12	0,13	1,22	0

Tabela 5.2: Média e desvio padrão dos valores de velocidade, da variação angular da orientação e da densidade populacional para o BioCrowds original e o algoritmo paralelo utilizando uma densidade de marcadores igual a 60 $marcadores/m^2$ e diferentes raios R para o espaço pessoal dos agentes. Os desvios padrão apresentados referem-se à dispersão das médias obtidas nos experimentos para uma mesma medida calculada.

GPU, discutidas no Capítulo 3 e Seção 4.3.2, como por exemplo a minimização da utilização de operações para controle de fluxo. Por este motivo, do ponto de vista da precisão dos resultados, o algoritmo paralelo apresentou valores aproximados aos encontrados no algoritmo sequencial. Como discutido na tese de Bicho (2009), a utilização de um raio pessoal do agente R igual a 1,2 m no algoritmo paralelo, ocasionou em ambas as densidades de marcadores uma maior variação angular da orientação ($\bar{\gamma}$), menor densidade populacional ($\bar{dp_g}$) e mesma velocidade média ($\bar{v}$). Este comportamento também é observado no BioCrowds paralelo, portanto, conclue-se, que os resultados do BioCrowds paralelo são equivaletes aos do algoritmo sequencial.

A taxa de quadros por segundo em cada simulação realizada também foi obtida, sendo a médias dos valores obtidos apresentados na Tabela 5.3. Para a obtenção da taxa de quadros por segundo do simulador BioCrowds sequencial, as simulações foram realizadas em um processador Intel®Core™2 Duo 2.13 GHz e 4 GB DDR2 em 800 MHz.

Densidade ($marcadores/m^2$)	R (m)	BioCrowds Paralelo (fps)	σ (fps)	BioCrowds (Bicho, 2009) (fps)
15	1,2	82,09	3,38	49,63
15	0,6	78,62	0,46	70,89
60	1,2	25,67	0,18	16,36
60	0,6	26,21	0,36	27,18

Tabela 5.3: Taxa de quadros por segundo obtida simulando multidões utilizando o algoritmo BioCrowds paralelo em uma placa de video Nvidia GTS 250 e o algoritmo BioCrowds sequencial em um processador Intel®Core™2 Duo 2.13 GHz e 4 GB DDR2 em 800 MHz. O desvio padrão representado por σ refere-se à dispersão das médias das taxas obtidas nos experientos para uma mesma medida calculada.

Observa-se que a taxa de quadros por segundo de ambos os simuladores é próxima para um raio pessoal do agente igual à 0,6 m. Com o aumento na densidade de marcadores ocorre um aumento no tempo de processamento e assim uma diminuição na taxa de quadros por segundo, tanto no algoritmo BioCrowds sequencial como também no algoritmo paralelo. Este aumento no tempo de processamento do algoritmo paralelo ocorre devido ao aumento nos cálculos referentes ao primeiro *kernel*. Mesmo com este aumento, a taxa de quadros por segundo obtida com o

algoritmo paralelo manteve-se superior à taxa obtida com o BioCrowds sequencial, como era esperado.

Com o mesmo cenário, realizou-se outro experimento utilizando uma densidade de marcadores igual a 60 $marcadores/m^2$ e raio pessoal R igual a 1,2 m com diferentes quantidades de agentes e representação finita. Nos dois primeiros experimentos, um grupo de 25 ou 50 agentes move-se de uma extremidade para outra no corredor. Nos demais experimentos, dois grupos de 25, 50, 100, 200 ou 400 agentes movem-se na mesma direção, mas em sentidos opostos. Cada experimento foi repetido vinte vezes para diferentes posições iniciais dos agentes e diferentes distribuições de marcadores no cenário, utilizando o deslocamento máximo por quadro $s_{max} = 1,2$ m e o resultado apresentado é a média dos valores obtidos para cada grupo simulado.

Algoritmo	Num. Agentes (num. grupos)	25 (1)	50 (1)	50 (2)	100 (2)	200 (2)	400 (2)	800 (2)
Sequencial	$\vec{v}$ (m/s)	1,19	1,19	1,17	1,16	1,14	1,11	1,09
	σ (m/s)	0,0006	0,0006	0,0021	0,0045	0,0096	0,0206	0,0319
Paralelo	$\vec{v}$ (m/s)	1,17	1,17	1,17	1,17	1,16	1,14	1,05
	σ (m/s)	0,018	0,02	0,015	0,01	0,02	0,012	0,01

Tabela 5.4: Média e desvio padrão dos valores das velocidades do agentes utilizando uma densidade de marcadores igual a 60 $marcadores/m^2$ e raio $R = 1,2$ m para o espaço pessoal dos agentes.

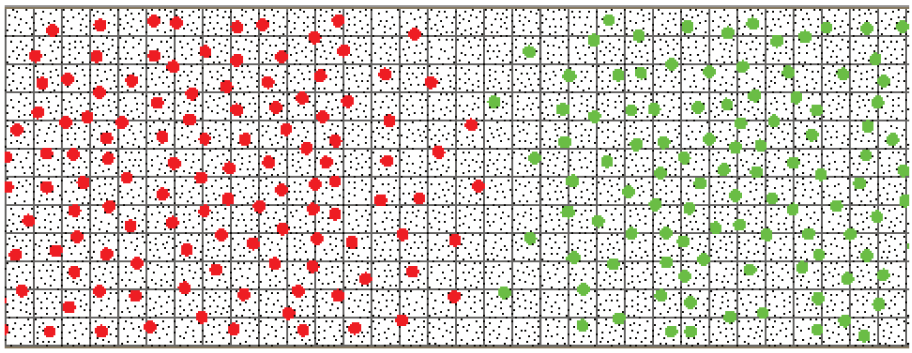
Com o aumento no número de agentes no cenário, como esperado, ocorreu em ambos os algoritmos uma diminuição de sua velocidade devido ao aumento na densidade de agentes em determinadas regiões do cenário, como no centro do corredor, local onde ocorre o cruzamento entre os grupos simulados. Em uma multidão real, quando ocorre o cruzamento de grupos de agentes em sentidos opostos, o fenômeno de formação de vias de pedestres emerge, minimizando o esforço realizado pelos agentes para alcançar seus respectivos objetivos. Este comportamento pode ser observado na Figura 5.2.

5.2 Simulação do efeito de pressão entre agentes

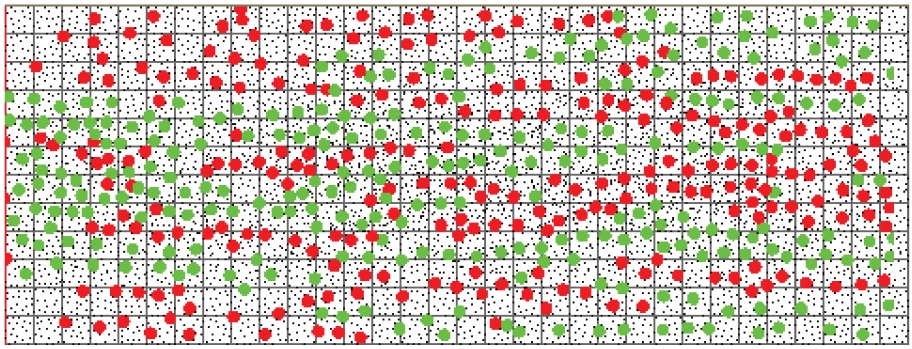
Nesta seção são apresentados os resultados das simulações realizadas para analisar a inclusão do efeito de pressão sobre os agentes no algoritmo BioCrowds paralelo. Para isto, os valores da velocidade média $\bar{v}$ (m/s) e a variação média angular $\bar{\gamma}$ $(grau)$ dos agentes serão comparados com os valores obtidos com as simulações realizadas por Pinheiro (2010).

As simulações utilizam como cenário um corredor de 50 m de comprimento por 12 m de largura com uma porta em seu fim de 1 m de largura, como mostra a Figura 5.3, onde foram simulados um total de 400 agentes com representação finita se deslocando em uma única direção e sentido.

Utilizou-se o deslocamento máximo do agente por segundo, s_{max}, igual à 1,2 m, o raio pessoal R igual a 1,2 m e uma taxa de quadros por segundos igual a 30. A Tabela 5.5 mostra os resultados para três simulações:



(a)



(b)

Figura 5.2: Nesta sequência de quadros observa-se a reprodução pelo BioCrowds paralelo de vias de pedestres.

1. densidade de marcadores constante em todo o cenário durante toda a simulação (BioCrowds proposto por Bicho (2009));

2. densidade de marcadores variando durante a simulação, em locais pré-estabelecidos, conforme a necessidade (*pushing* proposto por Pinheiro (2010));

3. densidade de marcadores variando durante a simulação, em locais onde haja necessidade, conforme proposta do presente trabalho (algoritmo BioCrowds paralelo).

Simulação	$\bar{v}$ (m/s)	σ m/s	$\bar{\gamma}$ $(grau)$	σ $(grau)$
1	0,54	0,02	13,74	0,23
2	0,59	0,01	15,49	0,13
3	0,99	0,01	22,88	0,81

Tabela 5.5: Média dos valores das velocidade e variação angular da orientação com seus respectivos desvio padrão obtidos ao simular o comportamento de *pushing* no algoritmo BioCrowds paralelo. Os resultados mostrados para 1 e 2 foram retirados do trabalho de Pinheiro (Pinheiro, 2010).

O efeito de pressão foi incluído no algoritmo através da mudança dinâmica da densidade de marcadores em regiões determinadas no cenário como já definido na Seção 4.2. Portanto, em

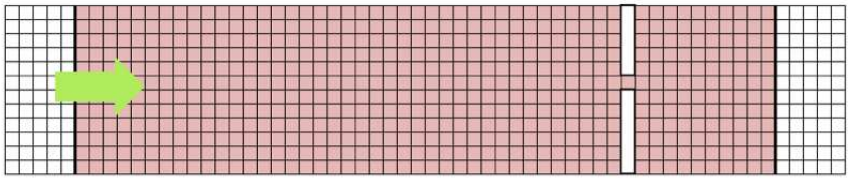


Figura 5.3: Cenário utilizado para análise do comportamento *pushing* no algoritmo BioCrowds paralelo. As áreas em vermelho na grade indicam a área pela qual o agente pode se deslocar, enquanto a área branca define uma região onde não há marcadores, portanto não poderam ultrapassar, como uma parede. A seta indica o sentido de deslocamento dos mesmos.

regiões onde há agentes estagnados devido a obstáculos locais como paredes, o próprio algoritmo aumenta a densidade de marcadores neste local. Isto implica em uma quantidade maior de marcadores a serem disputados pelos agentes e uma maior facilidade para estes alterarem sua trajetória em direção ao objetivo, aumentando a velocidade média dos mesmos (coluna $\bar{v}$ da Tabela 5.5). Por este motivo o valor para a média da variação angular da orientação dos agentes também foi maior do que em relação ao algoritmo sem o tratamento de *pushing* (coluna $\bar{\gamma}$ da Tabela 5.5). A taxa de quadros por segundo obtida com a inclusão do efeito de pressão nos agentes foi de 75 e em média, 3,71 agentes não alcançaram o seu destino nas simulações. A Figura 5.4 ilustra uma sequência de quadros de uma animação de uma multidão com o algoritmo BioCrowds paralelo com a inclusão do efeito de pressão nos agentes.

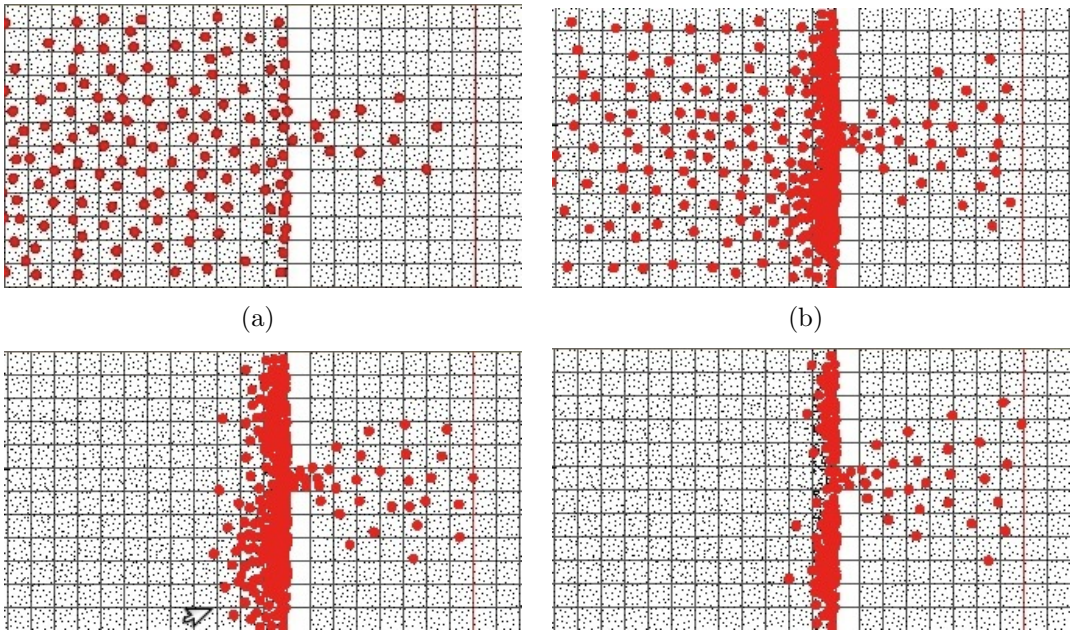


Figura 5.4: Sequência de quadros da simulação de uma multidão no algoritmo BioCrowds com a inclusão do comportamento de *pushing*.

5.3 Impacto do número de agentes e densidade de marcadores na taxa de quadros por segundo das simulações

No algoritmo BioCrowds sequencial, o custo computacional é dependente da quantidade de marcadores e de agentes no cenário simulado. Em sua tese, Bicho (2009) faz uma análise do algoritmo variando estes dois parâmetros e verificou que o aumento da densidade de marcadores no cenário é o fator que mais contribui para o aumento do custo computacional e, desta forma, para a diminuição na taxa de quadros por segundo das simulações.

Para verificar o custo computacional no algoritmo proposto nesta dissertação, utilizou-se um cenário com uma área quadrada, cuja dimensão é $80m$ x $80m$, para três densidades de marcadores diferentes: 15, 20 e 60 $marcadores/m^2$. Os resultados obtidos são mostrados na Figura 5.3 juntamente com os valores aproximados do algoritmo BioCrowds original, conforme Figura 5.5 da tese de Bicho (2009) [2].

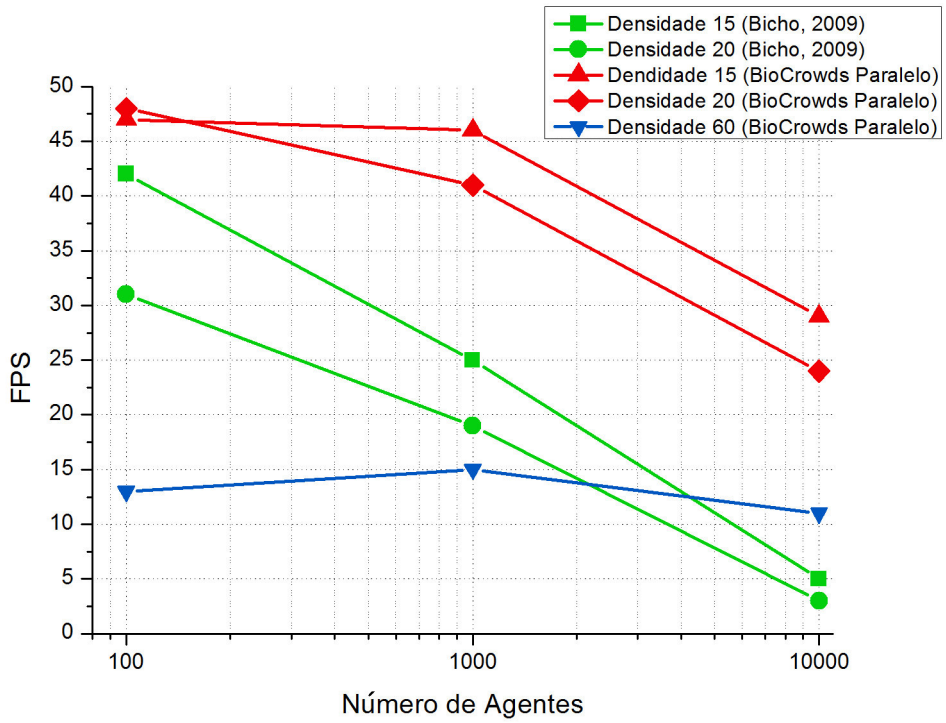


Figura 5.5: Taxa de quadros por segundo como função da quantidade de agentes, avaliada para três densidades de marcadores. Os resultados apresentados do BioCrowds sequencial foram obtidos da tese de Bicho (2009) e os resultados referentes ao BioCrowds paralelo foram simulados em uma placa de vídeo NVIDIA GTS 250.

No algoritmo paralelo, como era esperado, obteve-se um significativo aumento na taxa de quadros por segundo das simulações realizadas em relação ao algoritmo sequencial, principalmente quando o número de agentes na simulação é elevado. Entretanto, assim como no algoritmo

[2] As taxas de quadros por segundo do algoritmo BioCrowds sequencial foram obtidas da tese Bicho (2009).

sequencial, o aumento na densidade de marcadores utilizada é o principal fator que contribui para a diminuição consideravel da taxa de quadros por segundo da simulação e quando este valor é muito elevado, como foi no caso de 60 $marcadores/m^2$, a quantidade de agentes simulados interferiu muito pouco na taxa de quadros por segundo da simulação.

5.4 Considerações Finais

Neste capítulo foram apresentados resultados das simulações obtidas com o uso do algoritmo BioCrowds paralelo proposto neste trabalho, destacando-se o aumento na taxa de quadros por segundo obtido nas simulações realizadas em comparação aos resultados apresentados por Bicho (2009) e também uma simulação mais realista do efeito de pressão em comparação ao trabalho de Pinheiro (2010). Em relação às características avaliadas na multidão como velocidade média dos agentes e variação angular média da orientação dos mesmos, o algoritmo proposto apresentou valores aproximados, o que não influenciou na visualização da multidão simulada. Juntamente com esta dissertação, encontra-se um CD com o código fonte do simulador desenvolvido, além de vídeos das simulações realizadas neste trabalho.

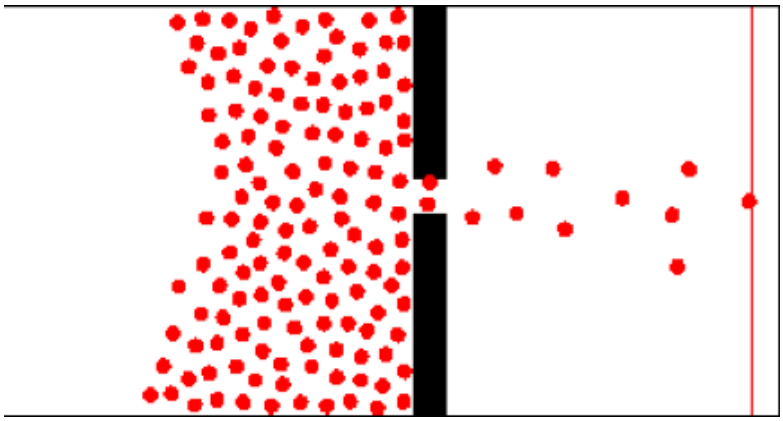
Capítulo 6

Conclusão

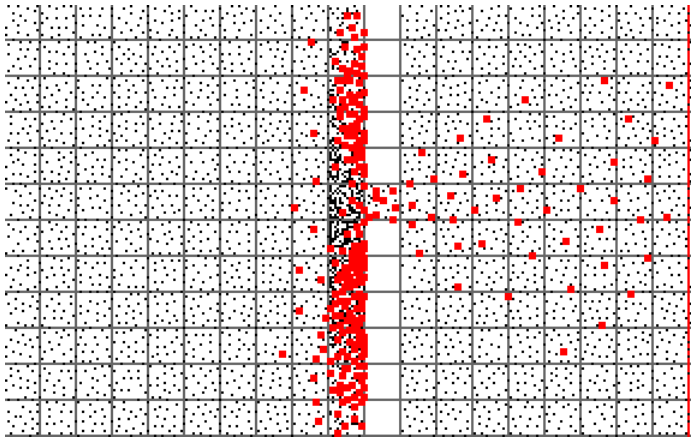
O trabalho proposto nesta dissertação prevê duas melhorias para o simulador BioCrowds que são a inclusão do efeito decorrente do aumento na densidade de indivíduos em determinadas regiões do cenário simulado, denominado de efeito de pressão ou *pushing*, e a paralelização do algoritmo BioCrowds para processamento utilizando uma unidade de processamento gráfico (GPU). A motivação para o desenvolvimento deste trabalho surgiu ao simular ambientes com um grande número de indivíduos e marcadores, pois o simulador BioCrowds apresentava uma diminuição na taxa de quadros por segundo, como mostrado na Figura 5.5. Além disto, o algoritmo original não tratava situações nas quais ocorria contato entre os agentes, portanto, não apresentava o comportamento *pushing*.

Para a inclusão do efeito *pushing* no BioCrowds, analisou-se o trabalho de Pinheiro (2010), o qual propôs uma solução para o tratamento deste comportamento de forma pré-programada em determinadas regiões do cenário, como discutido no Capítulo 4, e também a tese de Bicho (2009), na qual ele comenta que o efeito de pressão pode ser incluído através da alteração da Equação 4.4 de ponderação dos marcadores no cenário. Buscou-se então uma solução que não exigisse intervenção do usuário, e permitisse simular o efeito de pressão de forma mais natural. Assim, este trabalho baseou-se nesta idéia inicial e apresentou uma solução na qual o comportamento de *pushing* emerge durante a simulação em decorrência da proximidade entre os agentes simulados, sem qualquer pré-programação ou intervenção do usuário. É importante destacar que a avaliação deste comportamento foi realizada apenas em situações em que há o estreitamento do cenário simulado.

A Figura 6.1 mostra um quadro da simulação de um cenário com um estreitamento em seu fim, sendo realizado no simulador desenvolvido por Bicho (2009) em sua tese de doutorado sem o efeito de pressão (a), e outro quadro do mesmo cenário sendo realizado no simulador desenvolvido neste trabalho, com a inclusão do *pushing* (b). Visualmente percebe-se que no simulador desenvolvido neste trabalho, a inclusão do efeito de *pushing* ocasionou a diminuição da distância entre os agentes simulados e um aumento na vazão dos agentes após o estreitamento, gerando resultados mais realistas. Para uma melhor avaliação da inclusão do *pushing*, analisou-se o simulador quantitativamente, comparando a velocidade média dos agentes simulados e a variação média de sua orientação em ambos os simuladores. Os resultados foram apresentados na Tabela 5.5.



(a) Simulação realizada utilizando o simulador BioCrowds desenvolvido por Bicho (Pinheiro, 2010).



(b) Simulação realizada com o BioCrowds paralelo.

Figura 6.1: Diferença entre os simuladores BioCrowds original e o simulador desenvolvido neste trabalho. Em (a) não ocorre a simulação do efeito *pushing*, enquanto que em (b) o efeito é simulado.

Para a paralelização do modelo, fez-se inicialmente um estudo da arquitetura das placas de vídeo NVIDIA, analisando sua organização e as principais características da placa de vídeo utilizada neste trabalho (NVIDIA GTS 250). Em seguida, foram analisadas as principais plataformas para processamento genérico em GPU, que são CUDA e OpenCL. Para o desenvolvimento do simulador BioCrowds paralelo utilizou-se a plataforma OpenCL, proposta pelo consórcio Khronos Group e não a plataforma CUDA, por razão de uma maior portabilidade futura, pois a plataforma OpenCL permite que um mesmo programa possa ser processado por diferentes *hardware*, como GPUs de diferentes fabricantes e CPUs, desde que estes tenham suporte ao ambiente.

Com relação aos resultados apresentados no Capítulo 5, percebeu-se que a utilização da GPU permitiu atingir taxas de processamento muito superiores às taxas obtidas com o BioCrowds sequencial, como era esperado. Em seu trabalho, Bicho (2009) concluiu que o aumento na densidade de marcadores no cenário simulado tem maior impacto no custo computacional quando o número de agentes simulados não é significativo, caso contrário, a tendência no modelo sequencial é que a densidade de marcadores não seja fator preponderante. Além disto, Bicho concluiu

que a partir de uma certa densidade de marcadores, não há mudança significativa na variação angular média da orientação do agente. No modelo paralelo apresentado nesta dissertação, o aumento na densidade de marcadores é o principal fator para o aumento no custo computacional, como mostra a Figura 5.5. Isto se deve à organização dos *kernels* desenvolvidos neste trabalho, os quais realizam cálculos tanto para marcadores como para agentes, como discutido no Capítulo 4. Portanto, uma densidade entre 15 e 60 $marcadores/m^2$ no algoritmo paralelo é suficiente para a obtenção de trajetórias suaves para os agentes simulados, mantendo a taxa de quadros por segundo superior à taxa do algoritmo sequencial.

É importante destacar que a forma como os *kernels* apresentados neste trabalho foram organizados não é a única solução para a paralelização do algoritmo BioCrowds. Outras estratégias podem ser adotadas para a sua implementação em GPU, alterando, assim, a relação do custo computacional à densidade dos marcadores do cenário.

6.1 Trabalhos futuros

Para sequência deste trabalho, vislumbram-se diversos cenários. Pode-se adaptar o algoritmo paralelo desenvolvido para processamento em mais de uma GPU, permitindo assim um aumento na capacidade de simulação de cenários maiores e mais complexos. Pode-se, também, avaliar outras estratégias para a modelagem paralela do algoritmo BioCrowds, apresentando uma outra estrutura para os *kernels* do simulador. Por fim, pode-se aplicar o modelo BioCrowds para simulação de multidão em situações específicas de emergência, como evacuação de áreas de risco, onde não se pode realizar testes com uma multidão real por trazer riscos físicos ou mesmo por ser inviável.

Bibliografia

Berg, J. V. D., Patil, S., Sewall, J., Manocha, D., and Lin, M. (2008). Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, I3D '08, pages 139–147, New York, NY, USA. ACM.

Bicho, A. L. (2009). *Da modelagem de plantas à dinâmica de multidões: um modelo de animação comportamental bio-inspirado*. PhD thesis, Universidade Estadual de Campinas.

Bicho, A. L., Rodrigues, R. A., Musse, S. R., Jung, C. R., Paravisi, M., and Magalhães, L. P. (2012). Simulating crowds based on a space colonization algorithm. *Computers & Graphics; Graphics*, 36(2):70 – 79. Virtual Reality in Brazil 2011.

Bon, G. L. (1905). *La psychologie des foules (1895)*, volume 9. Édition Félix Alcan, Paris, França.

Bouvier, E., Cohen, E., and Najman, L. (1997). From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation. *J. Electronic Imaging*, 6(1):94–107.

Burstedde, C., Klauck, K., Schadschneider, A., and Zittartz, J. (2001). Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295:507 – 525.

Chen, D., Wang, L., Wu, X., Chen, J., Khan, S. U., Kołodziej, J., Tian, M., Huang, F., and Liu, W. (2012). Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture. *Future Generation Computer Systems*.

Cook, R. L. (1986). Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72.

Cutting, J., Vishton, P., and Braren, P. (1995). How we avoid collisions with stationary and with moving obstacles. *Psychological Review*, 102:627–651.

Goldenstein, S., Karavelas, M., Metaxas, D., Guibas, L., Aaron, E., and Goswami, A. (2001). Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, 25:983–998.

Goldenstein, S., Large, E., and Metaxas, D. (1999). Non-linear dynamical system approach to behavior modeling. *The Visual Computer*, 15:349–364.

Helbing, D., Farkas, I., and Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, 207(6803):487–490.

Helbing, D. and Molnar, P. (1997). Self-organization phenomena in pedestrian crowds. In *Self-organization of Complex Structures: From Individual to Collective Dynamics*, pages 569–577. Gordon and Breach, London, UK.

Henderson, L. F. (1971). The statistics of crowd fluids. *Nature*, 229(5284):381–383.

Hughes, R. L. (2003). The flow of human crowds. *Annual Review of Fluid Mechanics*, 35(1):169–182.

Jiang, H., Xu, W., Mao, T., Li, C., Xia, S., and Wang, Z. (2010). Continuum crowd simulation in complex environments. *Computers & Graphics*, 34(5):537 – 544.

Karamouzas, I., Geraerts, R., and Overmars, M. (2009). Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 113–120, New York, NY, USA. ACM.

Kim, S., Guy, S. J., Manocha, D., and Lin, M. C. (2012). Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 55–62, New York, NY, USA. ACM.

Kirk, D. B. and Hwu, W.-m. (2010). *Programming Massively Parallel Processors: a Hands-on Approach*. Elsevier Inc.

Lagae, A. (2009). *Wang Tiles in Computer Graphics*. Morgan & Claypool Publishers.

Musse, S. R. (2000). *Human Crowd Modelling with various levels of behaviour Control*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Suíça.

Musse, S. R. and Thalmann, D. (2001). Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164.

NVIDIA (2010). Opencl best practices guide. URL: http://www.nvidia.com/content/cudazone. NVIDIA. (acessado em: 25/03/2013).

NVIDIA (2012). Cuda c programming guide. URL: http://docs.nvidia.com/cuda. NVIDIA. (acessado em: 25/03/2013).

Ondřej, J., Pettré, J., Olivier, A.-H., and Donikian, S. (2010). A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph.*, 29(4):123:1–123:9.

O'Rourke, J. (1998). *Computational Geometry in C*. Cambridge University Press, second edition edition.

Passos, E. B., Joselli, M., Zamith, M., Clua, E. W. G., Montenegro, A., Conci, A., and Feijo, B. (2010). A bidimensional data structure and spatial optimization for supermassive crowd simulation on gpu. *Comput. Entertain.*, 7(4):60:1–60:15.

Pelechano, N., Allbeck, J. M., and Badler, N. I. (2007). Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 99–108, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Pinheiro, I. C. (2010). Simulação de multidões: um modelo bio-inspirado. *XVIII Congresso interno de Iniciação Científica da UNICAMP*, page 352.

Qiu, F. and Hu, X. (2010). Modeling group structures in pedestrian crowd simulation. *Simulation Modelling Practice and Theory*, 18(2):190 – 205.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34.

Runions, A., Fuhrer, M., Lane, B., Federl, P., Rolland-Lagan, A.-G., and Prusinkiewicz, P. (2005). Modeling and visualization of leaf venation patterns. *ACM Trans. Graph.*, 24(3):702–711.

Selye, H. (1956). *The Stress of Life*. McGraw-Hill.

Shopf, J., Barczak, J., Oat, C., and Tatarchuk, N. (2008). March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on gpu. In *ACM SIGGRAPH 2008 Games*, SIGGRAPH '08, pages 52–101, New York, NY, USA. ACM.

Song, Y., Gong, J., Li, Y., Cui, T., Fang, L., and Cao, W. (2013). Crowd evacuation simulation for bioterrorism in micro-spatial environments based on virtual geographic environments. *Safety Science*, 53(0):105 – 113.

Still, G. K. (2000). *Crowd dynamics*. PhD thesis, University of Warwick, Coventry, UK.

Thalmann, D. and Musse, S. R. (2012). *Crowd Simulation*. Segunda Edição. Springer-Verlag, London, UK.

Treuille, A., Cooper, S., and Popović, Z. (2006). Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168.

Tsuchiyama, R., Nakamura, T., Iizuca, T., Asahara, A., and Miki, S. (2009). *The OpenCL Programming Book: Parallel Programming for MultiCore CPU and GPU*. Fixstar Corporation.

Tu, X. and Terzopoulos, D. (1994). Artificial fishes: physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 43–50, New York, NY, USA. ACM.

Ulicny, B., Ciechomski, P. d. H., Musse, S. R., and Thalmann, D. (2006). Course on populating virtual environments with crowds. *Viena, Áustria: Eurographics, 2006*, page 95.

Xiong, M., Lees, M., Cai, W., Zhou, S., and Low, M. Y. H. (2010). Hybrid modelling of crowd simulation. *Procedia Computer Science*, 1(1):57 – 65.

Zhang, L., Song, Y., Zhou, S., Gong, J., and Li, W. (2010). The framework of crowd simulation modeling with social network. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, volume 3, pages V3–437 –V3–441.

Anexo A

Compute Capability (CC) das placas de vídeo NVidia

O termo *Compute Capability* é utilizado pela NVIDIA para designar diferentes arquiteturas para suas placas de vídeo. Novas arquiteturas usualmente processam mais rapidamente tanto programas desenvolvidos com CUDA quanto processamento gráfico. Na figura a seguir as principais características de cada geração de placas de vídeo da NVIDIA são apresentadas NVIDIA (2012).

<table>
<tr><td></td><td colspan="5">Compute Capability</td></tr>
<tr><td>Feature Support
(Unlisted features are supported for all compute capabilities)</td><td>1.0</td><td>1.1</td><td>1.2</td><td>1.3</td><td>2.x</td></tr>
<tr><td>Atomic functions operating on 32-bit integer values in global memory (Section B.11)</td><td rowspan="2">No</td><td colspan="4" rowspan="2">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in global memory (Section B.11.1.3)</td></tr>
<tr><td>Atomic functions operating on 32-bit integer values in shared memory (Section B.11)</td><td colspan="2" rowspan="4">No</td><td colspan="3" rowspan="4">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in shared memory (Section B.11.1.3)</td></tr>
<tr><td>Atomic functions operating on 64-bit integer values in global memory (Section B.11)</td></tr>
<tr><td>Warp vote functions (Section B.12)</td></tr>
<tr><td>Double-precision floating-point numbers</td><td colspan="3">No</td><td colspan="2">Yes</td></tr>
<tr><td>Atomic functions operating on 64-bit integer values in shared memory (Section B.11)</td><td colspan="4" rowspan="7">No</td><td rowspan="7">Yes</td></tr>
<tr><td>Atomic addition operating on 32-bit floating point values in global and shared memory (Section B.11.1.1)</td></tr>
<tr><td>__ballot() (Section B.12)</td></tr>
<tr><td>__threadfence_system() (Section B.5)</td></tr>
<tr><td>__syncthreads_count(), __syncthreads_and(), __syncthreads_or() (Section B.6)</td></tr>
<tr><td>Surface functions (Section B.9)</td></tr>
<tr><td>3D grid of thread blocks</td></tr>
</table>

	Compute Capability				
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**
Maximum dimensionality of grid of thread blocks	2				3
Maximum x-, y-, or z-dimension of a grid of thread blocks	65535				
Maximum dimensionality of thread block	3				
Maximum x- or y-dimension of a block	512				1024
Maximum z-dimension of a block	64				
Maximum number of threads per block	512				1024
Warp size	32				
Maximum number of resident blocks per multiprocessor	8				
Maximum number of resident warps per multiprocessor	24		32		48
Maximum number of resident threads per multiprocessor	768		1024		1536
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K
Maximum amount of shared memory per multiprocessor	16 KB				48 KB
Number of shared memory banks	16				32
Amount of local memory per thread	16 KB				512 KB
Constant memory size	64 KB				
Cache working set per multiprocessor for constant memory	8 KB				
Cache working set per multiprocessor for texture memory	Device dependent, between 6 KB and 8 KB				
Maximum width for a 1D texture reference bound to a CUDA array	8192				32768
Maximum width for a 1D texture reference bound to linear memory	2^{27}				
Maximum width and number of layers for a 1D layered texture reference	8192 x 512				16384 x 2048
Maximum width and height for a 2D texture reference bound to linear memory or to a CUDA array	65536 x 32768				65536 x 65535
Maximum width, height, and number of layers for a 2D layered texture reference	8192 x 8192 x 512				16384 x 16384 x 2048
Maximum width, height, and depth for a 3D texture reference bound to a CUDA array	2048 x 2048 x 2048				
Maximum number of textures that can be bound to a kernel	128				
Maximum width for a 1D surface reference bound to a CUDA array	N/A				8192
Maximum width and height for a 2D surface reference bound to a CUDA array					8192 x 8192
Maximum number of surfaces that can be bound to a kernel					8
Maximum number of instructions per kernel	2 million				

Anexo B

Cálculo para definição do tamanho do bloco de *threads*

Cada placa de vídeo da NVIDIA possui características específicas referentes a sua arquitetura, como número de *streaming multiprocessors*, número de *streaming processors*, número de *threads* e número de blocos que podem ser executados simultaneamente. As características referentes à placa NVIDIA GTS 250, utilizada neste trabalho, podem ser encontradas no Anexo A. Nele, encontram-se as seguintes informações sobre a placa:

- máximo número de *threads* permitidas simultaneamente em um SM (T_{SM}), igual a 768;
- máximo número de *warps* permitidos simultaneamente em um SM (W_{SM}), igual a 24;
- máximo número de blocos permitidos simultaneamente em um SM (B_{SM}), igual a 8.

Com estes valores, pode-se calcular o tamanho ideal do bloco a ser utilizado para processamento com a placa de vídeo escolhida. Primeiramente, supõe-se que o tamanho do bloco (B) a ser utilizado na aplicação seja de 512 *threads* (máximo permitido para a placa de vídeo utilizada). Acha-se o número de blocos por SM necessários para a aplicação (B'_{SM}) dividindo T_{SM} por B, como mostra a Equação A.1:

$$B'_{SM} = T_{SM} \div B \qquad \text{A.1}$$

O valor obtido para B'_{SM} é 1,5 que é aproximado para 2 blocos necessários.

Em seguida, acha-se o número de *warps* em um bloco (W'_B) para o tamanho de bloco escolhido, dividindo B pelo número de *threads* que compõem um *warp*, que é igual a 32. Este passo é definido na Equação A.2. Obtém-se $W'_B = 16$.

$$W'_B = B \div 32 \qquad \text{A.2}$$

Com os valores de B'_{SM} e W'_B, calcula-se o número possível de *warps* que executarão simultaneamente em cada SM (W'_{SM}), como mostra a Equação A.3:

$$W'_{SM} = B'_{SM} * W'_B \qquad \text{A.3}$$

O valor obtido para W'_{SM} neste exemplo é de 32, excedendo o valor máximo permitido de *warps* residenstes em um SM para a placa de vídeo utilizada neste trabalho. Assim, os mesmos cálculos foram realizados para os seguintes tamanhos de blocos (B): 256, 128, 64 e 32.

Para os valores de B iguais a 64 e 32, os valores para W'_B obtidos foram 12 e 24, respectivamente. Ambos excedem o valor máximo de blocos permitidos simultaneamente em um SM (B_{SM}). Portanto, caso estes valores sejam utilizados, apenas 8 blocos serão executados, subutilizando a placa de vídeo escolhida.

Para os valores de B iguais a 256 e 128, o valores obtidos para B'_{SM} foram 3 e 6, respectivamente, e o valor obtido para W'_{SM} para ambos foi de 24 *warps*, satisfazendo as características da placa utilizada. Escolheu-se utilizar o bloco com tamanho igual a 128, pois com este valor, garante-se a utilização de um número grande de blocos ($B'_{SM} = 6$), diminuindo o número de *warps* no bloco para o valor máximo permitido ($W_{SM} = 24$) e assim diminuindo o efeito da divergência de execução entre as *threads* de um *warp*.

Anexo C

Conteúdo do CD em anexo

Juntamente com o exemplar desta dissertação, encontra-se um CD com o seguinte conteúdo:

- Código fonte do simulador BioCrowds paralelo desenvolvido para GPU e CPU;
- Instruções para execução do simulador BioCrowds;
- Vídeos das simulações realizadas neste trabalho.