

Universidade Federal de Santa Catarina  
Programa de Pós-Graduação em  
Ciência da Computação

# **Tractografia em Tempo Real Através de Unidades de Processamento Gráfico**

**Adiel Mittmann**

Dissertação submetida à Universidade Federal de Santa Catarina  
como parte dos requisitos para a obtenção do título de Mestre em  
Ciência da Computação.

Orientador  
Dr. rer. nat. Aldo von Wangenheim

Florianópolis, novembro de 2009

L<sup>A</sup>T<sub>E</sub>X, R e Ipe foram utilizados na elaboração desta dissertação.

Catálogo na fonte pela Biblioteca Universitária da  
Universidade Federal de Santa Catarina,  
referente à impressão em A5

M685t Mittmann, Adiel

Tractografia em tempo real através de unidades de processamento gráfico [dissertação] / Adiel Mittmann; orientador, Aldo von Wangenheim. – Florianópolis, SC : 2009.

104 f.: il., graf., tabs.

Dissertação (mestrado) – Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da computação. 2. Sistemas em tempo real. 3. Tractografia. 4. Unidades de processamento gráfico (GPUs). 5. Ressonância magnética de tensores de difusão (DT-MRI, DTI). I. Wangenheim, Aldo von. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

# Tractografia em Tempo Real Através de Unidades de Processamento Gráfico

**Adiel Mittmann**

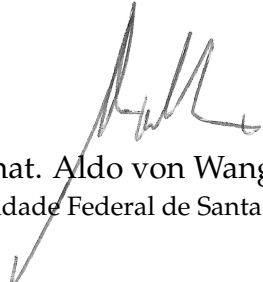
Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração de Sistemas de Conhecimento, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

**Coordenador**




Dr. Mauro Roisenberg  
Universidade Federal de Santa Catarina

**Orientador**



Dr. rer. nat. Aldo von Wangenheim  
Universidade Federal de Santa Catarina

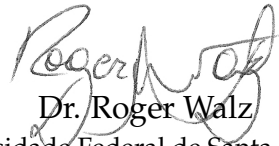
**Banca examinadora**



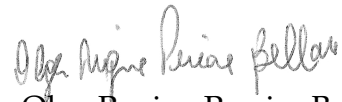
Dr. rer. nat. Eros Comunello  
Universidade Federal de Santa Catarina



Dr. Antonio Carlos dos Santos  
Faculdade de Medicina de Ribeirão Preto



Dr. Roger Walz  
Universidade Federal de Santa Catarina



Dra. Olga Regina Pereira Bellon  
Universidade Federal do Paraná

# Resumo

A tractografia por propagação de linhas é um método que, com base na ressonância magnética de tensores de difusão (DT-MRI), revela in vivo a disposição dos tratos neurais no cérebro humano. O alto custo computacional da tractografia, porém, implica um tempo de espera significativamente longo para o cálculo e exibição dos resultados. Esse tempo pode ser reduzido executando-se a tractografia de forma paralela, já que as trajetórias encontradas pelo método de propagação de linhas são independentes umas das outras. Uma plataforma atrativa para a execução paralela de programas é oferecida pelas unidades de processamento gráfico (GPUs) das placas de vídeo atuais. Desse modo, este trabalho propõe a execução paralela da tractografia por propagação de linhas em GPUs através da tecnologia CUDA. Experimentos foram conduzidos para avaliar o desempenho da tractografia em um processador central (CPU) e quatro GPUs distintas. Os resultados mostram que as GPUs são, no melhor caso, até 38 vezes mais velozes que a CPU na execução da tractografia, e até 8 vezes mais rápidas no pior caso. A velocidade obtida através das GPUs permite que os resultados tractográficos sejam calculados e exibidos em tempo real para um número de trajetórias superior a 3.000.

**Palavras-chave:** unidades de processamento gráfico (GPUs), tractografia, ressonância magnética de tensores de difusão (DT-MRI, DTI), sistemas em tempo real.

# Abstract

Streamline fiber tracking is a technique based on diffusion tensor magnetic resonance imaging (DT-MRI) which reveals in vivo the position of fiber tracts in the human brain. Fiber tracking, however, is computationally expensive, typically requiring a long time to calculate and show its results. This delay may be reduced by executing fiber tracking in a parallel fashion, since the trajectories found by the streamline method are independent from each other. An attractive platform for the parallel execution of programs is offered by the graphics processing units (GPUs) of current video cards. Thus, this work proposes that streamline fiber tracking be executed on GPUs by means of the CUDA technology. Experiments were conducted in order to assess the performance of fiber tracking in one central processor (CPU) and four different GPUs. Results show that GPUs execute fiber tracking up to 38 times faster than the CPU in the best case and up to 8 times faster in the worst case. The speed achieved by GPUs allows the results of fiber tracking to be calculated and shown in real time for a number of seeds greater than 3.000.

**Keywords:** graphics processing units (GPUs), fiber tracking, tractography, diffusion tensor magnetic resonance imaging (DT-MRI, DTI), real-time systems.

Aos meus avôs,  
Pedro Francisco Mittmann e Cesar Domingos Filippini,  
in memoriam.

“Τὴν γὰρ παροιμίαν ὅτι ποτὲ λέγει, τὸ ‘χαλεπὰ τὰ καλὰ,’ δοκῶ μοι εἰδέναι.”  
“Aquele ditado de que ‘as coisas belas são difíceis,’ acho que eu entendo.”  
(Platão, Hípias Maior, 304e)

# Agradecimentos

À minha família, que me apóia incondicionalmente,  
ao Prof. Aldo von Wangenheim e ao Prof. Eros Comunello, pela estrutura propícia à pesquisa,  
ao sobretudo amigo Wilson Heck Junior, pelo hardware emprestado,  
ao colega Tiago H. C. Nobrega, com quem muitas idéias e códigos foram engendrados,  
sinceramente agradeço.



# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Contextualização . . . . .	1
1.2	Objetivo . . . . .	2
1.3	Método . . . . .	4
1.4	Trabalhos relacionados . . . . .	5
1.5	Convenções e notações . . . . .	6
<b>2</b>	<b>Tractografia</b>	<b>8</b>
2.1	Princípios físicos e matemáticos . . . . .	8
2.2	Ressonância magnética de tensores de difusão . . . . .	10
2.2.1	Equação de Stejskal–Tanner . . . . .	11
2.2.2	Cálculo do tensor . . . . .	13
2.2.3	Índices invariantes . . . . .	16
2.3	Tractografia . . . . .	18
2.3.1	Propagação de linhas . . . . .	18
2.3.2	Formalização . . . . .	19
2.3.3	Interpolação e regularização . . . . .	20
<b>3</b>	<b>Unidades de processamento gráfico</b>	<b>23</b>
3.1	As GPUs programáveis . . . . .	24
3.2	Programação de propósito geral em GPUs . . . . .	26
3.3	A tecnologia CUDA . . . . .	27
3.3.1	Ferramentas . . . . .	28
3.3.2	Programação com CUDA . . . . .	28
3.3.3	Modelo de threads . . . . .	30
3.3.4	Modelo de memória . . . . .	31
3.3.5	Arquitetura de hardware . . . . .	32
3.4	O futuro da programação em GPUs . . . . .	33
<b>4</b>	<b>Tractografia em GPUs</b>	<b>34</b>
4.1	Método de tractografia . . . . .	34

4.2	A biblioteca tractográfica . . . . .	35
4.3	Implementação de referência em CPU . . . . .	36
4.4	Implementação em GPU . . . . .	37
4.4.1	O paralelismo . . . . .	37
4.4.2	Estratégia de exploração do paralelismo . . . . .	38
4.4.3	Características da implementação . . . . .	41
4.5	Experimentos . . . . .	43
4.5.1	Planejamento . . . . .	43
4.5.2	Execução . . . . .	44
4.6	Resultados . . . . .	45
<b>5</b>	<b>Tractografia em tempo real</b>	<b>49</b>
5.1	A estação radiológica . . . . .	49
5.2	O módulo de tractografia da estação radiológica . . . . .	50
5.3	Interatividade do módulo de tractografia . . . . .	52
5.3.1	Paradigma de atualidade permanente . . . . .	52
5.3.2	Equalização dos parâmetros . . . . .	53
5.3.3	Estratégia de renderização . . . . .	54
5.4	Experimentos . . . . .	55
5.4.1	O procedimento automático . . . . .	56
5.4.2	Estrutura dos experimentos . . . . .	56
5.5	Resultados . . . . .	57
5.5.1	Similaridade dos resultados das CPUs . . . . .	57
5.5.2	Conjuntos de dados . . . . .	58
5.5.3	Tempo de execução . . . . .	59
5.5.4	Ganho de desempenho . . . . .	62
5.5.5	Taxa de quadros por segundo . . . . .	63
<b>6</b>	<b>Conclusão</b>	<b>67</b>
6.1	Contribuições . . . . .	67
6.2	Discussão . . . . .	68
6.3	Limitações e trabalhos futuros . . . . .	69
<b>A</b>	<b>Descrição de materiais</b>	<b>72</b>
	<b>Referências</b>	<b>80</b>

# Capítulo 1

## Introdução

### 1.1 Contextualização

A ressonância magnética de tensores de difusão (DT-MRI) é uma técnica através da qual se determina a intensidade e a orientação da difusão da água no corpo humano (Basser et al., 1994a; Mori e Zhang, 2006). Com essa técnica, é possível calcular para cada voxel do exame obtido um tensor de difusão, que é um modelo matemático para caracterizar a difusão. Para possibilitar o cálculo do tensor de difusão, um mínimo de sete volumes precisa ser adquirido — seis que descrevem a difusão em diferentes direções e um que serve de referência.

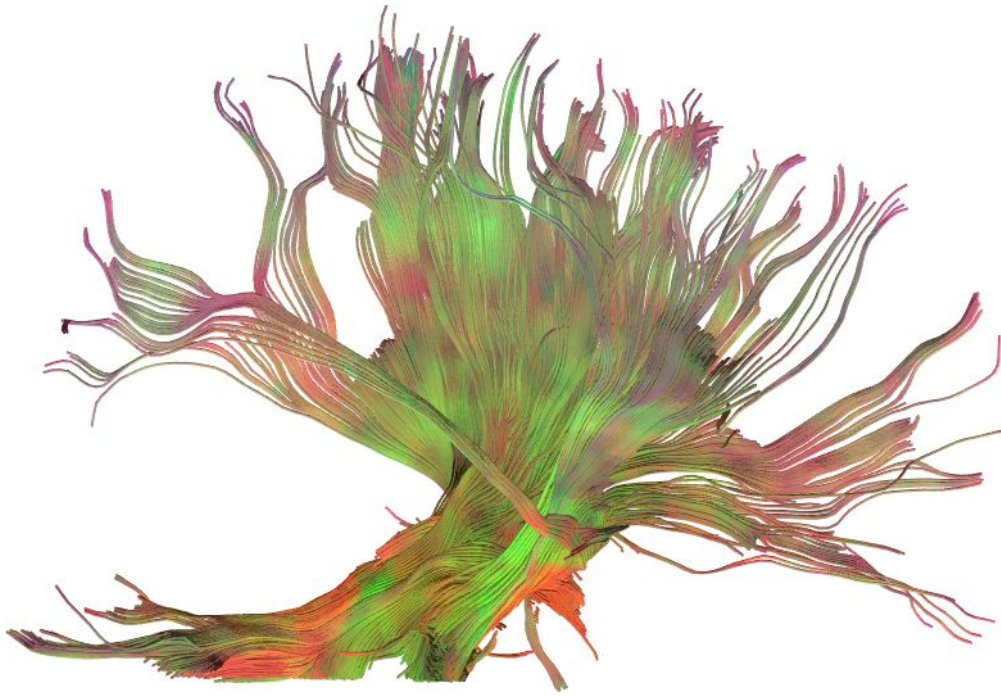
Partindo-se do princípio de que a água difunde-se na massa branca do cérebro com mais intensidade na direção das fibras (Conturo et al., 1999), é possível utilizar os tensores encontrados a partir de um exame de DT-MRI para determinar a localização das fibras nervosas no cérebro. Os métodos computacionais utilizados para identificar as fibras são em geral conhecidos como métodos de tractografia.

Um dos métodos de tractografia é a propagação de linhas (Mori, 2007). Nesse método, um número de pontos semente é distribuído em um volume de interesse. Em seguida, o método encontra um par de trajetórias para cada um dos pontos semente. O conjunto de trajetórias, que constitui o resultado tractográfico, é então exibido ao usuário. As Figuras 1.1–1.4 mostram exemplos de resultados tractográficos produzidos pelo método de propagação de linhas.

Os resultados produzidos pela tractografia encontram diversas aplicações, dentre as quais destaca-se seu uso no planejamento pré-cirúrgico e intra-cirúrgico de operações para a remoção de tumores ou outras lesões (Nimsky et al., 2005; Clark et al., 2003). O Capítulo 2 apresenta uma revisão detalhada sobre DT-MRI e tractografia.

A tractografia implica, contudo, um alto custo computacional (Jeong et al., 2007), que se traduz em um tempo de execução elevado. Diminuir o tempo que os resultados tractográficos levam para serem calculados significa aumentar a utilidade clínica da tractografia na prática (McGraw e Nadar, 2007).

Uma maneira de aumentar a velocidade da tractografia é construir implementações paralelas (Singh et al., 2006). É possível, por exemplo, calcular cada uma das trajetórias exibidas nas Figuras 1.1–1.4 de forma paralela, ou seja, simultaneamente, pois os pontos de uma trajetória não dependem dos pontos das outras (Mittmann et al., 2008). Uma implementação seqüencial, por outro lado, calcula apenas uma trajetória por vez.



**Figura 1.1:** Resultados tractográficos obtidos com o método de propagação de linhas, vistos lateralmente. O volume de interesse contém 8.000 pontos semente e está localizado no tronco cerebral.

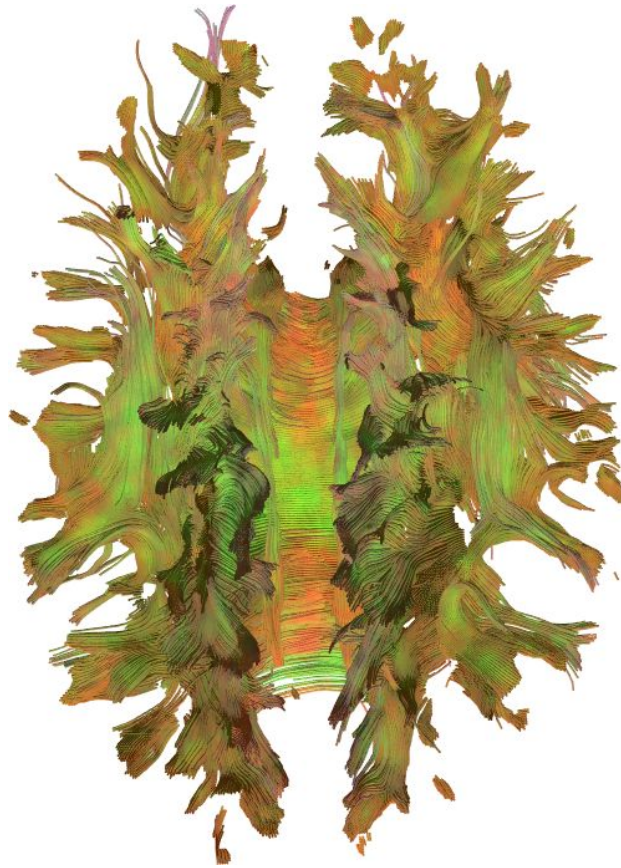
Uma plataforma de execução paralela de baixo custo é oferecida pelas unidades de processamento gráfico (GPUs) (Owens et al., 2008; Luebke e Humphreys, 2007), que são os processadores das placas de vídeo dos computadores pessoais hodiernos. Historicamente, o paralelismo nas GPUs surgiu para executar mais rapidamente operações gráficas, que podem ser aplicadas simultaneamente a vários elementos de dados. As GPUs são descritas em mais detalhe no Capítulo 3.

### 1.2 Objetivo

O objetivo deste trabalho é mostrar que as unidades de processamento gráfico atuais são capazes de executar a tractografia por propagação de linhas em tempo real. Duas ressalvas devem, porém, ser feitas: a primeira com relação à impossibilidade de se executar a tractografia em tempo real para todos os casos e a segunda com relação à própria definição de tempo real.

A tractografia por propagação de linhas calcula seu resultado a partir de um conjunto de pontos do espaço euclidiano, os chamados pontos semente. Cada um deles gera um par de trajetórias que integrará os resultados tractográficos finais. Como não existe um teto para o número de pontos semente, pode-se dizer que, para qualquer quantidade de tempo  $t$ , existe um número  $n$  de pontos semente que requererá um tempo maior que  $t$  para ser examinado pela tractografia por propagação de linhas em uma dada implementação.

Existem outros fatores que influenciam o tempo de execução de uma implementação da tractografia por propagação de linhas, mas o número de pontos semente já é suficiente para mostrar que, quando se diz que a tractografia está sendo executada em tempo real, *necessariamente* são assumidas algumas condições. Não se pode, portanto, afirmar de forma absoluta que uma implementação executa a tractografia em tempo real em



**Figura 1.2:** Resultados tractográficos vistos de cima. O volume de interesse está situado na altura do corpo caloso e contém 250.000 pontos semente distribuídos horizontalmente ao longo dos eixos  $x$  e  $y$ .

todos os casos. Para rejeitar uma tal asserção, bastaria escolher um número suficientemente grande de pontos semente para que o tempo de execução se tornasse arbitrariamente longo e, por conseguinte, incompatível com qualquer noção de tempo real.

Conseqüentemente, apenas é correto afirmar que a tractografia está sendo executada em tempo real dado um conjunto de circunstâncias — o conjunto de dados, o número de pontos semente e outros. Este trabalho mede o tempo de execução da tractografia em vários desses conjuntos circunstanciais: são utilizados quatro conjuntos de dados, um número variável de pontos semente e 100 posições diferentes do volume de interesse.

Também a noção de tempo real, no caso da tractografia, não é absoluta. Um programa que pretenda simular um processo físico, por exemplo, tem uma opção bastante natural para a definição de tempo real: se ele conseguir calcular e exibir resultados na mesma velocidade em que o processo simulado aconteceria na realidade ou mais rápido, então ele opera em tempo real.

No caso da tractografia, pode-se adotar uma definição perceptiva de tempo real. Nesse caso, a idéia é que o programa precisa responder dentro de um intervalo tempo que seja aceitável de acordo com a percepção humana (Kehtarnavaz e Gamadia, 2006). Neste trabalho, emprega-se a definição de que o intervalo de tempo aceitável é 100 ms.

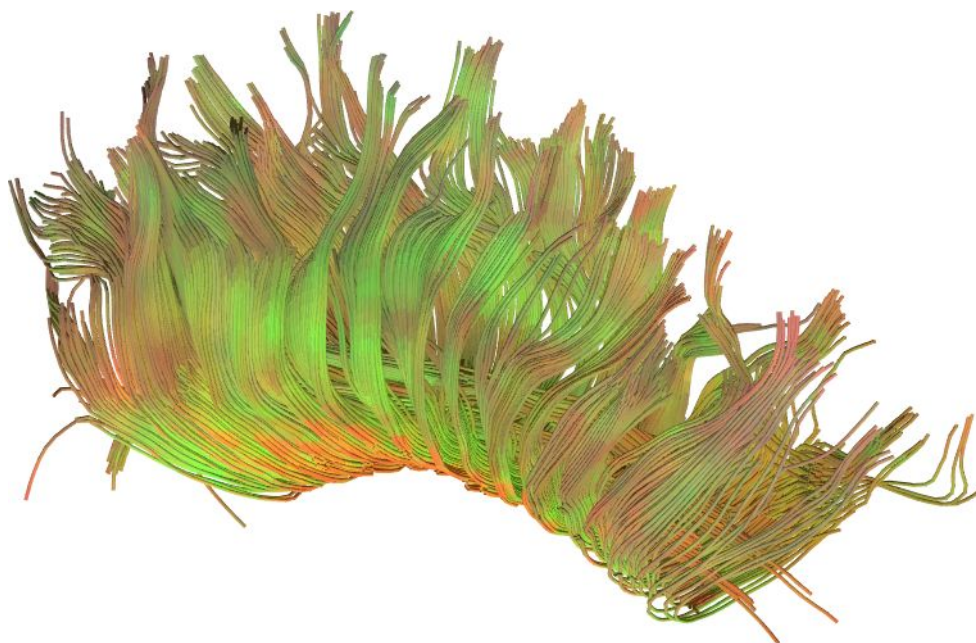


Figura 1.3: Resultados tractográficos obtidos a partir do corpo caloso com 4.000 pontos semente.

### 1.3 Método

A tractografia por propagação de linhas foi implementada para a GPU e para a CPU. As CPUs são a plataforma em que a maioria dos softwares atuais (inclusive os de tractografia) é executada em sua totalidade e por isso serve de referência para comparar tempos de execução. Quando aqui se fala em ganhos de desempenho, portanto, está sendo feita uma comparação entre os tempos de execução da GPU e da CPU.

Experimentos foram conduzidos com ambas implementações, executando-se a tractografia em todos voxels de quatro conjuntos de dados distintos, sem exibição de resultados na tela. Esses experimentos são estruturalmente bastante simples e fornecem uma comparação entre CPU e GPU sem as complexidades de uma interface gráfica. As implementações de tractografia e esses experimentos são o assunto do Capítulo 4.

Uma ferramenta de tractografia foi construída integrando-se as implementações de tractografia em CPU e GPU à estação radiológica desenvolvida pelo Grupo Cyclops (Grupo Cyclops, 2009). Este trabalho propõe um modelo de interatividade para essa ferramenta, chamado de paradigma de atualidade permanente, que, apesar de não requerer a execução da tractografia em tempo real, oferece ao usuário uma experiência otimizada quando isso acontece. A idéia desse paradigma é que os resultados tractográficos exibidos na tela estão sempre atualizados. Quando o usuário move o volume de interesse ou ajusta algum dos parâmetros, os resultados são automaticamente recalculados e exibidos. Em particular, os resultados são recalculados inclusive *enquanto* o usuário está arrastando o volume de interesse ou ajustando os parâmetros.

Para se atingir o objetivo deste trabalho, é preciso mostrar em quais circunstâncias a tractografia pode ser executada por GPUs em tempo real. Para esse fim, é definido um procedimento automático que arrasta o volume de interesse na tela ao longo de 100 posições, sendo registrado o tempo de execução de cada passo. Foram conduzidos experimentos que executam esse procedimento com 4 conjuntos de dados e várias quantidades de pontos semente. Após a execução do procedimento, é possível calcular o tempo médio de execução



**Figura 1.4:** Resultados tractográficos do cérebro inteiro, com 64.000 pontos semente.

da tractografia para o conjunto de dados e a densidade de pontos semente em questão. Os resultados finais do experimento revelam, dessa forma, as quantidades de pontos semente em que a tractografia pode ser executada em tempo real para cada conjunto de dados. Esses experimentos, bem como a ferramenta de tractografia interativa, são apresentados no Capítulo 5.

## 1.4 Trabalhos relacionados

A possibilidade de execução paralela da tractografia foi primeiro explorada por Singh et al. (2006). Os autores utilizam em sua abordagem FPGAs (field-programmable gate arrays), que são dispositivos de hardware que podem ser configurados para executar programas arbitrários. O processo da tractografia foi dividido em três núcleos: interpolação de tensores, diagonalização de tensores e cálculo da anisotropia. A aceleração da tractografia é obtida fazendo os núcleos executarem em pipeline. Mesmo sem nenhum paralelismo, a tractografia em FPGAs mostrou ser mais de 31 vezes mais rápida que a tractografia em CPU. Os autores demonstram que, com a execução paralela de 10 cadeias de núcleos, a tractografia pode ser executada mais de 300 vezes mais rápido.

Apesar de ganhos de desempenho realmente impressionantes, o uso de FPGAs tem a desvantagem do custo. Um software que os utilize para executar a tractografia precisa de um equipamento de hardware acoplado ao computador. Esse equipamento seria produzido especificamente para a tractografia e seria fornecido somente pelo fabricante do software. O custo desse equipamento sequer é estimado pelos autores, que apenas *simularam* a execução em FPGAs, ou seja, não produziram nenhum equipamento real de hardware para executá-la.

As GPUs foram utilizadas por Petrovic et al. (2007) para melhorar a visualização de resultados gerados pelo

método de propagação de linhas. O cálculo da tractografia, no entanto, não é executado na GPU. Com seu método, os autores conseguem gerar visualizações esteticamente atrativas. Um sistema de gerenciamento de nível de detalhe é apresentado e também um meio para marcar com rótulos as trajetórias visualizadas.

Um método de tractografia estocástico que é executado em GPUs foi apresentado por McGraw e Nadar (2007). Nesse método, que é diferente da propagação de linhas, um ponto semente gera mapas probabilísticos tridimensionais de conectividade. Um mapa probabilístico tem a vantagem de fornecer ao usuário um grau de certeza para a informação visualizada. O ganho de desempenho obtido pelos autores variou entre 10 e 27 vezes.

Um outro método de tractografia foi implementado para GPUs por Jeong et al. (2007). Dados dois pontos no volume, esse método encontra um conjunto de voxels que representa o melhor caminho entre os dois pontos, de acordo com um cálculo de custos. Os autores obtiveram ganhos entre 50 e 100 vezes.

O uso de GPUs para execução da tractografia pelo método de propagação de linhas foi proposto pela primeira vez por Mittmann et al. (2008). O método foi implementado na linguagem Cg e os autores obtiveram ganhos de até 40 vezes em comparação com a CPU. Além de medir o tempo de execução da tractografia, também são quantificadas as diferenças numéricas entre os resultados gerados pela CPU e pela GPU.

O método de propagação de linhas também foi implementado para a GPU por Köhn et al. (2009). Nesse trabalho, os recursos da GPU são explorados de maneira a minimizar a transferência de dados entre CPU e GPU. A GPU executou a tractografia até 40 vezes mais rápido que CPU.

Uma comparação entre uma implementação do método de propagação de linhas em GPUs e outra em clusters é apresentada por Mittmann et al. (2009). Os autores mostram que até mesmo GPUs relativamente simples executam a tractografia mais rápido que clusters de até 24 computadores. Eles argumentam, no entanto, que cada uma das duas plataformas apresentam vantagens e desvantagens que vão além do tempo de execução. Em particular, o tempo de *implementação* da tractografia para a GPU é muito maior do que para a CPU.

### 1.5 Convenções e notações

Este trabalho emprega muitas palavras do idioma inglês que não tem nenhum equivalente corrente em português (por exemplo, “hardware” e “thread”) ou cujo equivalente em português é pouco usado (“agregado” é às vezes usado para “cluster”). Para não obstruir a leitura do texto com torrentes de palavras em itálico, optou-se por grafar todas as palavras estrangeiras sem destaque nenhum. Pretende-se com isso tão-somente dar um aspecto mais agradável ao texto, sem nenhuma pretensão de lesar a língua vernácula. Também com o objetivo de desimpedir a leitura, as siglas estão escritas em versalete e não em maiúsculas, devido à grande frequência delas no texto (especialmente “CPU” e “GPU”).

Equações e fórmulas propriamente só aparecem no Capítulo 2, em que a tractografia por propagação de linhas é descrita. As seguintes notações são utilizadas naquele capítulo:



<b>Exemplo</b>	<b>Descrição</b>	<b>Entidades</b>
$\underline{D}, \underline{T}_1$	Letra latina sublinhada	Tensor
$\mathbf{x}, \mathbf{J}$	Letra latina em negrito	Vetor ou ponto
$\epsilon_1, \epsilon_2$	Letra grega $\epsilon$	Autovetores
$\lambda_1, \lambda_2$	Letra grega $\lambda$	Autovalores

Em adição às operações mais básicas, também estas são utilizadas:

<b>Exemplo</b>	<b>Descrição</b>
$\ x\ $	Norma do vetor $x$
$x^T$	Transposta do vetor $x$
$\underline{T}_1 \cdot \underline{T}_2$	Produto escalar dos tensores $\underline{T}_1$ e $\underline{T}_2$
$A^{-1}$	Inversa da matriz $A$

# Capítulo 2

## Tractografia

### 2.1 Princípios físicos e matemáticos

Em um meio líquido, a intensidade do fluxo de partículas é proporcional ao gradiente do campo de concentração e ao coeficiente de difusão. A primeira lei de Fick estabelece a relação entre o vetor de fluxo  $J$  e o campo de concentração  $C$  da seguinte forma:

$$J = -D\nabla C. \quad (2.1)$$

O escalar  $D$  é o coeficiente de difusão, que é uma propriedade do meio. O sinal negativo indica que o fluxo tem o sentido contrário com relação ao gradiente de concentração, isto é, o fluxo ocorre de áreas mais concentradas para menos concentradas. Se adotarmos o segundo como unidade de tempo, o milímetro como unidade de comprimento e o mol como medida de quantidade de matéria, as unidades da Equação 2.1 são

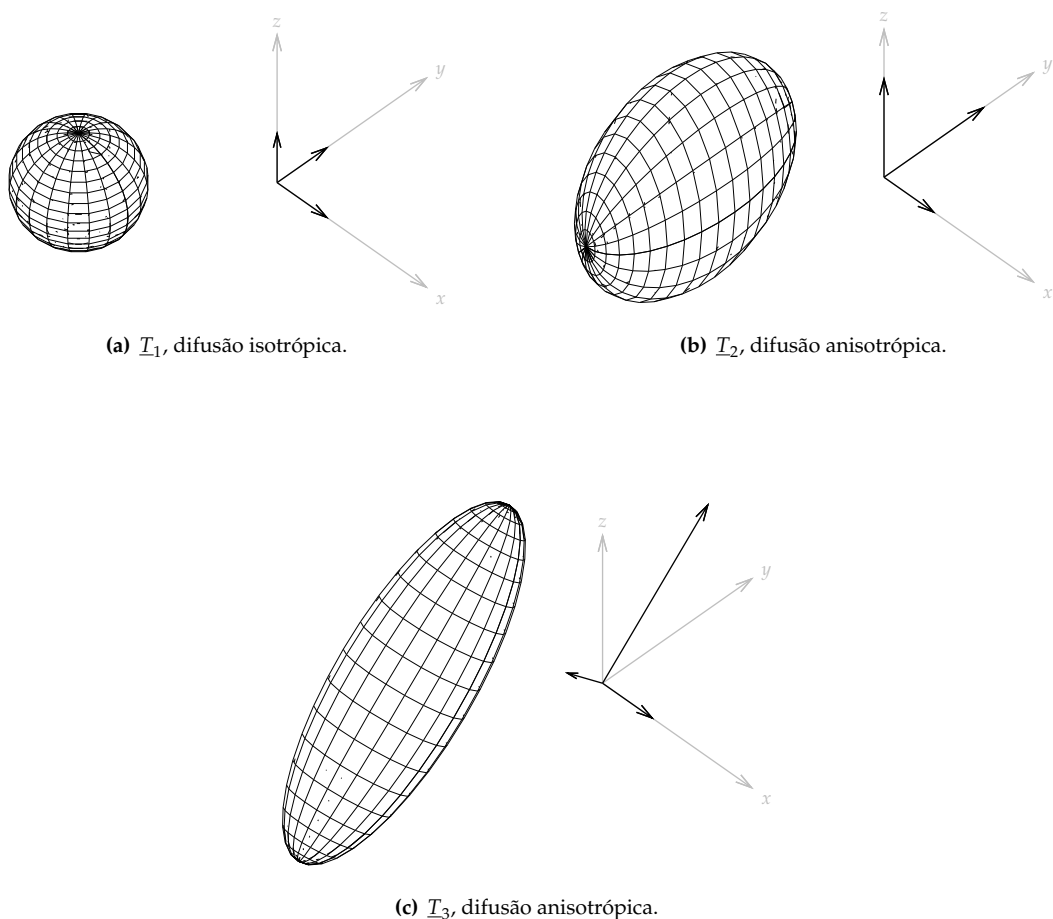
$$\frac{\text{mol}}{\text{mm}^2 \text{ s}} = \frac{\text{mm}^2}{\text{s}} \frac{\text{mol}}{\text{mm}^4}.$$

O fluxo, portanto, é expresso pelo número de partículas que passam por uma área de  $1 \text{ mm}^2$  em um segundo.

Quando a concentração é homogênea, ou seja,  $\nabla C = 0$ , o fluxo é zero. As moléculas de um líquido como a água, no entanto, não permanecem estáticas, mesmo quando não há diferenças de concentração. Ao invés disso, elas deslocam-se aleatoriamente no meio; esse deslocamento aleatório de moléculas é conhecido como movimento browniano. Uma intensidade de fluxo zero indica, assim, que o fluxo *médio* é zero, e não que não haja movimentação alguma de moléculas. Ademais, mesmo que o fluxo  $J$  seja zero, a intensidade do movimento browniano ainda depende do coeficiente de difusão.

Em tecidos biológicos o movimento browniano da água não é completamente livre, pois barreiras como as membranas celulares restringem seu movimento (Neil, 1997). A difusão que acontece dessa forma é chamada de anisotrópica, porque manifesta-se de modo desigual. O coeficiente escalar de difusão  $D$ , porém, não é capaz de expressar intensidades de difusão distintas em direções diferentes. Por esse motivo, os tensores de difusão são utilizados quando se sabe ou se espera que a difusão seja anisotrópica.

Tensores são generalizações de algumas entidades matemáticas. Eles são caracterizados por uma ordem, que indica a quantidade de índices necessária para referir-se aos seus elementos. Um tensor de ordem zero é um escalar, que não precisa de nenhum índice; um tensor de ordem 1 é um vetor (linha ou coluna); um tensor



**Figura 2.1:**  $\underline{T}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ ,  $\underline{T}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ , e  $\underline{T}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ .

de ordem 2 é uma matriz bidimensional. Tensores de ordem maior que 2 também são possíveis.

O tensor de difusão tem ordem 2 e sua matriz, em três dimensões, tem a seguinte forma:

$$\underline{D} = \begin{bmatrix} d_{xx} & d_{xy} & d_{xz} \\ d_{xy} & d_{yy} & d_{yz} \\ d_{xz} & d_{yz} & d_{zz} \end{bmatrix}.$$

A matriz do tensor de difusão, além de simétrica, também é positiva definida, isto é, seus autovalores são todos positivos. Essas duas propriedades asseguram que haja uma correspondência biunívoca entre cada tensor de difusão e uma elipsóide (em três dimensões). A forma da elipse ou elipsóide associada a um tensor de difusão representa a forma da difusão das partículas.

A Figura 2.1 mostra as elipsóides associadas a três tensores de difusão diferentes. O tensor  $\underline{T}_1$  expressa uma difusão isotrópica, constante em todas as direções. O tensor  $\underline{T}_2$  indica que a difusão acontece com o dobro de força ao longo dos eixos  $x$  e  $y$ . Por fim, o tensor  $\underline{T}_3$  indica que a principal direção de difusão está alinhada a um vetor rodado  $45^\circ$  ao longo do eixo  $x$  a partir do eixo  $z$ .

À direita de cada elipsóide da Figura 2.1 estão os seus eixos principais, que estão diretamente associados aos autovetores e autovalores do tensor de difusão. Os três autovetores  $\epsilon_i$  e seus autovalores associados  $\lambda_i$

podem ser encontrados de várias formas e, em particular, existem rotinas computacionais otimizadas para matrizes simétricas e positiva definidas.

No caso do tensor  $\underline{T}_3$  da Figura 2.1, por exemplo, os autovetores são

$$\epsilon_1 = \begin{bmatrix} 0 \\ \sqrt{2} \\ \sqrt{2} \end{bmatrix}, \epsilon_2 = \begin{bmatrix} 0 \\ \sqrt{2} \\ -\sqrt{2} \end{bmatrix} \text{ e } \epsilon_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

e os autovalores de cada autovetor são, respectivamente,

$$\lambda_1 = 3, \lambda_2 = 1 \text{ e } \lambda_3 = 1.$$

Porque a matriz do tensor de difusão é simétrica e positiva definida, os autovetores são sempre ortogonais (Basser et al., 1994b). Dois vetores são ortogonais se seu produto escalar é zero, o que pode ser facilmente verificado para os autovetores encontrados acima:

$$\begin{aligned} \epsilon_1 \cdot \epsilon_2 &= 0 \times 0 + \sqrt{2} \times \sqrt{2} + \sqrt{2} \times -\sqrt{2} &&= 0 + 2 - 2 = 0, \\ \epsilon_1 \cdot \epsilon_3 &= 0 \times 1 + \sqrt{2} \times 0 + \sqrt{2} \times 0 &&= 0 + 0 + 0 = 0, \\ \epsilon_2 \cdot \epsilon_3 &= 0 \times 1 + \sqrt{2} \times 0 - \sqrt{2} \times 0 &&= 0 + 0 + 0 = 0. \end{aligned}$$

Os autovetores e autovalores do tensor de difusão têm uma interpretação física bastante específica: os autovetores correspondem às principais direções de difusão e os autovalores aos coeficientes de difusão escalares ao longo desses vetores (Basser et al., 1994b). De maneira mais geral, o coeficiente de difusão  $D$  em qualquer direção pode ser obtido por

$$D = \|\underline{D}\mathbf{v}\|, \quad (2.2)$$

em que  $\underline{D}$  é o tensor de difusão e  $\mathbf{v}$  o vetor unitário que expressa a direção. Fica claro que os autovetores também obedecem à Equação 2.2, já que a propriedade fundamental de um autovetor  $\mathbf{x}$  de uma matriz  $A$  é que

$$A\mathbf{x} = \lambda\mathbf{x}.$$

A Equação 2.2, portanto, fica, para um autovetor  $\epsilon_i$  do tensor  $\underline{D}$ ,

$$D = \|\underline{D}\epsilon_i\| = \|\lambda_i\epsilon_i\| = \lambda_i.$$

A primeira lei de Fick pode, assim, ser reescrita da seguinte forma para utilizar um tensor de difusão ao invés de um escalar (Basser et al., 1994b):

$$J = -\underline{D}\nabla C. \quad (2.3)$$

## 2.2 Ressonância magnética de tensores de difusão

A ressonância magnética é uma técnica que produz imagens que revelam a concentração de água no corpo humano (Mori, 2007). As imagens são obtidas através da aplicação de uma série de campos magnéticos e

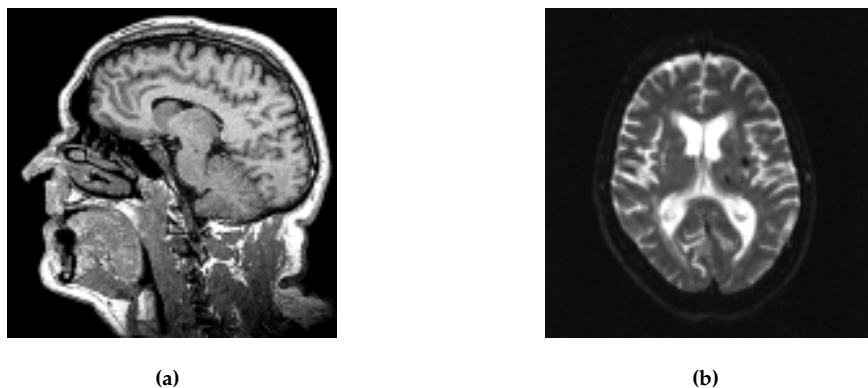


Figura 2.2: Exemplos de imagens de ressonância magnética.

pulsos de radiofrequência. Conforme os parâmetros da aquisição são alterados, o sinal das imagens pode refletir diferentes características dos tecidos, o que traz informações úteis, por exemplo, para o diagnóstico de doenças específicas (Mori e Barker, 1999). A Figura 2.2 mostra dois exemplos de imagens de ressonância magnética.

É possível também medir a intensidade da difusão da água através da ressonância magnética. Para fazê-lo, dois gradientes são aplicados aos campos magnéticos, separados por um intervalo de tempo. Ambos os gradientes têm a mesma intensidade e direção, porém sentidos opostos. Os gradientes fazem com que o campo magnético seja aplicado desigualmente, o que confere excitações diferentes aos prótons conforme sua localização ao longo dos gradientes. O segundo gradiente anula a excitação produzida pelo primeiro, pois tem sentido oposto, porém essa anulação não acontece com os prótons que se difundiram ao longo da direção do gradiente. O sinal contido nas imagens obtidas dessa forma contém justamente informação a respeito dos prótons que se moveram ao longo do gradiente.

A Figura 2.3 mostra seis imagens obtidas com diferentes gradientes aplicados ao campo magnético. As seis imagens são do mesmo exame e da mesma fatia.

### 2.2.1 Equação de Stejskal–Tanner

A equação de Stejskal–Tanner (Stejskal e Tanner, 1964) permite relacionar o sinal  $S$  obtido com os gradientes ativos ao coeficiente de difusão  $D$  do meio:

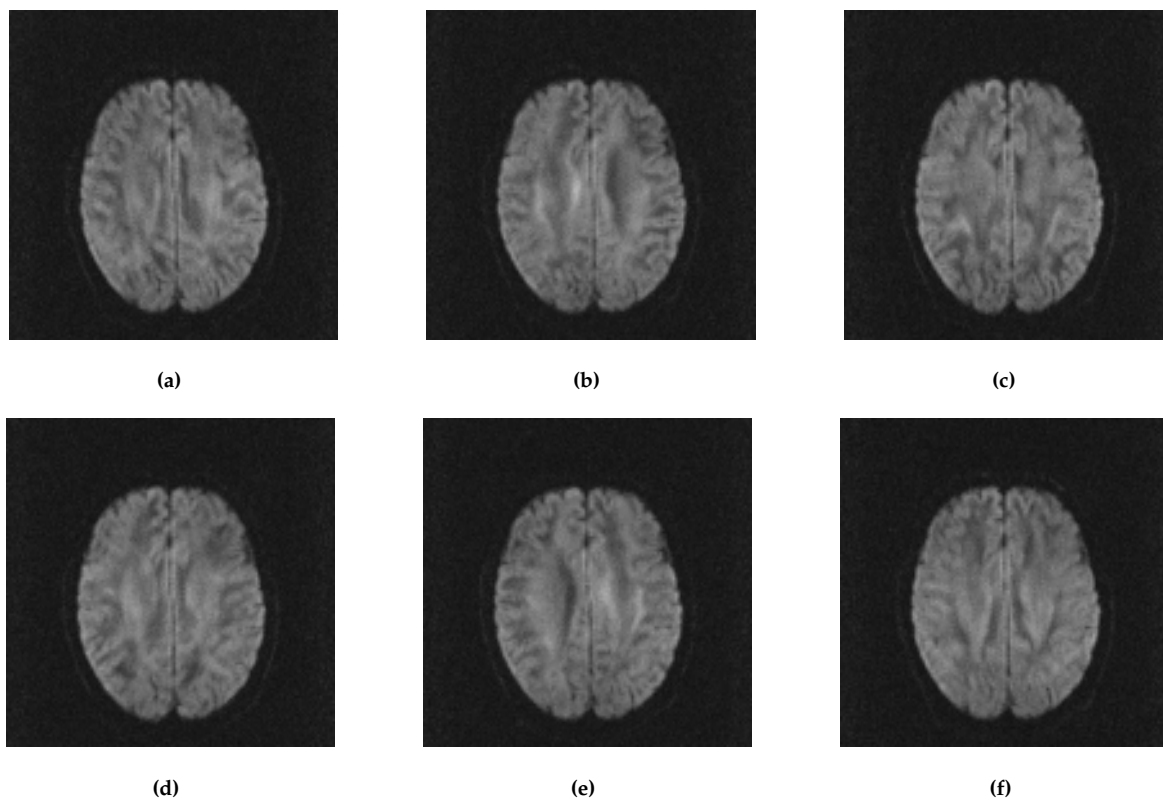
$$\ln\left(\frac{S}{S_0}\right) = -bD. \quad (2.4)$$

Aqui,  $S_0$  é o sinal obtido sem os gradientes e  $b$  é uma constante calculada a partir dos parâmetros de aquisição das imagens. Em particular, para gradientes na forma de quadrados, o valor de  $b$  é definido por (Basser et al., 1994a)

$$b = \gamma^2 G^2 \delta^2 \left( \Delta - \frac{\delta}{3} \right), \quad (2.5)$$

em que  $\gamma$  é a razão giromagnética do hidrogênio,  $G$  é a intensidade dos gradientes,  $\delta$  é a duração dos gradientes e  $\Delta$  o intervalo entre eles. O sinal  $S_0$  é obtido com  $b = 0$ .

Através de duas imagens, uma obtida com gradientes ativos e outra sem eles, é possível calcular o coeficiente de difusão. Sendo o valor de cada pixel das imagens  $S$  e  $S_0$ , respectivamente, o coeficiente de difusão



**Figura 2.3:** Exemplos de imagens de ressonância magnética obtidas com gradientes aplicados ao campo magnético. Cada imagem contém informação acerca da difusão em uma direção específica.

é

$$D = -\frac{\ln\left(\frac{S}{S_0}\right)}{b}. \quad (2.6)$$

O coeficiente escalar  $D$  obtido dessa forma, entretanto, refere-se à difusão somente ao longo do gradiente aplicado. Caso se trate de um meio isotrópico,  $D$  é suficiente para caracterizar completamente a difusão, mas em meios anisotrópicos como tecidos nervosos o tensor de difusão é utilizado. Nesse caso, a Equação 2.4 é reescrita da seguinte forma (Basser et al., 1994b):

$$\ln\left(\frac{S}{S_0}\right) = -b\mathbf{g}^T \mathbf{g} \cdot \underline{D}. \quad (2.7)$$

Aqui,  $\cdot$  é a operação de produto escalar,  $\underline{D}$  é o tensor de difusão, propriedade do meio, e  $\mathbf{g}$  é o vetor linha unitário na direção do gradiente aplicado.

Diferentemente da Equação 2.4, que pode ser resolvida com a aquisição de apenas duas imagens, a Equação 2.7 precisa de pelo menos sete imagens para que o tensor  $\underline{D}$  seja calculado, uma adquirida sem gradientes e seis outras com gradientes aplicados em direções não colineares. Com o sinal  $S_0$  sem gradientes e os sinais  $S_i$  obtidos com gradientes  $\mathbf{g}_i$ , obtém-se o sistema de equações

$$\ln\left(\frac{S_i}{S_0}\right) = -b\mathbf{g}_i^T \mathbf{g}_i \cdot \underline{D}. \quad (2.8)$$

As incógnitas do sistema são os seis elementos independentes do tensor  $\underline{D}$ .

### 2.2.2 Cálculo do tensor

Para resolver esse sistema, convencionalmente ele é expresso sob a forma

$$AX = B,$$

em que a matriz  $A$  contém os coeficientes do sistema, o vetor coluna  $X$  as incógnitas e o vetor coluna  $B$  as constantes. Assumindo que o número de imagens adquiridas seja sete, o vetor  $B$  pode ser expresso por

$$B = \begin{bmatrix} \ln(S_1/S_0) \\ \ln(S_2/S_0) \\ \ln(S_3/S_0) \\ \ln(S_4/S_0) \\ \ln(S_5/S_0) \\ \ln(S_6/S_0) \end{bmatrix},$$

o que corresponde ao lado esquerdo da Equação 2.8.

Para obter a matriz de coeficientes  $A$ , o lado direito da Equação 2.8 precisa ser expandido. Levando-se em consideração que o tensor de difusão pode ser expresso por

$$\underline{D} = \begin{bmatrix} d_{xx} & d_{xy} & d_{xz} \\ d_{xy} & d_{yy} & d_{yz} \\ d_{xz} & d_{yz} & d_{zz} \end{bmatrix},$$

o lado direito da equação é, para cada gradiente  $g_i$  correspondente a uma das seis imagens adquiridas com difusão,

$$\begin{aligned} -b g_i^T g_i \cdot \underline{D} &= -b \begin{bmatrix} g_{ix}g_{ix} & g_{ix}g_{iy} & g_{ix}g_{iz} \\ g_{ix}g_{iy} & g_{iy}g_{iy} & g_{iy}g_{iz} \\ g_{ix}g_{iz} & g_{iy}g_{iz} & g_{iz}g_{iz} \end{bmatrix} \cdot \begin{bmatrix} d_{xx} & d_{xy} & d_{xz} \\ d_{xy} & d_{yy} & d_{yz} \\ d_{xz} & d_{yz} & d_{zz} \end{bmatrix} \\ &= -b(g_{ix}^2 d_{xx} + g_{iy}^2 d_{yy} + g_{iz}^2 d_{zz} + \\ &\quad 2g_{ix}g_{iy}d_{xy} + 2g_{ix}g_{iz}d_{xz} + 2g_{iy}g_{iz}d_{yz}). \end{aligned}$$

A expansão acima mostra os coeficientes de cada uma das incógnitas  $d_{xx}$ ,  $d_{yy}$ ,  $d_{zz}$ ,  $d_{xy}$ ,  $d_{xz}$  e  $d_{yz}$ . A partir desses coeficientes, pode-se montar a matriz  $A$ , que assume a forma

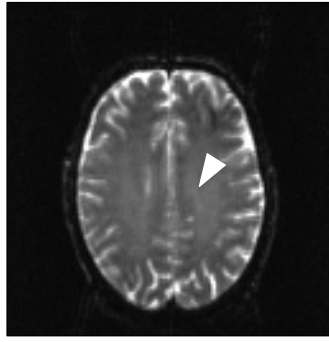
$$A = -b \begin{bmatrix} g_{1x}^2 & g_{1y}^2 & g_{1z}^2 & 2g_{1x}g_{1y} & 2g_{1x}g_{1z} & 2g_{1y}g_{1z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{6x}^2 & g_{6y}^2 & g_{6z}^2 & 2g_{6x}g_{6y} & 2g_{6x}g_{6z} & 2g_{6y}g_{6z} \end{bmatrix}.$$

O sistema pode finalmente ser resolvido, por exemplo, através da inversão da matriz  $A$ . Se o sistema original é

$$AX = B,$$

então multiplicam-se ambos os lados da equação pela inversa de  $A$ ,

$$A^{-1}AX = A^{-1}B.$$



**Figura 2.4:** Imagem de ressonância magnética obtida sem aplicação de gradientes.

No lado esquerdo agora a matriz identidade multiplica  $X$  e pode ser portanto omitida,

$$X = A^{-1}B.$$

A partir do valor das seis incógnitas, o tensor de difusão pode ser finalmente montado.

Considere, por exemplo, o cálculo do tensor no ponto marcado na imagem da Figura 2.4, que foi obtida juntamente com as outras seis da Figura 2.3, porém sem aplicação de gradientes. Primeiramente, sabe-se que os gradientes aplicados às imagens da Figura 2.3 foram

$$g_1 = \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \end{bmatrix}, \quad g_2 = \begin{bmatrix} \sqrt{2}/2 \\ 0 \\ -\sqrt{2}/2 \end{bmatrix}, \quad g_3 = \begin{bmatrix} 0 \\ -\sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix},$$

$$g_4 = \begin{bmatrix} -\sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \end{bmatrix}, \quad g_5 = \begin{bmatrix} \sqrt{2}/2 \\ 0 \\ \sqrt{2}/2 \end{bmatrix}, \quad g_6 = \begin{bmatrix} 0 \\ \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}.$$

Observa-se que esse vetores são todos unitários e não colineares.

Considerando-se que o valor de  $b$  para essas imagens foi  $b = 900$ , é possível encontrar a matriz de coeficientes  $A$  do sistema linear:

$$A = \begin{bmatrix} -450 & -900 & 0 & -450 & 0 & 0 \\ -450 & 0 & 900 & 0 & 0 & -450 \\ 0 & 0 & 0 & -450 & 900 & -450 \\ -450 & 900 & 0 & -450 & 0 & 0 \\ -450 & 0 & -900 & 0 & 0 & -450 \\ 0 & 0 & 0 & -450 & -900 & -450 \end{bmatrix}.$$

O valor dos pixels de cada uma das imagens com gradiente, que é o sinal observado, pode ser reunido em



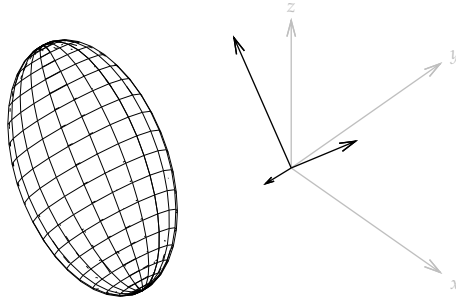


Figura 2.5: Elipsóide do tensor  $\underline{D}$ , calculado como exemplo.

uma matriz:

$$S = \begin{bmatrix} 227 \\ 160 \\ 214 \\ 188 \\ 204 \\ 206 \end{bmatrix}.$$

O sinal da imagem sem gradiente, no pixel indicado, foi  $S_0 = 394$ , o que permite que a matriz de constantes  $B$  do sistema seja obtida:

$$B = \begin{bmatrix} \ln(227/394) \\ \ln(160/394) \\ \ln(214/394) \\ \ln(188/394) \\ \ln(204/394) \\ \ln(206/394) \end{bmatrix} = \begin{bmatrix} -0.55140 \\ -0.90118 \\ -0.61037 \\ -0.73991 \\ -0.65823 \\ -0.64847 \end{bmatrix}.$$

De posse da matriz de coeficientes  $A$  e das constantes em  $B$ , obtém-se a solução do sistema:

$$X = \begin{bmatrix} 884.371 \\ -104.727 \\ -134.970 \\ 550.417 \\ 21.167 \\ 848.304 \end{bmatrix} \times 10^{-6}.$$

Mapeando-se os seis valores de  $X$  para o tensor de difusão, chega-se ao resultado

$$\underline{D} = \begin{bmatrix} 884.371 & -104.727 & -134.970 \\ -104.727 & 550.417 & 21.167 \\ -134.970 & 21.167 & 848.304 \end{bmatrix} \times 10^{-6}.$$

O tensor  $\underline{D}$  encontrado está representado na Figura 2.5.

### 2.2.3 Índices invariantes

A partir dos tensores de difusão pode-se calcular uma série de escalares, aqui chamados de índices, que ajudam a caracterizá-los, especialmente no que se refere à sua forma e tamanho. Esses escalares são invariantes, ou seja, têm o mesmo valor independentemente da orientação do tensor (Basser et al., 1994b). A utilidade dos índices está em fornecer informações relevantes sobre cada ponto do volume aos métodos de tractografia. Nesta seção, os três autovetores do tensor de difusão são denotados  $e_i$  e os autovalores associados  $\lambda_i$ .

Basser et al. (1994b) identificaram três escalares invariantes. São eles:

$$I_1 = \lambda_1 + \lambda_2 + \lambda_3, \quad (2.9)$$

$$I_2 = \lambda_1\lambda_2 + \lambda_3\lambda_1 + \lambda_2\lambda_3, \text{ e} \quad (2.10)$$

$$I_3 = \lambda_1\lambda_2\lambda_3. \quad (2.11)$$

A invariância desses escalares decorre diretamente do fato de que não se baseiam nos elementos do tensor, mas nos autovalores, que são em si invariantes. O índice  $I_1$  é um escalar mais conhecido pela designação de traço do tensor, e é mais freqüentemente indicado por

$$\text{tr}(\underline{D}) = \lambda_1 + \lambda_2 + \lambda_3 = d_{xx} + d_{yy} + d_{zz}. \quad (2.12)$$

Um importante índice derivado do traço do tensor é a difusividade média, que é a média das três principais intensidades de difusão do tensor:

$$\text{MD}(\underline{D}) = \frac{\text{tr}(\underline{D})}{3}. \quad (2.13)$$

Outra notação utilizada para se referir à difusividade média de um tensor  $\underline{D}$  é  $\langle D \rangle$ .

Basser e Pierpaoli (1996) propuseram índices para medir a isotropia e a anisotropia dos tensores de difusão. A abordagem desses autores consiste em decompor o tensor em duas componentes, uma isotrópica e outra anisotrópica. Em seguida, derivam um escalar que corresponde às magnitudes dessas componentes. Por fim, os escalares obtidos dessa forma são combinados para gerar os índices de anisotropia relativa (RA) e anisotropia fracional (FA).

A decomposição do tensor em sua parte isotrópica e anisotrópica é feita com base na equação (Basser e Pierpaoli, 1996)

$$\underline{D} = \langle D \rangle \underline{I} + (\underline{D} - \langle D \rangle \underline{I}), \quad (2.14)$$

em que o tensor  $\underline{I}$  é o tensor unitário, isto é,

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

A primeira parcela da soma da Equação 2.14 corresponde à sua parte isotrópica e a segunda à sua parte anisotrópica. Essas duas partes serão aqui denominadas, respectivamente,

$$\underline{D}^i = \langle D \rangle \underline{I}$$

e

$$\underline{D}^a = \underline{D} - \langle D \rangle I.$$

A magnitude de um tensor  $\underline{D}$  é a raiz quadrada do produto escalar (Basser e Pierpaoli, 1996),  $\sqrt{\underline{D} \cdot \underline{D}}$ , em que

$$\underline{D} \cdot \underline{D} = \sum_{i=1}^3 \sum_{j=1}^3 d_{ij}^2 = \lambda_1^2 + \lambda_2^2 + \lambda_3^2. \quad (2.15)$$

Com base nessas definições, Basser e Pierpaoli (1996) definiram o índice de anisotropia relativa RA de um tensor  $\underline{D}$  como sendo

$$RA = \frac{\sqrt{\underline{D}^a \cdot \underline{D}^a}}{\sqrt{\underline{D} \cdot \underline{D}}}, \quad (2.16)$$

isto é, a razão entre a magnitude da componente anisotrópica e a magnitude da componente isotrópica do tensor. O índice de anisotropia fracional FA é dado por

$$FA = \frac{\sqrt{3}}{\sqrt{2}} \frac{\sqrt{\underline{D}^a \cdot \underline{D}^a}}{\sqrt{\underline{D} \cdot \underline{D}}}, \quad (2.17)$$

ou seja, a razão entre a magnitude da componente anisotrópica e a magnitude do tensor original. O índice FA assume valores no domínio  $[0, 1]$ . Um meio totalmente isotrópico tem como resultado  $FA = 0$ , e um meio em que uma das direções de difusão é muito maior que as demais, isto é,  $\lambda_1 \gg \lambda_2 \approx \lambda_3$ , resulta em um valor de FA próximo de 1.

Baseando-se na relação entre os autovalores do tensor de difusão, Westin et al. (1999, 2002) dividiram a difusão em três casos:

- Caso linear, em que  $\lambda_1 \gg \lambda_2 \approx \lambda_3$ : a difusão acontece principalmente na direção do autovetor  $\epsilon_1$ ;
- Caso planar, em que  $\lambda_1 \approx \lambda_2 \gg \lambda_3$ : a difusão restringe-se ao plano definido pelos autovetores  $\epsilon_1$  e  $\epsilon_2$ ;
- Caso esférico, em que  $\lambda_1 \approx \lambda_2 \approx \lambda_3$ : a difusão é isotrópica.

Para quantificar cada um dos três casos, Westin et al. (2002) definiram os seguintes índices:

$$c_l = \frac{\lambda_1 - \lambda_2}{\lambda_1}, \quad (2.18)$$

$$c_p = \frac{\lambda_2 - \lambda_3}{\lambda_1}, \quad (2.19)$$

e

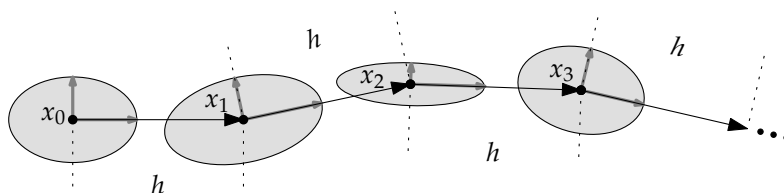
$$c_s = \frac{\lambda_3}{\lambda_1}. \quad (2.20)$$

Todos esses índices variam entre zero e um e apresentam a propriedade adicional de que

$$c_l + c_p + c_s = 1,$$

ou seja, a difusão do tensor é efetivamente particionada entre os três casos.

Shimony et al. (1999) também propuseram índices de anisotropia. Os autores sugerem, entre outros, o índice  $A_\sigma$ , baseado na variância do tensor.



**Figura 2.6:** Funcionamento do método de propagação de linhas. A partir de cada ponto  $x_n$ , um novo ponto  $x_{n+1}$  é encontrado a partir do anterior através da principal direção de difusão.

## 2.3 Tractografia

### 2.3.1 Propagação de linhas

O método de propagação de linhas busca revelar a localização dos feixes de nervos do cérebro de uma maneira bastante intuitiva. Dado um ponto  $x_0$  no espaço, esse método encontra o tensor de difusão em  $x_0$ . Os autovetores  $\epsilon_i$  do tensor são calculados e o método assume que a principal direção de difusão do tensor, dada por  $\epsilon_1$ , corresponde ao alinhamento das fibras naquele ponto do espaço, que, presume-se, esteja dentro do cérebro do paciente. O método então calcula um segundo ponto a partir do ponto  $x_0$  e do autovetor  $\epsilon_1$ , segundo a equação

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\epsilon_1, \quad (2.21)$$

em que  $\epsilon_1$  é o principal autovetor do tensor em  $x_n$  e  $h$  é uma constante que indica o tamanho do passo.

Dessa forma, o método de propagação de linhas encontra uma seqüência de pontos  $x_n$  seguindo a direção principal de difusão. Essa estratégia está ilustrada na Figura 2.6, em que tensores bidimensionais são representados por elipses. As setas cinzas dentro das elipsóides representam as duas principais direções de difusão. As setas pretas mostram a direção seguida pelo método em cada ponto. Observa-se que a distância entre os pontos sucessivamente calculados pelo método é sempre igual a  $h$ .

Em algum momento, o método precisa parar de encontrar novos pontos. Os feixes nervosos do cérebro têm tamanho finito, então nada mais natural que o método encontre um número finito de pontos. Para saber quando deve parar, o método de propagação de linhas adota um conjunto de critérios de parada. Os critérios de parada são tipicamente baseados nos índices invariantes vistos anteriormente. O método pode estabelecer, por exemplo, que deve parar se a difusividade média cair abaixo de um certo limiar, o que indicaria que a difusão de água está muito fraca para ser considerada relevante; ou se a anisotropia fracional ficar muito próxima de zero, o que indica que a principal direção de difusão não é bem distinta das demais e que portanto não existe propriamente uma direção principal de difusão.

Em adição aos índices invariantes, outro critério de parada utilizado é um valor máximo para o ângulo formado entre dois segmentos de reta sucessivos. Ao encontrar o ponto  $x_{n+2}$ , o método encontra o ângulo  $\theta$  entre os segmentos de reta  $(x_{n+2}, x_{n+1})$  e  $(x_{n+1}, x_n)$ . Se  $\theta$  for maior que um certo limiar, o método pode decidir parar para evitar que mudanças bruscas de direção apareçam no resultado. Porque os segmentos de reta seguem a direção dos autovetores principais, o ângulo  $\theta$  pode ser calculado através da relação

$$\epsilon_1^{n+1} \cdot \epsilon_1^n = |\epsilon_1^{n+1}| |\epsilon_1^n| \cos(\theta). \quad (2.22)$$

É interessante observar que o tensor de difusão indica as principais direções de difusão, mas não o *sentido*

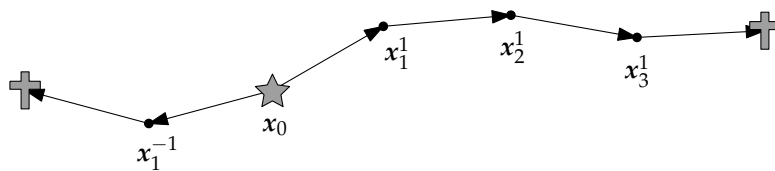


Figura 2.7: As duas trajetórias encontradas a partir de um ponto semente.

da difusão. Em outras palavras, se a principal direção de difusão de um tensor é dada pelo vetor  $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ , sabe-se que a difusão acontece ao longo do eixo  $z$ , mas não se pode dizer que ela ocorre para cima ou para baixo. No caso da ressonância magnética de tensores de difusão, essa propriedade do tensor não representa uma limitação, pois a difusão que se quer medir é a do movimento browniano, que, apesar de restrita, acontece aleatoriamente em todas as direções.

Essa ambigüidade de sentido precisa ser levada em consideração pelo método de propagação de linhas. Ao iniciar de um ponto  $x_0$ , o método precisa seguir *dois* sentidos, um correspondente a  $\epsilon_1$  e outro a  $-\epsilon_1$ . Esse duplo caminho, no entanto, afeta somente o ponto  $x_0$ . A partir do ponto  $x_1$ , somente o sentido mais alinhado ao segmento de reta recém encontrado é levada em conta. Considera-se, por conseguinte, que o sentido menos alinhado não representa um caminho válido. Os pontos de um dos caminhos seguidos pelo método são aqui denotados por  $x_i^1$  e os do outro caminho  $x_i^{-1}$ .

Uma trajetória é um conjunto de  $n + 1$  pontos encontrados pelo método de propagação de linhas. A partir de um ponto inicial  $x_0$  as duas trajetórias encontradas,  $T^1$  e  $T^{-1}$ , são denotadas por

$$T^i = \{x_k^i : k \in \mathbb{Z} \wedge 0 \leq k \leq n\}. \quad (2.23)$$

O ponto inicial  $x_0$  de uma trajetória recebe o nome especial de ponto semente, pois é a partir dele que os demais pontos “brotam”. A Figura 2.7 mostra um ponto semente e as duas trajetórias encontradas a partir dele. A estrela marca o ponto semente e as cruces os pontos em que o método decidiu parar.

Quando o método de propagação de linhas é adotado, em geral não se utiliza apenas um ponto semente, mas um conjunto deles. O tamanho desse conjunto varia muito de uma aplicação para outra, mas são comuns, por exemplo, conjuntos de pontos semente com número de elementos entre 50 e 5.000. Ao conjunto de trajetórias  $T_i^j$  encontrados pelo método de propagação de linhas dá-se o nome de resultado tractográfico.

### 2.3.2 Formalização

O método de propagação de linhas pode ser expresso de uma maneira mais formal do que a expressa na seção anterior (Basser et al., 2000, 2002; Bammer et al., 2003). Considerando-se que a curva que o método tenta encontrar pode ser representada por um vetor  $\mathbf{r}(s)$ , parametrizado pelo comprimento de arco  $s$ , obtém-se a equação

$$\frac{d\mathbf{r}(s)}{ds} = \mathbf{t}(s), \quad (2.24)$$

em que  $\mathbf{t}(s)$  é o vetor unitário tangente à curva  $\mathbf{r}(s)$  na posição  $s$ . Sendo  $\mathbf{t}(s)$  identificado com a principal direção de difusão, segue que

$$\mathbf{t}(s) = \epsilon_1(\mathbf{r}(s)).$$

Como o método começa a partir de um ponto semente  $x_0$ , sabe-se também que

$$\mathbf{r}(0) = \mathbf{x}_0. \quad (2.25)$$

A solução para a Equação 2.24 pode ser encontrada, a partir da condição inicial especificada pela Equação 2.25, através de métodos clássicos. A maneira como o ponto  $\mathbf{x}_{n+1}$  é obtido a partir do ponto  $\mathbf{x}_n$  na seção anterior corresponde ao método de Euler. Nesse método, a solução da Equação 2.24 seria aproximada com o passo

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\epsilon_1(\mathbf{r}_n), \quad (2.26)$$

em que  $h$  é o tamanho do passo. Note-se que essa equação é idêntica à Equação 2.21.

O método de Euler, entretanto, tem algumas desvantagens conhecidas (Press et al., 1992). Ele não é muito preciso quando comparado a outros métodos e também não é numericamente muito estável. Por esse motivo, o método de Runge–Kutta de quarta ordem é mais freqüentemente utilizado. Nesse método, o passo é, para a Equação 2.24,

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \frac{h}{6} (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4), \quad (2.27)$$

em que os  $\mathbf{k}_i$  são definidos por

$$\begin{aligned} \mathbf{k}_1 &= \epsilon_1(\mathbf{r}_n), \\ \mathbf{k}_2 &= \epsilon_1\left(\mathbf{r}_n + \frac{h\mathbf{k}_1}{2}\right), \\ \mathbf{k}_3 &= \epsilon_1\left(\mathbf{r}_n + \frac{h\mathbf{k}_2}{2}\right) \text{ e} \\ \mathbf{k}_4 &= \epsilon_1(\mathbf{r}_n + h\mathbf{k}_3). \end{aligned}$$

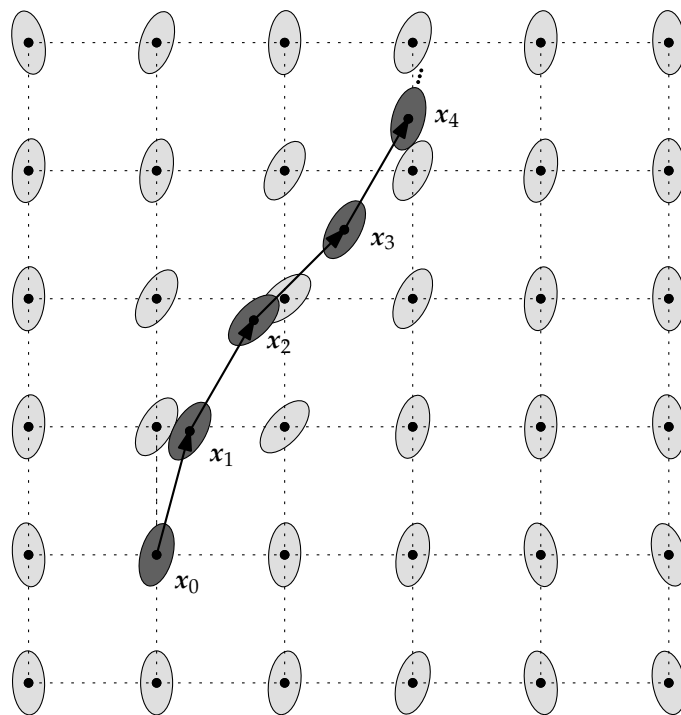
O método de Runge–Kutta não utiliza a derivada apenas no ponto anterior, mas sim em quatro pontos diferentes. Ele atinge, dessa forma, uma melhor estimativa do ponto  $\mathbf{r}_{n+1}$ .

É importante notar que, apesar de o passo do método de Runge–Kutta ser diferente do método de Euler, as demais observações da seção anterior continuam válidas.

### 2.3.3 Interpolação e regularização

A propagação de linhas, usando o método de Euler, Runge–Kutta ou qualquer outro, parte de um ponto inicial  $\mathbf{x}_0$  e sucessivamente encontra pontos  $\mathbf{x}_n$  que constituem uma trajetória. Os tensores estimados a partir das imagens de ressonância magnética, no entanto, estão presentes somente no centro dos voxels do volume. Ainda que se possa escolher o ponto  $\mathbf{x}_0$  para coincidir com o centro de algum voxel, não se pode prever onde estará localizado o ponto  $\mathbf{x}_n$ . Essa situação é ilustrada, em duas dimensões, pela Figura 2.8. O ponto  $\mathbf{x}_0$  está localizado sobre o centro de um pixel, mas os pontos  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ ,  $\mathbf{x}_3$  e  $\mathbf{x}_4$  não estão.

Por esse motivo, o método de propagação de linhas requer um mecanismo de interpolação ou aproximação que estime o valor do tensor de difusão em um ponto arbitrário  $\mathbf{x}$ . A interpolação (e aproximação) de tensores beneficia-se do fato de que seus elementos podem ser interpolados diretamente (Pajevic et al., 2002). Em outras palavras, encontrar o valor interpolado de um tensor resume-se à tarefa de interpolar cada um de seus



**Figura 2.8:** Pontos de uma trajetória sobre um campo de tensores bidimensional. Os pontos nem sempre coincidem com o centro dos voxels.

seis componentes escalares independentes. No restante dessa seção, portanto, a discussão ficará limitada à interpolação de escalares.

Um dos métodos de interpolação mais simples e conhecidos é a interpolação trilinear, em três dimensões, e seus correspondentes bidimensional e unidimensional, a interpolação bilinear e linear, respectivamente. A idéia desses métodos é obter o valor aproximado de um tensor em um ponto  $x$  a partir dos tensores vizinhos mais próximos na grade de tensores do volume. No caso bidimensional — mais simples de ser ilustrado aqui —, sejam  $(x, y)$  as coordenadas do ponto  $x$  e  $d_{xy}$  o valor da interpolação a ser calculado naquele ponto. Encontram-se então as coordenadas dos quatro pontos vizinhos mais próximos; sejam essas coordenadas  $(x_1, y_1)$ ,  $(x_2, y_1)$ ,  $(x_1, y_2)$  e  $(x_2, y_2)$ , e os valores nesses pontos  $d_{11}$ ,  $d_{21}$ ,  $d_{12}$  e  $d_{22}$ , conforme mostra a Figura 2.9. Desse modo, primeiro faz-se uma interpolação ao longo do eixo  $x$ , obtendo-se dois valores:

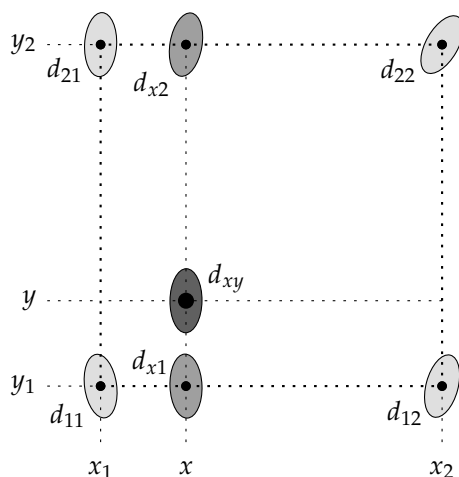
$$d_{x1} = d_{12} \frac{x - x_1}{x_2 - x_1} + d_{11} \frac{x_2 - x}{x_2 - x_1},$$

$$d_{x2} = d_{22} \frac{x - x_1}{x_2 - x_1} + d_{21} \frac{x_2 - x}{x_2 - x_1}.$$

Por fim, interpolam-se esses dois valores para se obter o valor final:

$$d_{xy} = d_{x2} \frac{y - y_1}{y_2 - y_1} + d_{x1} \frac{y_2 - y}{y_2 - y_1}.$$

A interpolação trilinear, em três dimensões, procede de maneira similar. Primeiro, encontram-se quatro valores interpolados para o eixo  $x$ , em seguida dois para o eixo  $y$  e por fim obtém-se o valor final da interpolação. Tanto na interpolação bilinear quanto trilinear, a ordem dos eixos não é relevante. Na interpolação trilinear, poder-se-ia começar com quatro valores para o eixo  $z$ , por exemplo, e depois proceder para o eixo  $x$  e finalmente o eixo  $y$ .



**Figura 2.9:** Ilustração da interpolação bilinear de um ponto  $(x, y)$ .

Uma revisão de vários métodos de interpolação aplicados a imagens médicas é dada por Lehmann et al. (1999). Os autores descrevem os métodos e apresentam gráficos de seu comportamento. Os diferentes métodos de interpolação são extensivamente comparados entre si.

Outros autores propuseram esquemas de interpolação ou aproximação mais específicos para a área de ressonância magnética de tensores de difusão. Pajevic et al. (2002) propuseram um esquema de aproximação baseado em B-splines que suaviza o campo de tensores, filtrando assim o ruído dos dados. O grau de suavização é controlado por um parâmetro  $\Delta$ . Em particular, quando  $\Delta = 1$ , a aproximação torna-se interpolação.

Mishra et al. (2006, 2007) introduziram um método de interpolação que suaviza o campo de tensores através de um filtro anisotrópico. O objetivo é obter um campo de tensores suave, contínuo e que preserve limites entre estruturas.

Também existem propostas para a regularização de dados. Poupon et al. (2000) definiram um meio para regularizar o campo de direção obtido a partir da principal direção de difusão dos tensores. A regularização é feita por meio da inclusão de conhecimento a priori acerca da baixa curvatura da maioria das fibras. Coulon et al. (2004) trabalham com a mesma idéia, mas também reorientam o campo de tensores a partir do campo de direção regularizado.



## Capítulo 3

# Unidades de processamento gráfico

As unidades de processamento gráfico, comumente designadas pela sigla GPU, são os componentes dos computadores modernos responsáveis pela manipulação e exibição de gráficos. Fisicamente, encontram-se diretamente acopladas às placas-mãe ou em placas de vídeo, que por sua vez podem ser conectadas às placas-mãe através de aberturas de expansão. Uma placa de vídeo atual pode ser vista na Figura 3.1.

As GPUs são freqüentemente contrapostas às CPUs, que são as unidades centrais de processamento. Nos computadores modernos, as CPUs constituem o processador em que a ampla maioria dos programas é executada, do sistema operacional aos programas de usuário.

Esse contraste entre CPU e GPU surgiu a partir do momento em que as GPUs ganharam a capacidade de executar programas arbitrários. Antes disso, CPUs e GPUs eram entidades muito diferentes para que uma dicotomia verdadeira pudesse surgir. Atualmente, porém, um programa pode ser escrito de forma que algumas de suas partes sejam executadas na CPU e outras na GPU, estabelecendo assim uma interação entre os dois tipos de processadores. Em alguns contextos, a CPU é chamada de hospedeiro e a GPU de dispositivo.

Essa interação entre CPU e GPU é justificável quando se leva em consideração a natureza das GPUs. Quando elas são utilizadas, um programa pode ser executado de forma altamente paralela, ou seja, um grande número de threads pode ser executados simultaneamente. Em muitos casos, esse ambiente paralelo oferecido pelas GPUs pode ser explorado de forma a aumentar o desempenho de certos programas.

O paralelismo não é, contudo, atributo exclusivo das GPUs. Até mesmo as CPUs atuais contam com mais de um núcleo de processamento. Existe, de fato, um grande otimismo atualmente com relação ao paralelismo: há quem diga que o “paralelismo é o futuro da computação” (Owens et al., 2008, p. 879). As GPUs expressam essa tendência muito claramente: a placa de vídeo GeForce GTX 275 da NVIDIA é capaz de executar 240 threads simultaneamente (NVIDIA, 2009d), enquanto as CPUs dos computadores pessoais ainda não são capazes de executar sequer 16 threads.

A tractografia por propagação de linhas é um dos problemas que obtêm ganhos expressivos de desempenho quando as GPUs são utilizadas. O objetivo deste capítulo é descrever as tecnologias e conceito sobre os quais baseiam-se os capítulos seguintes. A Seção 3.1 mostra como surgiram as GPUs e o que as levou a ter sua forma atual; a Seção 3.3 trata da tecnologia CUDA, utilizada na implementação do software de tractografia, que provê acesso ao poder computacional das GPUs; por fim, a Seção 3.4 descreve brevemente as perspectivas para



**Figura 3.1:** Placa de vídeo GeForce 9600 GT, da NVIDIA (NVIDIA, 2009f).

o futuro das GPUs.

### 3.1 As GPUs programáveis

As GPUs têm como função primordial a manipulação e a exibição de gráficos. Elas recebem da CPU requisições para a execução de tarefas gráficas e também interagem com o monitor acoplado para exibir os gráficos resultantes. A utilização de processadores feitos especialmente para a execução de rotinas gráficas traz duas vantagens significativas: a) as GPUs podem ser otimizadas para realizar cálculos gráficos mais rapidamente que um processador de uso geral e b) a CPU fica livre para efetuar outras computações.

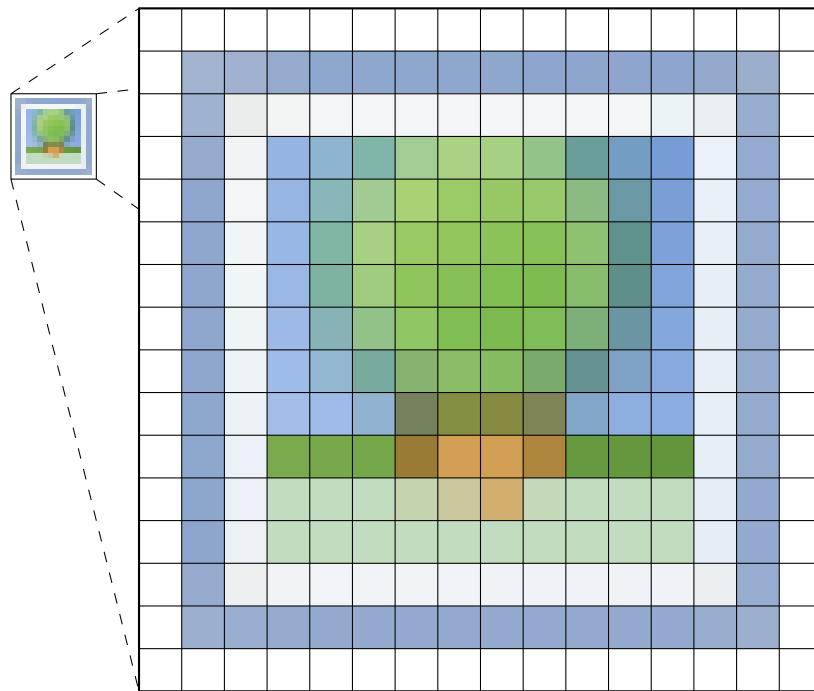
Os fabricantes de GPUs, portanto, sempre buscaram meios para obter processadores gráficos que explorassem muito bem a natureza do problema que eles resolvem. Uma das principais estratégias adotadas pelos fabricantes para tornar as GPUs mais velozes é tirar proveito do paralelismo inerente aos cálculos gráficos. As tarefas executadas pela GPU encaixam-se, de fato, na categoria de problemas embaraçosamente paralelos (Fernando e Kilgard, 2003), que nada mais são do que problemas que não exigem grande esforço para que uma implementação paralela seja produzida (Foster, 1995).

Considere-se, por exemplo, o momento em que uma GPU precisa gerar um quadro (frame) em um jogo de corrida. Essa tarefa deve ser executada rapidamente, pois o jogo precisa manter uma taxa de quadros por segundo interativa. Cada quadro é, essencialmente, uma imagem (veja a Figura 3.2). O valor de cor para cada pixel dessa imagem precisa ser calculado para que o resultado final seja obtido. Se um monitor com uma resolução mediana estiver sendo utilizado, o número total de pixels em cada imagem é

$$1.024 \times 768 = 786.432.$$

Note, no entanto, que a cor de cada um dos 786.432 pixels pode ser calculada de maneira independente das cores dos outros pixels. O resultado do cálculo da cor de um pixel não tem influência nenhuma sobre o resultado do cálculo dos outros, inclusive dos adjacentes.

Essa independência entre os pixels permite que as GPUs computem o valor deles em paralelo, ou seja, em um dado momento uma GPU pode estar calculando a cor de vários pixels simultaneamente. No caso do exemplo anterior, uma GPU poderia teoricamente calcular a cor de todos os 786.432 pixels ao mesmo tempo.



**Figura 3.2:** Ilustração da independência entre os pixels de uma imagem. Quando a GPU precisa calcular uma imagem para exibi-la na tela, cada um dos pixels pode ser computado de maneira independente dos demais.

Ao longo dos últimos dez ou quinze anos, enquanto os fabricantes de CPU preocupavam-se com a execução de programas genéricos, que nem sempre podem ser resolvidos facilmente de maneira paralela, os fabricantes de GPU podiam investir seus transistores para aumentar o paralelismo do processamento efetuado pelas GPUs.

Inicialmente, a CPU tinha um acesso indireto aos recursos computacionais das GPUs. Os programas que eram executados na CPU podiam solicitar, através de interfaces de programação de aplicativos (APIs), que determinadas tarefas fossem executadas na GPU. O conjunto de tarefas passíveis de serem executadas na GPU era fixo e era voltado exclusivamente a cálculos gráficos. Dizia-se que as GPUs eram altamente configuráveis, mas não programáveis (Fernando e Kilgard, 2003), pois as tarefas que elas executavam podiam ter seu comportamento alterado de muitas maneiras, mas não era possível a execução de programas arbitrários na GPU.

Para que os aplicativos gráficos pudessem ter um controle mais amplo e preciso sobre o processamento gráfico efetuado pelas GPUs, os fabricantes começaram a permitir que pequenos programas fossem executados em estágios específicos da seqüência de operações que as GPUs realizavam. Esses programas são geralmente chamados de núcleos. Assim, o comportamento padrão da GPU podia ser modificado de maneira arbitrária, o que permitiu aos aplicativos gráficos produzirem efeitos visuais que não eram antes possíveis com as configurações pré-programadas das GPUs.

Para que não fosse necessário programar as GPUs por meio da linguagem assembly, a Microsoft e a NVIDIA criaram, em parceria, a linguagem de programação Cg (ou C for Graphics) (Fernando e Kilgard, 2003). Trata-se de uma linguagem de alto nível, inspirada no C, com extensões específicas para o processamento gráfico. Um programa escrito em Cg é enviado à GPU para alterar o comportamento padrão de uma etapa específica de processamento.

```
samplerRECT texture1: TEXUNIT0;  
samplerRECT texture2: TEXUNIT1;  
  
void Sum(in float2 position: TEXCOORD0,  
         out float4 result: COLOR0)  
{  
    float4 a = texRECT(texture1, position);  
    float4 b = texRECT(texture2, position);  
    result = (a + b) / 2;  
}
```

---

**Figura 3.3:** Exemplo de programa escrito em Cg. Este programa calcula a média entre duas imagens.

Um programa escrito em Cg não precisa, necessariamente, ter seus resultados exibidos na tela. A seqüência padrão de processamento das GPUs pode ser subvertida de forma que os resultados dos cálculos sejam lidos pela CPU ou então processados novamente pela GPU. A Figura 3.3 contém um exemplo de programa escrito em Cg que pode ser utilizado para calcular a média de duas imagens; o resultado pode ou não ser exibido na tela, de acordo com a configuração que a CPU estabelecer.

## 3.2 Programação de propósito geral em GPUs

Por um certo tempo, as tecnologias pertinentes à programação das GPUs foram, naturalmente, voltadas aos programas gráficos (Thompson et al., 2002). Contudo, mesmo com um viés gráfico tão forte, logo percebeu-se que o poder de computação das GPUs podia ser usado para fins mais gerais. Cunhou-se, por esse motivo, a sigla GPGPU (General Purpose computation on GPUs), que passou a designar de modo geral as tecnologias e os esforços centrados na programação de aplicações não-gráficas em GPUs.

Por poderem executar programas arbitrários, as GPUs programáveis foram desde cedo consideradas co-processadores (Harris et al., 2002; Macedonia, 2003), de maneira análoga aos co-processadores de ponto flutuante que haviam existido anteriormente (Miljanic e Kaeli, 1992). Os aplicativos podiam enviar programas às GPUs e obter ganhos de desempenho na execução de certas tarefas.

Ainda que o poder de processamento das GPUs estivesse se tornando cada vez mais acessível à CPU, os programadores tinham de se submeter às idiossincrasias oriundas do passado gráfico das GPUs. Apesar de o conteúdo dos programas executados pela GPU ser arbitrário, eles precisavam se encaixar no modelo de programação das GPUs, o que tornava a programação das GPUs uma tarefa árdua (Harris et al., 2002) e “dolorosa” (Macedonia, 2003, p. 108).

O programa em Cg da Figura 3.3, por exemplo, precisava de uma estrutura auxiliar relativamente grande e complexa para ser executado. Era preciso fazer muitas chamadas a interfaces de programação para enviar e receber dados da GPU e também para executar o programa. Mesmo que um aplicativo desejasse realizar cálculos não-gráficos, ele precisava lidar com toda a complexidade das interfaces de programação gráficas. Owens resume bem a situação quando diz que “uma das dificuldades históricas na programação de aplicações GPGPU tem sido que, apesar de as tarefas de propósito geral não terem nenhuma relação com gráficos, as aplicações tinham de ser programadas usando interfaces de programação gráficas.” (Owens et al., 2008, p. 884)

No contexto da pesquisa que originou essa dissertação, uma implementação da tractografia foi feita com base na linguagem Cg (Mittmann et al., 2008). Para ilustrar o tipo de dificuldade que se encontrava ao utilizar-se essa linguagem e as tecnologias associadas, considerem-se alguns dos problemas enfrentados:

- Os pontos tridimensionais das trajetórias correspondentes às fibras eram armazenados em imagens. Cada ponto era armazenado em um pixel e tinha suas coordenadas  $x$ ,  $y$  e  $z$  mapeadas para os canais R, G e B do pixel. Os tensores também eram armazenados em imagens e cada um ocupava dois pixels, pois a matriz simétrica do tensor possui seis elementos únicos.
- Para que a computação da tractografia fosse disparada, o aplicativo solicitava que um quadrado fosse desenhado em uma janela invisível. Esse era o meio de solicitar que a GPU executasse a tractografia. Normalmente, aplicações gráficas só ordenariam que um quadrado fosse desenhado se quisessem um quadrado na tela.
- Por causa da complexidade inerente ao algoritmo da tractografia, o compilador freqüentemente informava que o número máximo de registradores havia sido atingido ou que a compilação resultava em um número de instruções maior que o suportado.

Mesmo tendo que adaptar o problema da tractografia a um ambiente gráfico e inóspito, foi possível obter bons ganhos de desempenho: em um caso, a tractografia podia ser executada 40 vezes mais rápido na GPU do que na CPU (Mittmann et al., 2008).

Por causa das dificuldades de se programar para a GPU, iniciativas como o ambiente Brook (Buck et al., 2004) foram criadas. Brook permitia que a GPU fosse programada sem a necessidade de lidar com as interfaces de programação gráficas e tornava a implementação de programação de propósito geral mais simples. Uma das limitações de Brook é que os programas escritos nele são limitados pela linguagem Cg.

A idéia por trás de Brook e das demais novas tecnologias é fazer com que as aplicações de propósito geral possam ser programadas utilizando um ambiente efetivamente genérico, e não um ambiente que força os programadores a se expressarem em termos gráficos independentemente da tarefa a ser realizada.

### 3.3 A tecnologia CUDA

A tecnologia CUDA foi introduzida pela NVIDIA no início de 2007 como “uma nova abordagem para a computação, em que centenas de processadores em um único chip simultaneamente se comunicam e cooperam para resolver problemas computacionais complexos.” (NVIDIA, 2007b). O nome CUDA era originalmente uma sigla — arquitetura de dispositivo unificada para computação —, mas aparentemente a NVIDIA não usa mais o termo como uma sigla, a julgar pela maneira como as últimas versões do manual de programação usam o termo (NVIDIA, 2008a,b).

A Figura 3.4 mostra um programa de limiarização de imagens escrito com a tecnologia CUDA. O programa aplica a limiarização a uma imagem em tons de cinza, paralelizando a operação para todos os píxeis da imagem. Esse pequeno programa incorpora muitos dos aspectos relevantes de CUDA e será usado ao longo das seções seguintes como exemplo.

Devido à complexidade do seu objetivo, a tecnologia CUDA é composta por componentes que abrangem desde uma arquitetura de hardware até um compilador, passando por drivers e APIs. As ferramentas de CUDA, como o compilador, são apresentados na Seção 3.3.1; o modelo de programação, incluídas aí as APIs e as extensões a C e C++, é o assunto da Seção 3.3.2; o modelo de threads e o modelo de memória são vistos, respectivamente, nas Seções 3.3.3 e 3.3.4; por fim, a Seção 3.3.5 discorre sobre a arquitetura de hardware de CUDA.

#### 3.3.1 Ferramentas

As ferramentas e acessórios necessários ao desenvolvimento de programas que utilizam CUDA estão contidos no toolkit CUDA. Ele contém o compilador NVCC e também cabeçalhos e bibliotecas, além de documentação. A utilização do kit não depende nem da presença de uma placa de vídeo em particular, nem da instalação de um driver. Em particular, o compilador NVCC oferece uma opção para a execução do código em modo de emulação, de maneira que uma placa de vídeo com suporte a CUDA e o driver respectivo só são necessários para a execução de programas diretamente no dispositivo, mas não para seu desenvolvimento.

O compilador NVCC não tem apenas a função de compilar código a ser executado no dispositivo. O programa de exemplo da Figura 3.4, por exemplo, contém a função `Limiarizar`, que é executada no dispositivo, e a função `main`, que é executada no hospedeiro. O NVCC, ao deparar-se com um código como esse, compila as funções que devem ser executadas no dispositivo e chama o compilador do sistema (tipicamente o GCC no Linux e o CL no Windows) para compilar as funções do hospedeiro.

#### 3.3.2 Programação com CUDA

As funções escritas para o dispositivo devem utilizar a linguagem C, ainda que com algumas particularidades devem ser observadas, já que o código-fonte como um todo é processado de acordo com as regras sintáticas do C++ (NVIDIA, 2008b). Além disso, a alocação dinâmica de memória, característica fundamental de C, não pode ser feita em funções de dispositivo. As funções para o hospedeiro podem usar C++. Elas são compiladas pelo compilador do sistema, apesar de o NVCC ser responsável por interpretar algumas extensões.

A tecnologia CUDA introduz algumas extensões para tornar mais agradável a programação de GPUs. As extensões são de quatro tipos (NVIDIA, 2008b):

- Qualificadores que indicam se a função é executada no hospedeiro ou no dispositivo;
- Uma diretiva utilizada para o hospedeiro invocar funções no dispositivo;
- Quatro variáveis que funções no dispositivo podem usar para obter informações de identificação sobre o thread em execução. São vistas na Seção 3.3.3;
- Qualificadores que especificam o local de armazenamento de variáveis. São vistos na Seção 3.3.4.

Os qualificadores de função são três e especificam onde uma função é executada (hospedeiro ou dispositivo) e de onde é possível chamá-la. Eles são:

---

```

1  #include <math.h>
2
3  void LerImagem(float **imagem, unsigned *largura, unsigned *altura);
4  void GravarImagem(float *imagem, unsigned largura, unsigned altura);
5
6  ❶ __global__ void Limiarizar(float *imagem, float limiar, unsigned tamanho)
7  {
8  ❷  int i =
9      blockIdx.x * (blockDim.x * blockDim.y) +
10     threadIdx.y * blockDim.x +
11     threadIdx.x;
12
13  ❸  if (i >= tamanho)
14     return;
15
16     if (imagem[i] >= limiar)
17         imagem[i] = 1.0;
18     else
19         imagem[i] = 0.0;
20 }
21
22 int main()
23 {
24     float *imagem_cpu;
25     unsigned largura, altura;
26     LerImagem(&imagem_cpu, &largura, &altura);
27
28     float *imagem_gpu;
29     unsigned tamanho = largura * altura;
30  ❹  cudaMalloc((void **)&imagem_gpu, tamanho * sizeof(float));
31  ❺  cudaMemcpy(imagem_gpu, imagem_cpu, tamanho * sizeof(float),
32             cudaMemcpyHostToDevice);
33
34     dim3 block(16, 16);
35     dim3 grid(floor(tamanho / 256));
36  ❻  Limiarizar<<<grid, block>>>(imagem_gpu, 0.8, tamanho);
37
38  ❼  cudaMemcpy(imagem_cpu, imagem_gpu, tamanho * sizeof(float),
39             cudaMemcpyDeviceToHost);
40
41     GravarImagem(imagem_cpu, largura, altura);
42 }

```

---

**Figura 3.4:** Programa de exemplo escrito utilizando a tecnologia CUDA. O núcleo definido em ❶ limiariza uma imagem em tons de cinza.

- `__global__`: indica que a função em questão deve ser executada no dispositivo e pode ser chamada apenas pelo hospedeiro. Essa diretiva é usada para marcar os núcleos que podem ser chamados pelo hospedeiro, como em ❶ no programa de exemplo.
- `__device__`: a função será executada no dispositivo e pode ser chamada somente pelo dispositivo. As funções auxiliares dos núcleos são marcadas com essa diretiva.
- `__host__`: a função é executada no hospedeiro e só pode ser chamada pelo hospedeiro. Esse é o padrão caso nenhum qualificador seja utilizado.

É importante notar que a diretiva `__host__` pode ser usada juntamente com a diretiva `__device__`. Nesse caso, a função será compilada tanto para a CPU quanto para a GPU. Nota-se, também, que em nenhum caso uma função de GPU pode chamar outra de CPU.

Outra extensão relevante à linguagem C é usada para chamar as funções do tipo `__global__`. Além do nome da função e dos argumentos, essas funções são chamadas com especificações adicionais delimitadas por `<<<` e `>>>`, como em ❷ no programa de exemplo.

Os ponteiros passados como argumentos aos núcleos apontam para a memória do dispositivo, e não para a memória do hospedeiro. É possível alocar memória no dispositivo através da função `cudaMalloc` e transferir dados da memória do hospedeiro para a do dispositivo e vice-versa através da função `cudaMemcpy`.

No programa de exemplo, memória suficiente para armazenar a imagem a ser limiarizada é alocada no dispositivo em ❸. Para preencher a memória já alocada com a imagem original, em ❹ é chamada a função `cudaMemcpy` com o argumento `cudaMemcpyHostToDevice`, que indica o sentido da transferência de dados. Uma vez que o núcleo foi executado, a imagem já processada é copiada em ❺ de volta à memória do hospedeiro.

#### 3.3.3 Modelo de threads

Quando um núcleo é chamado, o hospedeiro deve especificar não apenas o número de threads a serem criados, mas também a sua disposição em uma grade de blocos, como mostra a Figura 3.5. A grade é uma estrutura bidimensional de blocos, cada qual por sua vez uma estrutura tridimensional de threads (NVIDIA, 2008b). Todos blocos têm o mesmo tamanho. As dimensões da grade e do bloco são especificadas quando um núcleo é chamado. No programa de exemplo, o núcleo é invocado em ❻ com as dimensões já calculadas da grade e do bloco.

O número máximo de threads em um bloco é 512, ainda que frequentemente tamanhos menores de bloco tenham de ser utilizados, dependendo da quantidade de memória requerida pelo núcleo. Como o tamanho máximo do bloco é relativamente pequeno, em geral o tamanho da grade é que varia conforme a massa de dados que o núcleo processa. No programa de exemplo, o tamanho do bloco está fixo em  $16 \times 16 = 256$  threads, e o tamanho da grade varia conforme o tamanho da imagem. Para uma imagem de  $1.024 \times 768 = 786.432$  píxeis, a grade teria 3.072 blocos.

A função que está sendo executada no dispositivo pode acessar duas variáveis para se instruir a respeito do tamanho da grade e do bloco (`gridDim` e `blockDim`) e outras duas que informam a localização do thread



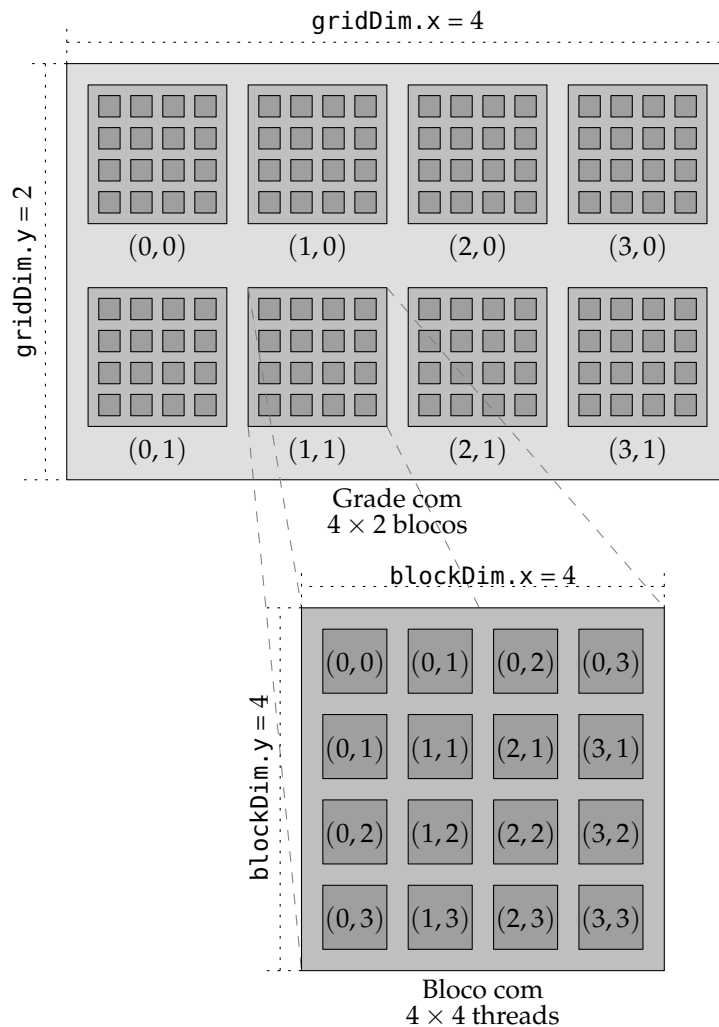


Figura 3.5: Estrutura de grade e blocos de CUDA.

dentro da grade (`blockIdx`) e dentro do bloco (`threadIdx`). Essas variáveis conseguem dar uma resposta à questão existencial dos threads: indicam-lhes qual elemento de dados eles devem processar.

Nota-se, portanto, que não existe uma ligação formal entre os elementos a serem processados pelo núcleo e sua posição dentro da estrutura de grade e blocos. Todos os threads nascem iguais e podem, em princípio, processar os elementos que bem entenderem. No programa de exemplo, cada thread calcula em ❷ o índice do píxel que ele processará. Como é possível que a estrutura de grade e blocos contenha mais threads do que a imagem tem píxeis, os threads verificam em ❸ se devem de fato levar adiante sua empreitada.

### 3.3.4 Modelo de memória

Fisicamente, existem três memórias acessíveis a um thread na arquitetura CUDA: os registradores, a memória compartilhada e a memória do dispositivo (NVIDIA, 2008b). Os registradores em geral não requerem ciclos adicionais para serem acessados. A memória compartilhada é comum aos threads de um bloco e é, em muitas circunstâncias, tão rápida quanto os registradores. A memória do dispositivo é a memória principal, significativamente mais lenta que as demais.

Do ponto de vista do programador, a memória do dispositivo recebe nomes diferentes conforme a situação:

- Memória global: leitura e escrita, sem cache, de propósito geral;
- Memória local: leitura e escrita, sem cache, utilizada apenas pelo compilador;
- Memória de constante: somente leitura, com cache;
- Memória de textura: somente leitura, com cache, otimizada para localidade espacial 2D.

Os registradores e a memória local são gerenciados pelo compilador, que decide armazenar as variáveis em uma memória ou outra. Em geral, ele decide armazenar variáveis na memória local quando elas ocupariam uma porção muito grande dos registradores (NVIDIA, 2008b). As demais memórias (ou seja, a compartilhada, a global, a de constante e a de textura) são gerenciadas pelo programador, que deve escolher com cuidado onde armazenar suas variáveis, sob pena de tornar seu programa lento.

Os qualificadores de variável são usados para definir variáveis que são armazenadas na memória do dispositivo. Eles são:

- `__device__`: a variável é armazenada na memória global do dispositivo;
- `__constant__`: a variável é armazenada na memória de constantes do dispositivo;
- `__shared__`: a variável é armazenada na memória que é compartilhada por um bloco de threads.

Para utilizar a memória de textura, um artifício mais complexo precisa ser utilizado. O código do hospedeiro deve associar uma região de memória a uma textura através de chamada à API, e o código do dispositivo deve utilizar funções específicas para acessá-la.

#### 3.3.5 Arquitetura de hardware

Os dispositivos da NVIDIA que implementam a arquitetura CUDA dispõem de uma série de multiprocessadores, cada um composto de oito núcleos de processamento. O número exato de multiprocessadores varia bastante de uma placa de vídeo para outra e é uma das principais características que influenciam o desempenho. Quando um núcleo de software é invocado pelo hospedeiro, cada bloco de threads é executado por um multiprocessador. Conforme blocos de threads terminam de ser processados, novos blocos são alocados aos multiprocessadores.

Os multiprocessadores dividem o bloco de threads que lhes foi destinado em grupos de 32, chamados de warps. Os threads em um warp são executados de acordo com uma arquitetura chamada de SIMT (single-instruction, multiple-thread): as instruções dos threads são executadas em paralelo, desde que todos eles sigam o mesmo caminho de execução. Caso os threads sigam desvios diferentes, cada ramo é executado individualmente.

A arquitetura SIMT é similar à mais tradicional SIMD (single-instruction, multiple-data), já que em ambas as arquiteturas as instruções são executadas em paralelo para o conjunto de dados. A arquitetura SIMT, entretanto, não é tão rígida, pois permite que os threads sigam caminhos distintos, ainda que o desempenho seja melhor quando isso não ocorre.

Os multiprocessadores da arquitetura SIMT são capazes de criar, gerenciar e executar threads sem gastar nenhum ciclo adicional para isso (NVIDIA, 2008b). Essa capacidade permite que CUDA seja utilizada para problemas com paralelismo de granularidade extremamente fina, como é o caso de imagens, em que cada thread processa um único píxel.

## 3.4 O futuro da programação em GPUS

A NVIDIA tem conseguido um sucesso comercial grande no mercado de GPUS programáveis, especialmente por ter largado na frente com a tecnologia CUDA (Santo e Adeo, 2009). Essa tecnologia, porém, é proprietária da NVIDIA, o que praticamente elimina a possibilidade de algum outro fabricante implementá-la.

A proposta da AMD/ATI, chamada de Stream, é “um conjunto de tecnologias avançadas de hardware e software que permite aos processadores gráficos (GPUS) da AMD, trabalhando em concerto com o processador central (CPU) do sistema, acelerar muitas aplicações de formas que vão além da gráfica.” (AMD, 2009b). A tecnologia Stream, entretanto, não tem se mostrado um competidor à altura de CUDA (Santo e Adeo, 2009)

A Intel tem trabalhado no desenvolvimento de sua própria arquitetura paralela, chamada de Larrabee (Intel, 2009). Apesar de ainda não ter sido lançada, essa arquitetura é alardeada como séria competição à NVIDIA (Santo e Adeo, 2009). Uma das vantagens que Larrabee terá é a possibilidade de se utilizar a linguagem C++.

Uma outra proposta recente é a OpenCL. Trata-se do “primeiro padrão aberto e livre de royalties para a programação multiplataforma e paralela dos processadores modernos encontrados em computadores pessoais, servidores e dispositivos handheld/embutidos.” (Khronos Group, 2009). Muitos fabricantes participam do desenvolvimento de OpenCL, inclusive nomes de peso como AMD, NVIDIA e Intel.

É importante ressaltar que OpenCL vai além de tecnologias como CUDA, que são específicas para GPUS. Com efeito, OpenCL é anunciado como “o padrão aberto para programação paralela *heterogênea*” (grifo ausente no original) (Munshi, 2009, p. 38), pois um de seus objetivos é prover acesso a todos os recursos computacionais de um sistema, o que inclui tanto GPUS quanto CPUs. A linguagem de programação em si é baseada em C.

A NVIDIA já anunciou drivers para OpenCL (NVIDIA, 2009g) e a AMD já lançou uma plataforma de desenvolvimento para OpenCL (AMD, 2009a). Não está claro neste momento se a Intel também implementará o OpenCL.

## Capítulo 4

# Tractografia em GPUS

Foram escritas duas implementações do mesmo método de tractografia: uma em GPU e outra em CPU. O objetivo desta última é servir de referência para comparações entre GPU e CPU. Ambas implementações foram agrupadas em uma biblioteca de tractografia. Dois programas de tractografia foram construídos com base nessa biblioteca: o programa de tractografia em lote e a ferramenta interativa de tractografia. O objetivo do programa de lote é processar um único volume DT-MRI e gravar os resultados tractográficos, enquanto o objetivo da ferramenta interativa é permitir que o usuário explore graficamente os resultados tractográficos.

Os experimentos deste capítulo utilizam o programa de lote para obter seus resultados. A ferramenta gráfica e os experimentos que a utilizam são apresentados no capítulo seguinte. Os equipamentos e os conjuntos de dados dos experimentos deste e do próximo capítulo estão descritos no Apêndice A.

### 4.1 Método de tractografia

Dentre as várias estratégias para a localização dos tratos neurais a partir de exames de ressonância magnética, este trabalho emprega o método de propagação de linhas. Esse método, entretanto, apresenta variações em alguns de seus aspectos, como o esquema de interpolação utilizado ou o conjunto de critérios de parada. Os detalhes desses aspectos do método de propagação de linhas utilizado pelas implementações descritas mais adiante são apresentados a seguir.

Como visto no Capítulo 2, o método de propagação de linhas recebe como entradas um campo discreto de tensores, um conjunto de pontos semente e uma quantidade de parâmetros. Como resultado, o método produz um conjunto de trajetórias, que são compostas por vários segmentos de reta. Para a geração do resultado, os pontos semente são individualmente examinados e a partir deles as trajetórias são propagadas.

As trajetórias são calculadas ponto o ponto, cada um sendo calculado a partir do anterior. Neste trabalho, os pontos das trajetórias são calculados utilizando o método de Runge–Kutta de quarta ordem, com  $h = 0,5$ . Ao invés de armazenar cada ponto encontrado pelo algoritmo de integração, os pontos só são armazenados após uma distância mínima de 1 mm ter sido percorrida com relação ao ponto anterior.

O algoritmo de integração numérica precisa estimar o valor de tensores em pontos arbitrários do espaço, isto é, não apenas no centro dos voxels do volume original. A interpolação trilinear é adotada para estimar

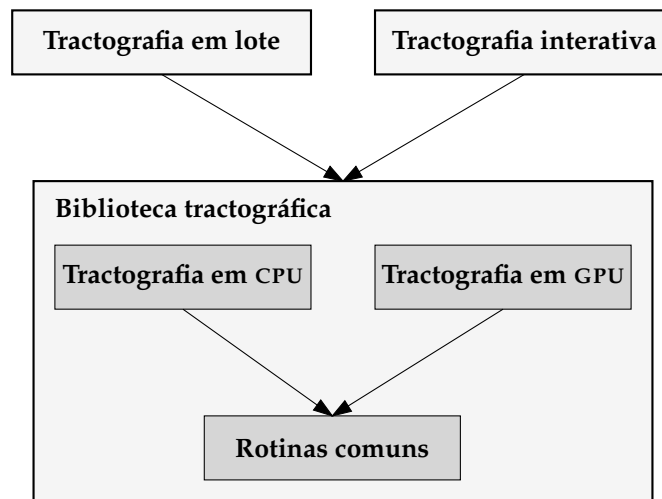


Figura 4.1: Estrutura interna da biblioteca tractográfica e sua relação com programas externos.

tensores em qualquer ponto do volume. Trata-se de um algoritmo relativamente simples e eficiente, apesar de não ser tão sofisticado quanto outros, como o esquema de Pajevic et al. (2002).

Os critérios de parada do método de tractografia são os seguintes:

- Valor de anisotropia fracional menor que o mínimo;
- Valor de difusividade média menor que o mínimo;
- Ângulo entre segmentos de reta da trajetória maior que o máximo.

Além dos critérios de parada, é imposto um limite máximo de 150 pontos para cada trajetória. Averiguou-se experimentalmente que esse é um bom valor para o tamanho máximo de cada trajetória, já que poucas vão além desse limite.

## 4.2 A biblioteca tractográfica

Uma biblioteca tractográfica foi escrita para concentrar as implementações de tractografia e as rotinas acessórias em um único artefato. Dessa forma, não apenas as implementações de tractografia beneficiam-se de rotinas comuns, mas também facilita-se a integração com programas externos, como é o caso do programa de tractografia em lote deste capítulo e da estação radiológica descrita no Capítulo 5. A relação entre os componentes internos da biblioteca e também a relação desta com os programas externos estão ilustradas na Figura 4.1

A linguagem C++ foi utilizada para a escrita da biblioteca tractográfica. Trata-se de uma linguagem que permite a criação de programas otimizados e para a qual muitas outras bibliotecas existem. O uso de C++ também facilita a integração com a tecnologia CUDA, utilizada pela implementação da tractografia em GPU.

Entre as rotinas comuns disponíveis na biblioteca tractográfica estão a leitura de arquivos DICOM e NIFTI, que armazenam os exames de DT-MRI; a estimação de tensores a partir dos dados de DT-MRI; e a gravação de resultados tractográficos em arquivos.

A biblioteca tractográfica oferece aos programas externos uma interface abstrata para as implementações de tractografia, ou seja, os programas externos podem escolher qual implementação querem utilizar sem que para isso tenham de ser alterados.

De particular relevância é o fato de que, através dessa interface abstrata, os programas podem solicitar a inclusão ou não de cores no resultado. A inclusão de cores implica um custo ligeiramente maior de computação, pois, para cada ponto de cada trajetória, uma cor é calculada e armazenada. As trajetórias assim coloridas geram visualizações mais informativas, porém são desnecessárias se o programa em questão não pretende mostrar os resultados na tela.

Durante o desenvolvimento da biblioteca tractográfica, foi possível constatar que a maneira como os resultados tractográficos são armazenados na memória RAM da CPU tem um impacto significativo no tempo total de execução da tractografia, tanto na implementação em CPU quanto na implementação em GPU. Esse impacto faz-se sentir também na implementação em GPU porque os resultados, apesar de serem gerados pela GPU, precisam ser armazenados na RAM da CPU antes de serem exibidos na tela ou gravados em disco. Por causa desse impacto, a biblioteca de tractografia provê um meio único para o armazenamento de resultados tractográficos na memória RAM. Tanto a implementação em CPU quanto a em GPU utilizam esse mesmo meio para armazenar os resultados que produzem, garantindo assim que, qualquer que seja o impacto do armazenamento no tempo de execução, ele é o mesmo para ambas implementações.

### 4.3 Implementação de referência em CPU

A implementação de tractografia em CPU é considerada como a implementação de referência, pois é com ela que a implementação em GPU é comparada. A implementação em CPU foi nomeada referencial, e não a em GPU, porque a CPU é a plataforma tradicionalmente usada para executar a totalidade dos programas em um computador pessoal.

Estruturalmente, a tractografia em CPU foi implementada de maneira simples. Ela pode ser descrita como uma série de laços aninhados:

1. Processar cada um dos pontos semente;
2. Calcular cada uma das duas trajetórias a partir do ponto semente;
3. Determinar cada ponto da trajetória, percorrendo ao menos a distância de 1 mm.

Fisicamente, é evidente, esses laços estão convenientemente distribuídos em diferentes funções e classes.

Quando um ponto semente começa a ser processado, ele é antes validado de acordo com os critérios de parada. Se o ponto semente não satisfizer nenhum critério de parada, as duas trajetórias (correspondentes aos dois sentidos da difusão) são então calculadas.

A implementação de referência foi escrita com especial cuidado no que se refere à otimização do código. Esse cuidado é necessário para que a comparação entre CPU e GPU faça algum sentido. Por esse motivo, a implementação em CPU foi zelosamente elaborada para obter um desempenho otimizado. Por exemplo, o uso de polimorfismo foi restrito aos casos essenciais, para evitar chamadas de métodos virtuais; funções críticas

foram definidas como inline sempre que apropriado; e o compilador foi instruído a aplicar as otimizações cabíveis.

Um dos aspectos críticos de qualquer implementação de tractografia é o modo como os tensores são interpolados e como os autovalores e autovetores correspondentes são calculados. A implementação de referência acessa os tensores e os interpola de maneira direta, para garantir a eficiência da operação. Os autovalores e os autovetores são calculados pelas rotinas de matrizes simétricas reais da biblioteca GSL.

Um outro cuidado foi tomado durante a escrita da implementação de referência para garantir a justiça na comparação com a implementação em GPU: a utilização de precisão numérica simples, isto é, de 32 bits. Como a maior parte das GPUs atuais e, em especial, todas as GPUs utilizadas nos experimentos, não oferecem precisão maior do que a simples, a implementação em CPU utiliza o tipo numérico correspondente à precisão simples. Se a CPU utilizasse tipos numéricos mais precisos, geraria resultados melhores, porém levaria mais tempo para fazê-lo, o que comprometeria a comparação de tempos de execução entre CPU e GPU.

É importante observar que, qualquer que seja a precisão numérica adotada, sempre haverá um erro inerente presente nos resultados. Ademais, a diferença entre os resultados produzidos por uma implementação que utiliza precisão simples e outra que utiliza precisão dupla é pequena (Mittmann et al., 2008).

### 4.4 Implementação em GPU

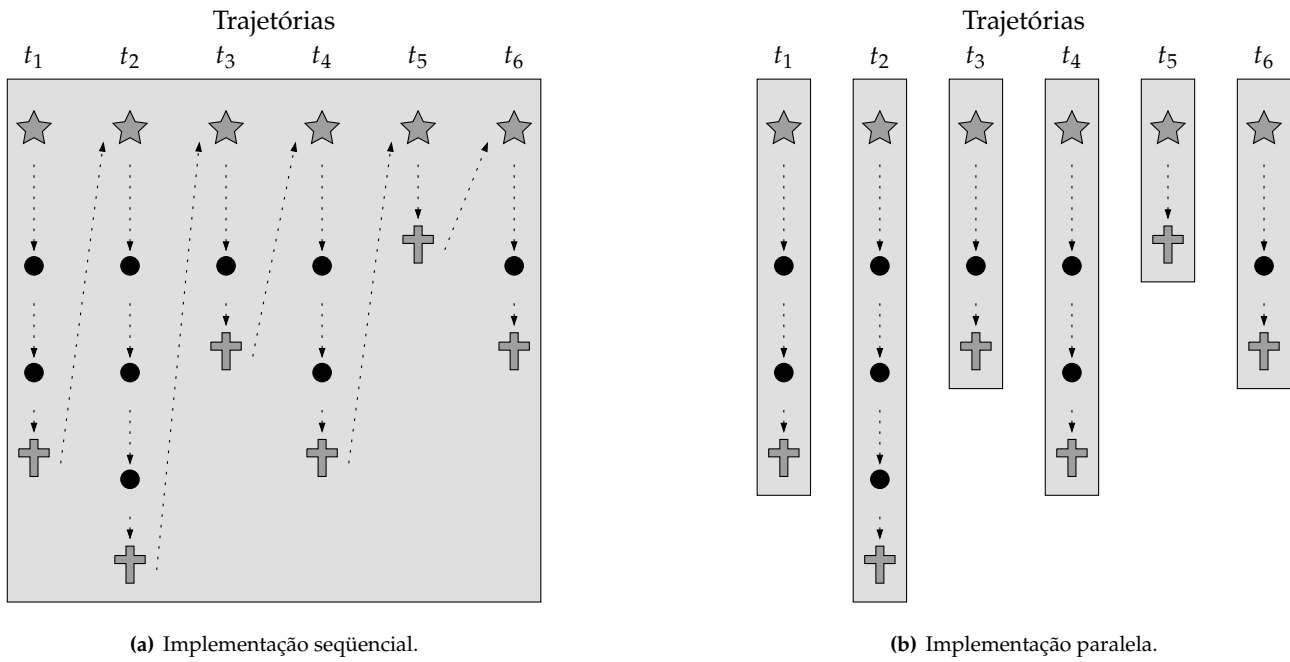
A implementação da tractografia em GPU é similar à implementação de referência em muitos aspectos, mas difere substancialmente no que se refere à sua arquitetura e na maneira como os dados são manipulados. Os pormenores da interpolação de tensores e da integração numérica são os mesmos, mas, devido à natureza paralela das GPUs e ao modo indireto de acesso aos recursos computacionais delas, os dados são agrupados de forma diferente, requerem estruturas de dados adicionais e necessitam de um sistema mais complexo para gerenciar a execução da tractografia.

#### 4.4.1 O paralelismo

A tractografia através do método de propagação de linhas, utilizado neste trabalho, tem como elemento básico de paralelismo as trajetórias. Os pontos de uma trajetória não dependem dos pontos de outra trajetória, ou seja, a computação de uma trajetória não depende da computação das outras. Uma implementação paralela, portanto, pode calcular mais de uma trajetória ao mesmo tempo e obter um ganho de desempenho.

A maneira como implementações seqüenciais e implementações paralelas de tractografia calculam as trajetórias pode ser visualizada na Figura 4.2. Uma implementação seqüencial explora os pontos semente e as trajetórias correspondentes um por vez. Assim que uma trajetória é finalizada, a implementação seqüencial explora a próxima. Apesar de não haver dependência entre as trajetórias, o cálculo de uma trajetória só pode começar quando a anterior já tiver sido calculada.

Uma implementação paralela, por outro lado, pode criar vários threads e dessa forma computar mais de uma trajetória ao mesmo tempo, cada thread sendo responsável pela computação de uma única trajetória. Neste trabalho, esses threads são executados no contexto de GPUs, mas também é possível que eles executem em um aglomerado de computadores ou em outras arquiteturas paralelas (Mittmann et al., 2009).



**Figura 4.2:** Em uma implementação sequencial, (a) apenas um thread calcula todas as trajetórias. Em uma implementação paralela, (b) vários threads são criados, cada um calculando uma trajetória.  $\star$  = ponto semente,  $\bullet$  = ponto de trajetória,  $\dagger$  = fim de trajetória.

O número exato de trajetórias que podem ser calculados simultaneamente por uma implementação paralela depende de qual equipamento está sendo utilizado. No caso das GPUs, é comum que elas consigam calcular dezenas de trajetórias ao mesmo tempo.

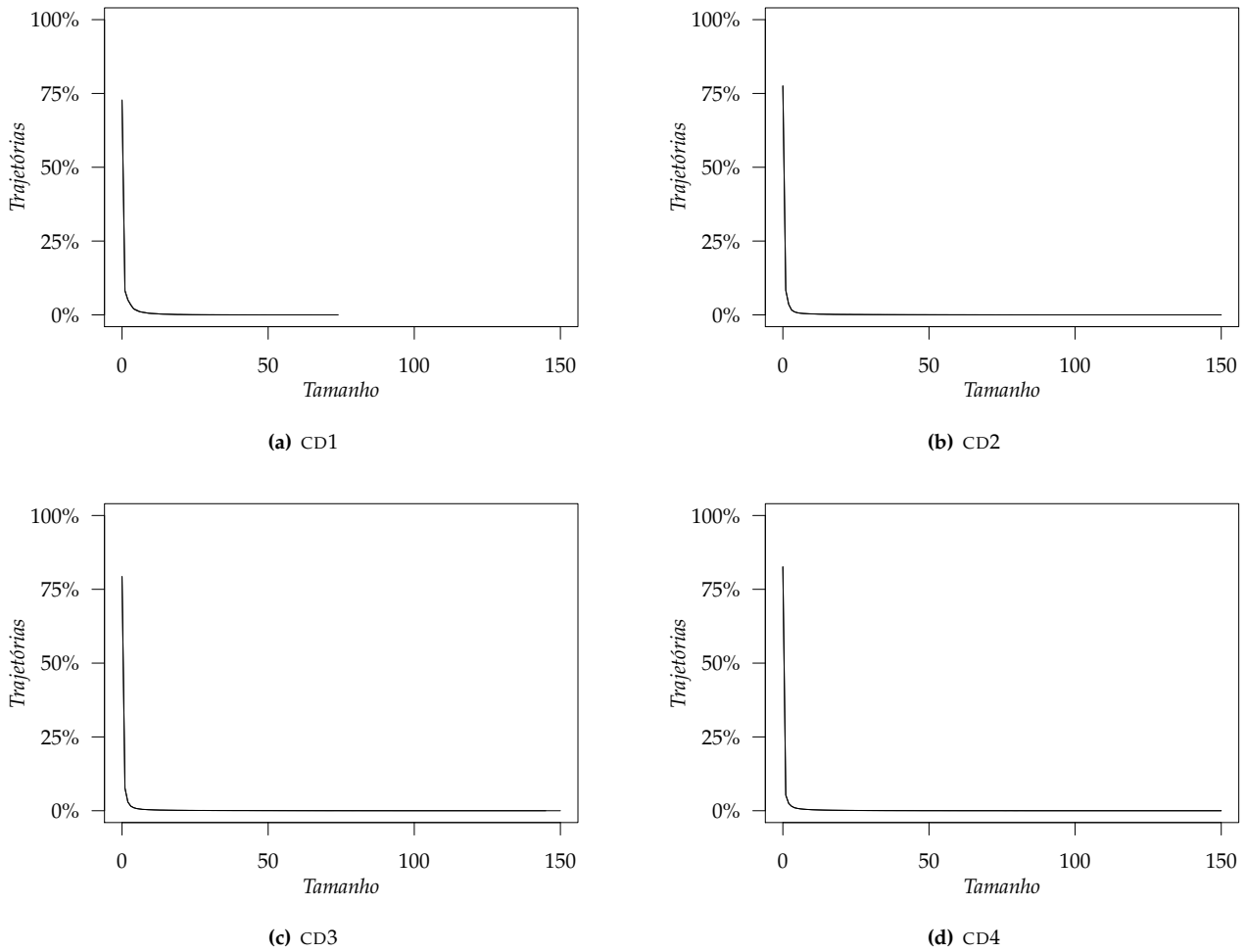
#### 4.4.2 Estratégia de exploração do paralelismo

A implementação em GPU utiliza as trajetórias como elemento básico de paralelização. Alguns cuidados, no entanto, foram tomados para garantir que um bom desempenho fosse atingido. Ao invés de a GPU ser invocada apenas uma vez e cada thread calcular todos os pontos de uma trajetória, a GPU é chamada várias vezes e cada thread calcula um único ponto. Os motivos para que essa estratégia tenha sido adotada são apresentados a seguir.

Uma das limitações impostas aos programas que executam nas GPUs é que eles não podem alocar memória. Toda memória utilizada por eles precisa ter sido alocada antes de eles começarem sua execução. No caso da tractografia, essa limitação tem uma implicação severa: um thread que pretenda calcular todos os pontos de uma trajetória precisa ter memória *pré-alocada* suficiente para armazenar todos os pontos dessa trajetória.

O número de pontos de uma trajetória, contudo, só é conhecido quando ela termina de ser calculada. É necessário, portanto, que se defina um tamanho máximo para as trajetórias e que cada thread tenha reservada para si uma quantidade de memória suficiente para armazenar uma trajetória de tamanho máximo. Como cada thread, porém, pode encontrar um número de pontos que varia entre zero e o número máximo de pontos, todos os threads que encontrarem menos pontos que o máximo estarão desperdiçando uma certa quantidade de memória.





**Figura 4.3:** Histogramas do tamanho das trajetórias para os 4 conjuntos de dados. Os dados foram obtidos com pontos semente no centro de todos voxels dos volumes.

Essa memória desperdiçada não apenas implica uma necessidade maior de memória disponível na GPU, mas também a cópia entre CPU e GPU de dados que não contém informação nenhuma. Se essa quantidade de memória desperdiçada for muito grande, o desempenho da tractografia será prejudicado.

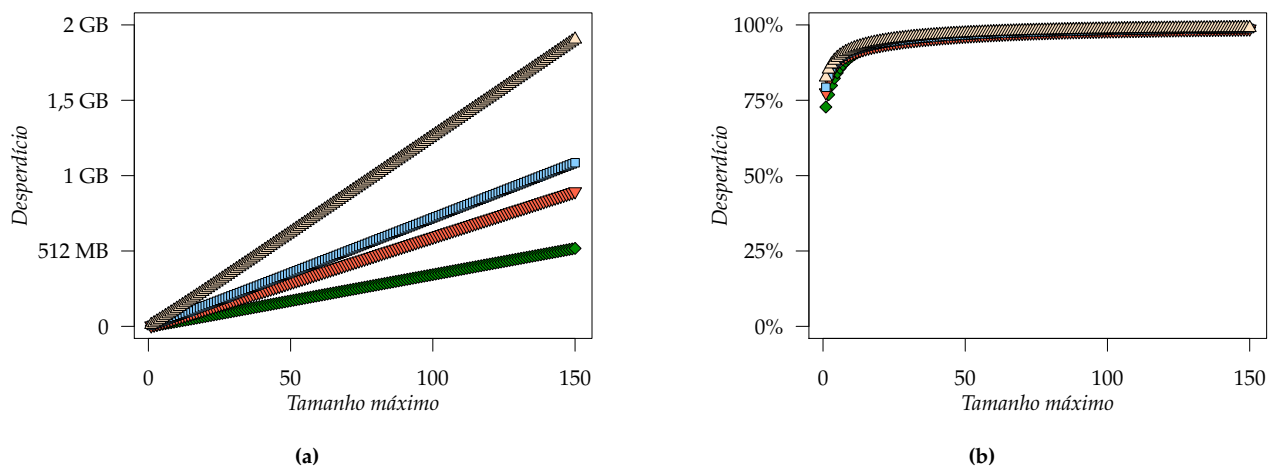
Para averiguar a quantidade de memória desperdiçada, o seguinte experimento foi realizado: a tractografia em GPU foi executada para todos os 4 conjuntos de dados e o tamanho das trajetórias foi registrado. Os pontos semente foram posicionados no centro de todos os voxels de cada volume — o que significa que o número de pontos semente variou conforme o conjunto de dados. O tamanho máximo das trajetórias foi limitado a 150 pontos.

O histograma do tamanho das trajetórias para os 4 conjuntos de dados pode ser visto na Figura 4.3. Nota-se que a maior parte das trajetórias tem uma quantidade pequena de pontos. A Tabela 4.1 mostra o número absoluto de trajetórias em diferentes faixas de tamanho.

Apesar de proporcionalmente a maioria das trajetórias terem poucos ou nenhum ponto, o número de trajetórias que atingem, por exemplo, 50 pontos (um número que já é capaz de gerar visualizações relevantes) é significativo, especialmente quando se leva em consideração que os pontos semente foram distribuídos por *todo* o volume, inclusive em áreas fora do corpo do paciente. Nota-se também que pouquíssimas trajetórias

**Tabela 4.1:** Número absoluto de trajetórias que terminam em determinadas faixas de tamanho.

	Tamanho							
	0	1 – 24	25 – 49	50 – 74	75 – 99	100 – 124	125 – 149	> 150
CD1	453.010	166.601	2.863	118	0	0	0	0
CD2	838.682	216.198	19.249	5.906	1.110	188	9	2
CD3	1.039.612	241.987	20.666	6.336	1.706	359	51	3
CD4	1.895.024	364.794	25.416	6.150	1.825	475	64	12



**Figura 4.4:** Desperdício de memória, (a) absoluto e (b) relativo, com cada thread calculando todos os pontos de uma trajetória. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

têm 150 pontos ou mais.

O alto número de trajetórias que contêm poucos ou nenhum ponto acarreta um alto desperdício, pois a cada trajetória é reservada memória suficiente para armazenar 150 pontos. Reduzir o número máximo de pontos por trajetória diminuiria o desperdício, mas também poderia as trajetórias mais longas, que via de regra são as mais importantes.

A Figura 4.4 mostra o desperdício de memória relativo ao total alocado e também em termos absolutos, para um tamanho máximo variando entre 1 e 150 pontos. Os gráficos foram obtidos com base nos histogramas da Figura 4.3. Nota-se que, para tamanhos máximos acima de 50 pontos, praticamente toda a memória é desperdiçada. Em termos absolutos, o CD4 desperdiça até quase 2 GB de memória, o que é uma quantidade significativamente alta para a quantidade de memória disponível na maioria das GPUs da atualidade.

Para resolver o problema do desperdício de memória, a solução é fazer com que cada thread calcule apenas uma certa quantidade de pontos de sua trajetória, e não todos. Após o término da execução dos threads, os dados são então lidos pela CPU, que os armazena de maneira apropriada e invoca novamente a GPU mas dessa vez apenas para as trajetórias que ainda não terminaram.

A implementação em GPU da tractografia adota essa solução de um modo simples, mas que mostrou-se bastante efetivo: cada thread calcula apenas *um* ponto de sua trajetória. Desta forma, o desperdício de memória

fica reduzido ao mínimo. A Figura 4.5 ilustra a diferença entre a estratégia adotada pela implementação em GPU e a estratégia com alto desperdício de memória.

Na estratégia empregada, a quantidade de memória desperdiçada por cada trajetória é igual ao espaço necessário para armazenar um único ponto tridimensional, ou seja,  $3 \times 4 \text{ bytes} = 12 \text{ bytes}$ . Uma desvantagem dessa estratégia, entretanto, é a necessidade de gerenciar quais trajetórias estão sendo calculadas a cada chamada à GPU. Quando os resultados tractográficos são lidos da GPU, é preciso saber a qual trajetória o ponto calculado por cada thread pertence.

A implementação em GPU mantém, com esse propósito, uma estrutura de índices, como ilustra a Figura 4.6. Na primeira chamada à GPU, quando havia 6 trajetórias sendo calculadas, a estrutura continha os 6 índices correspondentes àquelas trajetórias. A trajetória  $t_5$  encontrou algum critério de parada na segunda chamada e já não consta da lista de trajetórias da terceira chamada. A estrutura continua sendo atualizada até que todas as trajetórias tenham sido terminadas.

### 4.4.3 Características da implementação

A implementação em GPU utiliza a tecnologia CUDA para acessar os recursos computacionais das placas de vídeo. Dois núcleos foram escritos para serem executados na GPU, `PassoTractográfico` e `Validação`. Este último é executado apenas uma vez, no início da tractografia, para validar os pontos semente. Estritamente, a validação dos pontos semente poderia ser feita dentro do `PassoTractográfico`, mas optando-se pela separação em dois núcleos é possível otimizar a validação. Como visto na seção anterior, cerca de 75% das trajetórias têm tamanho zero, o que justifica essa otimização. A declaração dos dois núcleos pode ser vista na Figura 4.7.

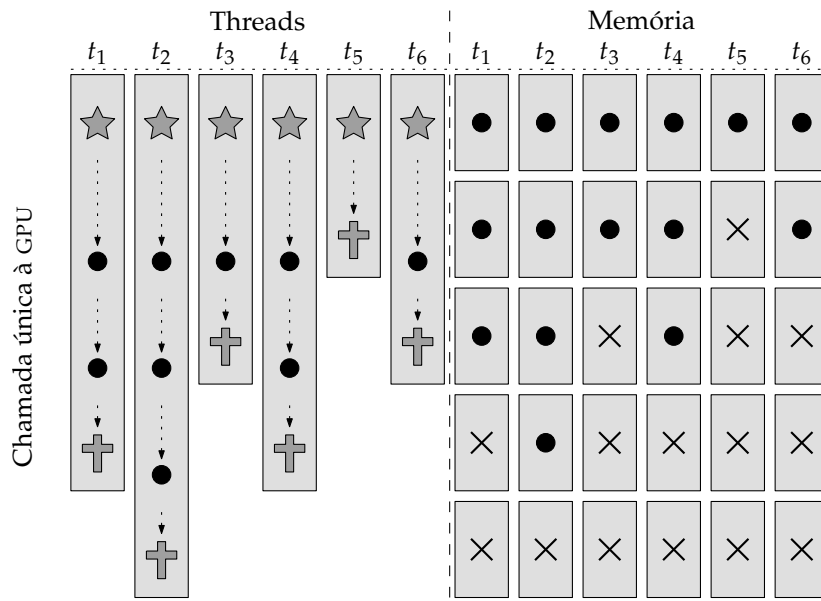
O núcleo `Validação` recebe os pontos semente a serem validados, um local para armazenar as cores, a quantidade dos pontos semente e um local para armazenar o resultado da validação. O local para a gravação das cores dos pontos semente pode ser nulo, caso em que as cores não são calculadas.

O núcleo `PassoTractográfico` recebe os pontos das trajetórias, um local para armazenar as cores, a quantidade de trajetórias, o sentido a ser inspecionado e um local para armazenar a indicação das trajetórias que terminaram. O local das cores, também nesse núcleo, pode ser nulo. O ponto calculado pelo núcleo é armazenado no mesmo local em que o ponto anterior foi lido. O sentido assume os valores  $-1$  ou  $1$ , correspondendo aos dois possíveis sentidos da difusão.

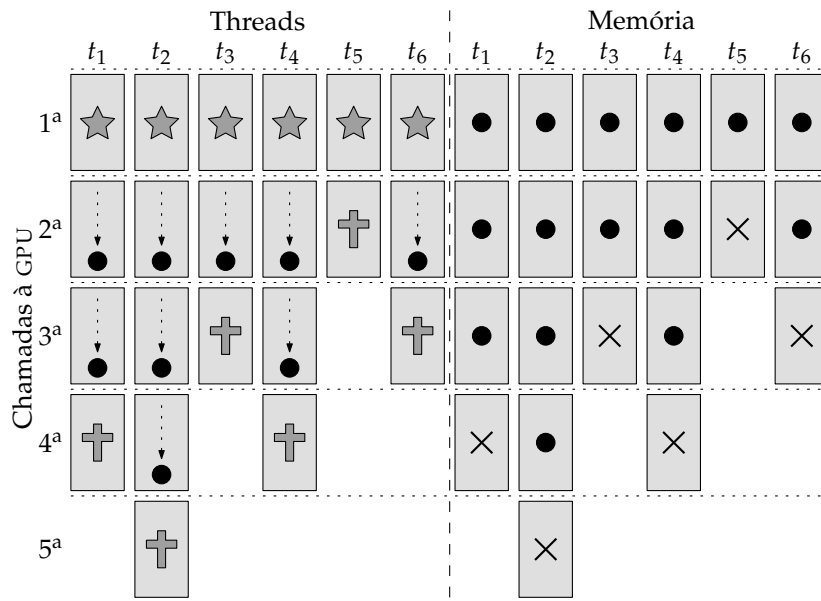
Ambos os núcleos são executados com 64 threads por bloco dispostos bidimensionalmente em  $8 \times 8$  threads. A grade é unidimensional e contém o menor número de blocos suficiente para processar todas as trajetórias ainda não terminadas.

O campo discreto de tensores calculado previamente a partir do conjunto de dados fica armazenado em uma textura devido à alta frequência com que é acessado. Os núcleos acessam o campo de tensores para fazer a interpolação trilinear em pontos arbitrários do espaço. A textura do campo de tensores é acessada com frequência pelo núcleo `PassoTractográfico`. Cada tensor ocupa dois `float4` da textura; a interpolação trilinear utiliza 8 tensores em seu cálculo; o algoritmo Runge–Kutta de 4ª ordem precisa de 4 tensores interpolados. Portanto, o total de acessos aos valores da textura é, para cada ponto adicional,

$$2 \text{ float4} \times 8 \text{ tensores} \times 4 = 64 \text{ float4},$$



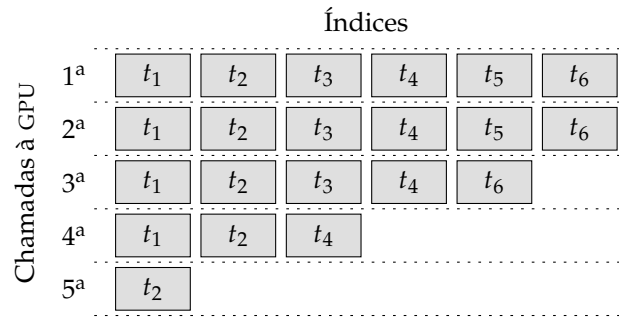
(a)



(b)

Figura 4.5: Estrutura de threads e uso de memória com (a) uma única chamada à GPU e com (b) múltiplas chamadas à GPU.

★ = ponto semente, ● = ponto de trajetória, † = fim de trajetória, ✕ = memória desperdiçada.



**Figura 4.6:** Estrutura de índices usada pela CPU para controlar quais trajetórias estão sendo calculadas em um dado momento.

```

__global__ void Validação(float4 *pontos_semente,
                        float4 *cores,
                        unsigned quantidade,
                        int *resultado);

__global__ void PassoTractográfico(float4 *pontos,
                                   float4 *cores,
                                   unsigned quantidade,
                                   int sentido,
                                   int *terminado);

```

**Figura 4.7:** Declaração dos dois núcleos utilizados pela implementação em GPU.

ou seja,

$$64 \times 4 \text{ float} \times 4 \text{ bytes} = 1024 \text{ bytes} = 1 \text{ KB.}$$

O número final de acessos pode ser ainda maior, pois o passo tractográfico só é completado quando a distância de 1 mm é percorrida, o que pode requerer mais de um ponto calculado pelo algoritmo de Runge–Kutta.

Por ser acessada tantas vezes para o cálculo de um único ponto de trajetória, esse é um ponto crítico da implementação de GPU. Na tecnologia CUDA, os acessos à memória são otimizados quando são coalescidos, ou seja, são feitos de maneira regular. No caso da tractografia, é difícil arquitetar um método em que os acessos sejam coalescidos, já que eles dependem da localização das trajetórias. A implementação em GPU provavelmente obteria ganhos de desempenho mais significativos se conseguisse acessar a textura do campo de tensores de maneira mais ordenada.

## 4.5 Experimentos

### 4.5.1 Planejamento

Os experimentos deste capítulo medem o tempo de execução da tractografia em lote aplicada aos quatro conjuntos de dados. Define-se aqui como tempo de execução o tempo que o software leva para calcular todos os pontos de todas as trajetórias, porém nesse tempo não estão incluídos a leitura dos arquivos, o cálculo do campo discreto de tensores e a gravação dos resultados. Ainda que tais etapas sejam todas essenciais, este trabalho preocupa-se somente com o cálculo dos resultados tractográficos.

Os experimentos consistem em 4 subexperimentos, que correspondem aos 4 conjuntos de dados. A necessidade dessa divisão decorre do fato de que há pouco sentido em comparar tempos de execução da tractografia aplicada a diferentes conjuntos de dados. Por exemplo, um conjunto de dados com ruído muito alto fará com que poucas trajetórias sejam encontradas e, por conseguinte, a tractografia seja executada rapidamente. A separação em 4 subexperimentos visa a manter distintos os resultados obtidos com cada conjunto de dados.

Uma diferença importante entre os subexperimentos é a quantidade de pontos semente utilizada em cada um deles. Os pontos semente foram postos no centro de cada voxel do volume correspondente, aproveitando-se assim o conjunto de dados inteiro. Como o número de voxels varia de um conjunto de dados para outro, o tempo de execução da tractografia também varia entre os conjuntos de dados. Mas porque os conjuntos de dados levariam de qualquer forma a tempos de execução bastante distintos, essa diferença adicional não prejudica a qualidade dos resultados.

Os subexperimentos são estruturalmente idênticos. De maneira formal, constituem-se na aplicação de dois tratamentos, a implementação em CPU e em GPU, a quatro unidades experimentais, que correspondem às quatro configurações de hardware. Para estimar melhor o valor médio dos tempos de execução, cada tratamento foi replicado 64 vezes.

As medições foram executadas em ordem aleatória no contexto de cada unidade experimental, inclusive entre subexperimentos. O total de medições feitas em cada unidade experimental foi, desta forma,

$$2 \text{ tratamentos} \times 64 \text{ replicações} \times 4 \text{ conjuntos de dados} = 512 \text{ medições.}$$

A mesma ordem aleatória foi utilizada em todas as configurações.

#### 4.5.2 Execução

Os experimentos foram conduzidos de maneira automática por intermédio de scripts, para reduzir ao mínimo a possibilidade de erros causados por interferências manuais. A coleta dos resultados e seu posterior agregamento em tabelas também foi realizado de forma automática.

Os parâmetros do método de tractografia foram ajustados em todos os casos para os seguintes valores:

- Valor mínimo de anisotropia fracional:  $150 \times 10^{-3}$ ;
- Valor mínimo de difusividade média:  $50 \times 10^{-6} \text{ mm}^2\text{s}^{-1}$ ;
- Ângulo máximo entre segmentos de reta:  $20^\circ$ .

Como havia quatro GPUs distintas e apenas dois computadores físicos, os experimentos não foram conduzidos simultaneamente. Cronologicamente, os experimentos foram conduzidos assim:

1. A ordem de execução das 512 medições em cada configuração foi determinada aleatoriamente;
2. As duas primeiras configurações de hardware, 1 e 2, foram montadas;
3. A tractografia foi executada de acordo com a ordem estabelecida;
4. As duas últimas configurações de hardware, 3 e 4, foram montadas;

5. A tractografia foi executada de acordo com a ordem estabelecida.

Antes do início da execução da tractografia em cada configuração, um script certificava-se de que nenhum processo supérfluo estava em segundo plano. Apenas os processos fundamentais ao sistema e um processo que permitia monitoração e controle remoto dos experimentos foram mantidos em execução.

Apesar de que os experimentos deste capítulo não geram resultado gráfico nenhum na tela, o servidor de gráficos também era iniciado automaticamente, por causa de limitações no driver de vídeo da NVIDIA. Quando o servidor gráfico não estava ativo, o desempenho da tractografia em GPU caía bastante, por motivos que permanecem obscuros até este momento.

Para minimizar o número de fatores que pudessem influenciar os tempos de execução, a temperatura ambiente foi controlada para que permanecesse constante.

### 4.6 Resultados

As medições obtidas com cada conjunto de dados na implementação em CPU constituem uma variável independente da configuração, pois o único componente variável nas configurações é a placa de vídeo. Como a implementação em CPU calcula os resultado mas não os exibe na tela, a placa de vídeo não é um fator determinante no tempo de execução.

Para a apresentação dos resultados, portanto, define-se que existem cinco *plataformas*: a CPU e cada uma das quatro GPUs. Assim, para a plataforma CPU foram obtidas 4 vezes mais medições replicadas do que para as demais.

A Tabela 4.2 apresenta os resultados dos experimentos para os quatro conjuntos de dados e as cinco plataformas. Os valores médios de cada variável são mostrados de forma gráfica na Figura 4.8.

Em todos os casos, as melhores médias foram obtidas pela GPU3. A GPU1 foi a mais lenta, mas ainda assim foi significativamente mais rápida que a CPU. Considerando-se, por exemplo, o CD3, vê-se que a CPU leva cerca de 1 minuto e 47 segundos para executar a tractografia, enquanto a GPU mais lenta o faz em menos de 13 segundos e a mais rápida em pouco menos de 3 segundos.

O ganho de desempenho obtido pelas GPUs em comparação às CPUs é mostrado analiticamente pela Tabela 4.3 e graficamente pela Figura 4.9. Observa-se que o ganho de desempenho das GPUs 3 e 4 ficou, para todos os conjuntos de dados, acima de 30 vezes e que, mesmo uma GPU modesta como a primeira conseguiu consistentemente executar a tractografia mais de 8 vezes mais rápido que a CPU.

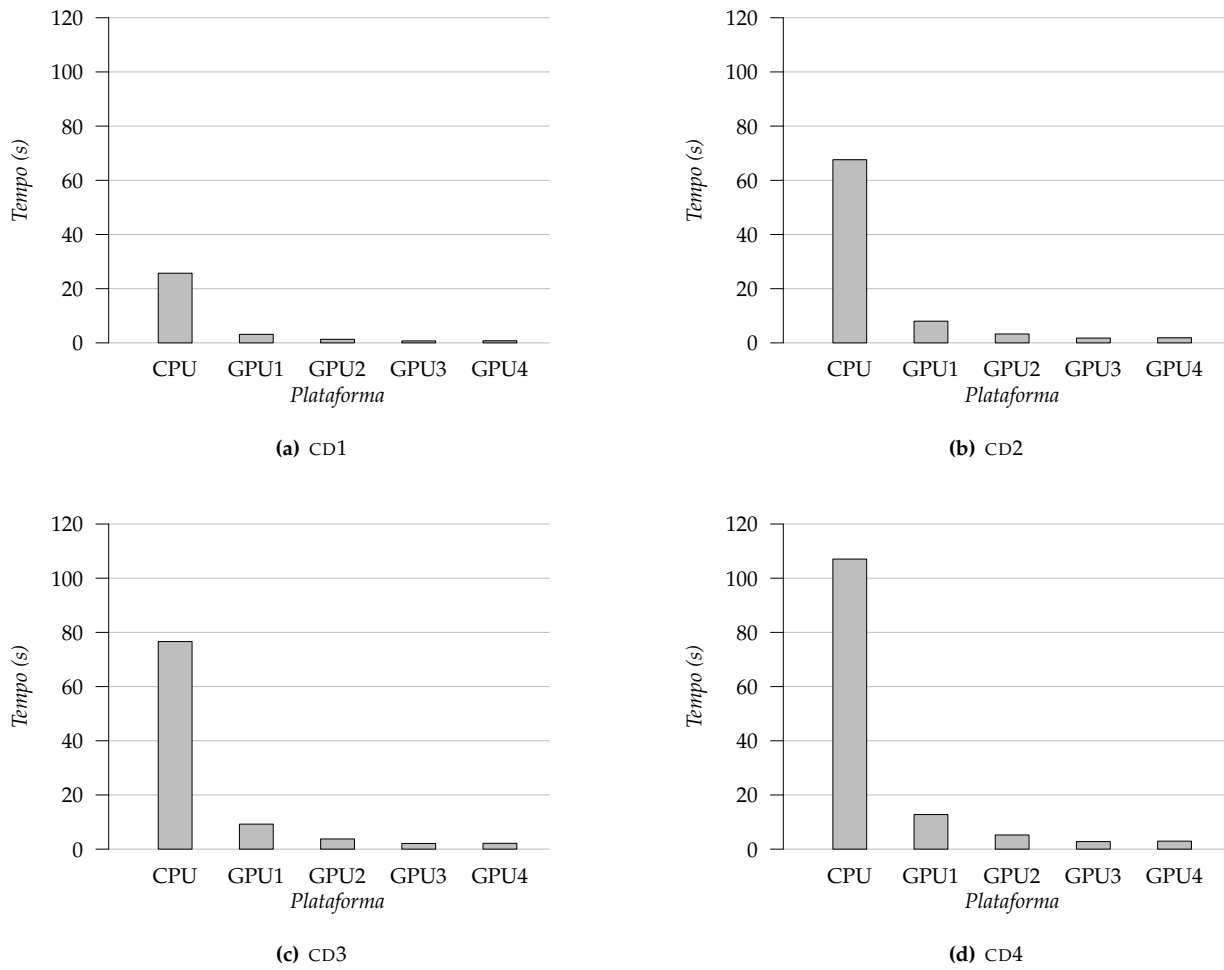
**Tabela 4.2:** Resultado dos experimentos para todos conjuntos de dados e plataformas. Os valores em negrito indicam os menores da linha correspondente.

		CPU	GPU1	GPU2	GPU3	GPU4
CD1	Replicações	256	64	64	64	64
	Média	25,71	3,15	1,32	<b>0,73</b>	0,77
	Mínimo	25,33	3,14	1,31	<b>0,73</b>	0,77
	Máximo	26,16	3,16	1,32	<b>0,74</b>	0,77
	Desvio padrão	0,18	0,00	0,00	0,00	0,00
CD2	Replicações	256	64	64	64	64
	Média	67,60	8,01	3,28	<b>1,77</b>	1,87
	Mínimo	66,40	7,99	3,27	<b>1,76</b>	1,86
	Máximo	69,75	8,02	3,29	<b>1,78</b>	1,88
	Desvio padrão	0,61	0,00	0,00	0,00	0,00
CD3	Replicações	256	64	64	64	64
	Média	76,62	9,23	3,76	<b>2,10</b>	2,14
	Mínimo	75,36	9,22	3,75	<b>2,09</b>	2,13
	Máximo	79,14	9,25	3,77	<b>2,11</b>	2,15
	Desvio padrão	0,69	0,01	0,00	0,00	0,00
CD4	Replicações	256	64	64	64	64
	Média	107,06	12,76	5,23	<b>2,79</b>	2,96
	Mínimo	105,02	12,74	5,21	<b>2,77</b>	2,94
	Máximo	110,31	12,78	5,24	<b>2,80</b>	2,97
	Desvio padrão	1,08	0,01	0,01	0,01	0,01

**Tabela 4.3:** Ganho de desempenho das GPUs em comparação às CPUs. Os valores foram calculados a partir das médias da Tabela 4.2. Os valores em negrito indicam o melhor desempenho na linha correspondente.

	GPU1	GPU2	GPU3	GPU4
CD1	8,16	19,55	<b>35,14</b>	33,42
CD2	8,44	20,59	<b>38,18</b>	36,13
CD3	8,30	20,37	<b>36,51</b>	35,74
CD4	8,39	20,48	<b>38,41</b>	36,20





**Figura 4.8:** Valores médios dos resultados dos experimentos para todos conjuntos de dados e plataformas. A mesma escala é utilizada em todos os gráficos.

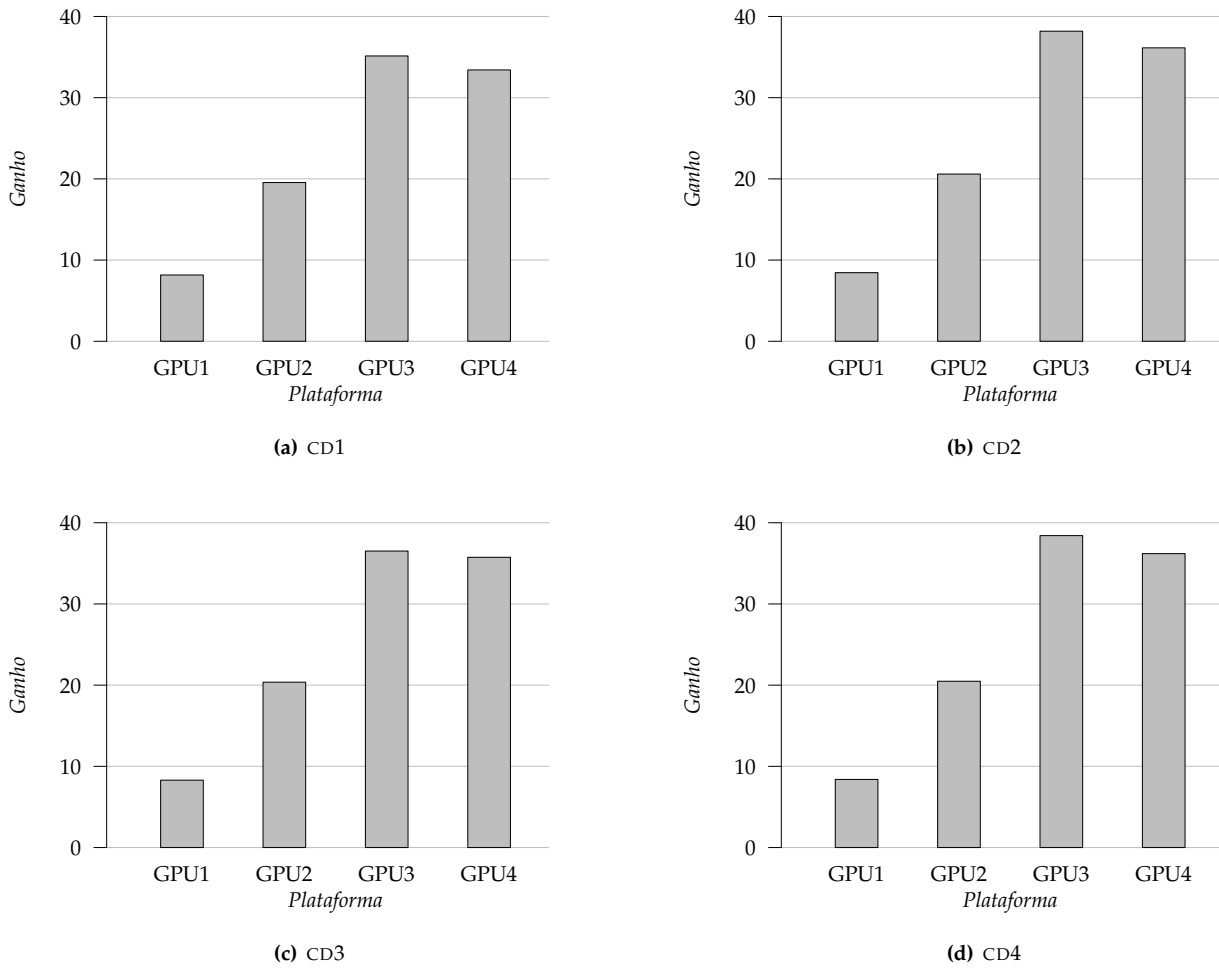


Figura 4.9: Ganho de desempenho das GPUs em comparação às CPUs, de acordo com a Tabela 4.3.

## Capítulo 5

# Tractografia em tempo real

### 5.1 A estação radiológica

A abordagem de tractografia interativa proposta neste trabalho foi implementada no contexto da estação radiológica tridimensional desenvolvida pelo Grupo Cyclops (Carvalho et al., 2008; Silva et al., 2009). Essa estação disponibiliza um ambiente completo para a manipulação de objetos tridimensionais, constituindo assim uma plataforma natural para a integração da biblioteca de tractografia descrita no Capítulo 4.

A estação radiológica foi desenvolvida na linguagem C++, que permite o desenvolvimento de softwares complexos e otimizados, e faz uso também da biblioteca wxWidgets (wxWidgets, 2009), que disponibiliza componentes para a criação de interfaces gráficas, e do padrão OpenGL (OpenGL, 2009), utilizado para a produção de gráficos 2D e 3D. O fato de que tanto a estação quanto a biblioteca tractográfica foram escritas em C++ tornou menos árduo o processo de integração.

Exames de ressonância magnética e tomografia computadorizada podem ser abertos pela estação radiológica. Assim que um exame é carregado, os planos sagital, coronal e axial são exibidos na tela. Eles podem ser movimentados ao longo de seus eixos e ocultados ou reexibidos a qualquer momento. Os planos são sempre interpolados, o que garante uma visualização consistente para qualquer posição que eles assumam.

Funções básicas de visualização também estão disponíveis na estação radiológica. É possível controlar o zoom e girar e reposicionar a câmera. Os objetos exibidos na tela podem ter seu nível de transparência ajustado e fontes de luz podem ser adicionadas. Esses recursos permitem a elaboração de visualizações expressivas de dados médicos.

Também possível na estação radiológica é a reconstrução tridimensional de estruturas anatômicas. Partindo de um limite inferior e superior para os valores das imagens, a estação gera reconstruções tridimensionais através do algoritmo de cubos marchantes (Lorensen e Cline, 1987), produzindo visualizações como a da Figura 5.1.

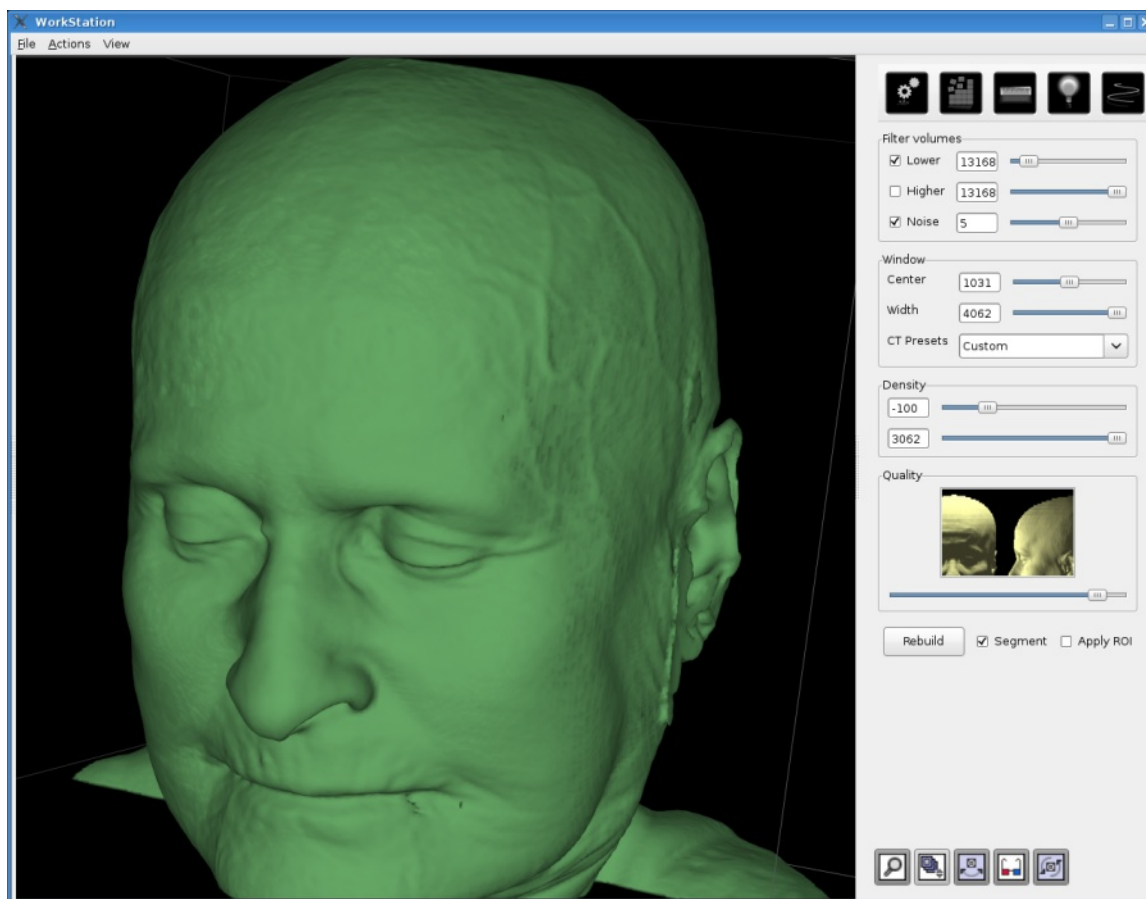


Figura 5.1: Reconstrução anatômica tridimensional gerada pela estação radiológica do Grupo Cyclops.

## 5.2 O módulo de tractografia da estação radiológica

A tractografia foi integrada à estação radiológica tridimensional através da adição de um painel, que é exibido quando o botão de tractografia é pressionado. Quando o painel é ativado, um volume de interesse é exibido na tela. É a partir desse volume de interesse que os resultados tractográficos são calculados, de acordo com o método estabelecido no capítulo anterior. A Figura 5.2 mostra a estação radiológica sendo utilizada para explorar resultados tractográficos. O painel de tractografia está localizado à direita da tela, e o botão no canto superior direito.

O painel de tractografia contém controles para o ajuste do comportamento do algoritmo de tractografia. Os critérios de parada do algoritmo são ajustados através dos três primeiros controles (de cima para baixo), que são os seguintes:

- Valor mínimo para a anisotropia fracional, entre 0 e  $500 \times 10^{-3}$  (valores adimensionais); o valor inicial é  $150 \times 10^{-3}$ . O índice de anisotropia fracional indica o quão dirigida é a difusão. Ajustar esse controle para um valor alto diminui o tamanho das trajetórias exibidas, pois apenas pontos com um valor de anisotropia fracional superior ao valor mínimo são levados em consideração.
- Valor mínimo para a difusividade média, variando entre 400 e  $1200 \text{ mm}^2\text{s}^{-1}$ ; o valor inicial é  $400 \text{ mm}^2\text{s}^{-1}$ . O índice de difusividade média denota a força da difusão. Valores altos nesse controle também diminuem

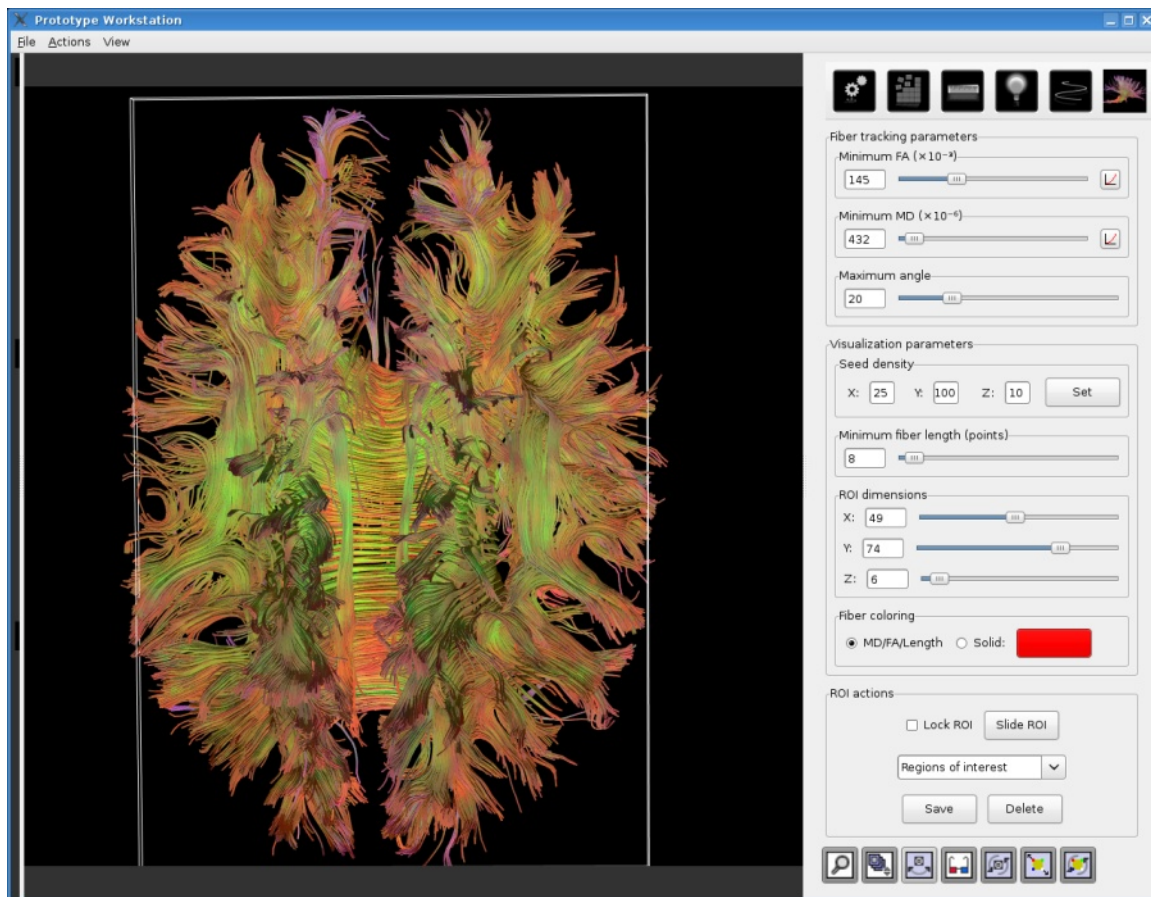


Figura 5.2: Resultados tractográficos sendo exibidos na estação radiológica.

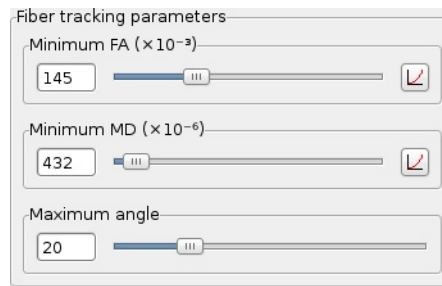
o tamanho das trajetórias encontradas.

- Valor máximo para o ângulo entre dois segmentos sucessivos das trajetórias, entre 0 e 90°; o valor inicial é 20°. Um valor *baixo* nesse controle faz com que curvas repentinas não sejam levadas em conta, o que diminui o tamanho das trajetórias.

O número de pontos semente dentro do volume de interesse é controlado através de três campos, que correspondem às dimensões  $x$ ,  $y$  e  $z$  do espaço euclidiano. Os pontos semente são distribuídos ao longo do volume de interesse de acordo com uma grade tridimensional cujas dimensões são dadas por esses três campos. O valor inicial para os três controles é 10, o que resulta num total de 1.000 pontos semente. O painel de tractografia não impõe um limite máximo para nenhum desses campos; os valores máximos dependem da quantidade de memória disponível no computador (na CPU e na GPU) em que a estação radiológica está sendo executada.

O tamanho do volume de interesse é definido por três outros controles, que correspondem às três dimensões do espaço euclidiano. O tamanho é ajustado através de valores percentuais relativos ao tamanho total do volume. O valor inicial para as três dimensões é 10%.

A cor das trajetórias exibidas na tela incorpora os índices de difusividade média e anisotropia fracional e também a posição dos pontos dentro da trajetória. O cálculo da cor é realizado pela GPU ao mesmo tempo em que a tractografia é executada e é feito da seguinte maneira: aos canais R e G (vermelho e verde) são atribuídos



**Figura 5.3:** Os critérios de parada do algoritmo de tractografia são ajustados arrastando estes controles. Enquanto o usuário os arrasta, os resultados tractográficos são constantemente atualizados para refletir os novos valores.

os valores de difusividade média e anisotropia fracional, respectivamente; ao canal B (azul) é atribuído o número de pontos já encontrados na trajetória. Esses valores são redistribuídos entre 0 e 1 de acordo com valores mínimos e máximos para cada um dos índices e do número de pontos.

Ainda que seja comum o uso de esquemas de cor em que os tons indicam a direção de difusão, eles não foram utilizados aqui porque a estação radiológica do Grupo Cyclops trabalha naturalmente com objetos tridimensionais. A direção de difusão das trajetórias pode, por esse motivo, ser verificada de maneira direta através da visualização gerada.

### 5.3 Interatividade do módulo de tractografia

A interatividade proporcionada pelo módulo de tractografia não se restringe à manipulação visual dos resultados e ao controle através do painel de tractografia. Ela incorpora, de fato, um paradigma específico de interatividade, cuja descrição e desdobramentos são apresentados nas subseções seguintes.

É relevante observar que a proposta de interatividade apresentada aqui *não* exige, de forma alguma, o uso de GPUs. No entanto, por exigir um alto desempenho no cálculo dos resultados tractográficos, essa proposta beneficia-se significativamente do uso de GPUs. Os experimentos deste capítulo exprimem exemplarmente a diferença de desempenho entre GPUs e CPUs nessa proposta de interatividade.

#### 5.3.1 Paradigma de atualidade permanente

O tipo de interatividade que a ferramenta de tractografia proporciona pode ser melhor descrita introduzindo-se o paradigma de atualidade permanente, que norteou o desenvolvimento da ferramenta. Nesse paradigma, os resultados tractográficos exibidos na tela estão sempre atualizados, ou seja, toda e qualquer ação do usuário na interface é refletida imediatamente nos resultados.

A adoção do paradigma de atualidade permanente têm duas conseqüências importantes. Uma delas é que o cálculo dos resultados tractográficos precisa ser feito em vários momentos, como, por exemplo, quando o usuário está movendo o volume de interesse ou quando um dos controles está sendo deslizado (esse último caso está ilustrado na Figura 5.3). A interface gráfica, portanto, precisa estar atenta para todas as ações do usuário, de maneira que em nenhum momento os resultados da tela estejam desatualizados.

A conseqüência mais importante, contudo, é que os resultados tractográficos precisam ser gerados rapida-

mente. Se os resultados tardarem muito a aparecer na tela, a interatividade da ferramenta será muito prejudicada, pois o usuário terá um momento de espera a cada passo que tomar. A rapidez na geração dos resultados é, de fato, mais importante no paradigma de atualidade permanente do que em outros paradigmas, já que os resultados são recalculados com uma frequência maior e, portanto, a lentidão no cálculo dos resultados também seria percebida com uma frequência maior.

Do ponto de vista do usuário, o paradigma de atualidade permanente é especialmente relevante nas interações compostas por vários pequenos passos, como é o caso do movimento do volume de interesse na tela e do ajuste dos critérios de parada do algoritmo de tractografia.

O movimento do volume de interesse na tela faz com que os pontos semente sejam reposicionados. Para que o paradigma de interatividade seja mantido, portanto, os resultados tractográficos são recalculados para cada nova posição que o volume de interesse assuma no espaço, sem que nenhum armazenamento de resultados aconteça. Em particular, quando o usuário está arrastando o volume de interesse na tela, o menor movimento do mouse faz com que os resultados sejam atualizados. O que o usuário observa na tela, ao arrastar o volume de interesse, é um efeito muito semelhante a uma animação: as trajetórias aos poucos agrupam-se de diferentes maneiras, novos grupos são formados ou desaparecem, as cores variam.

Quando os critérios de parada do algoritmo de tractografia são ajustados, a ferramenta recalcula os resultados automaticamente, inclusive enquanto o usuário está arrastando o controle na interface gráfica. O resultado é que os controles, ao serem arrastados, fazem com que as trajetórias aumentem ou diminuam de tamanho, em um efeito que também lembra uma animação. O usuário, dessa forma, pode ver de maneira imediata o impacto dos critérios de parada nos resultados tractográficos.

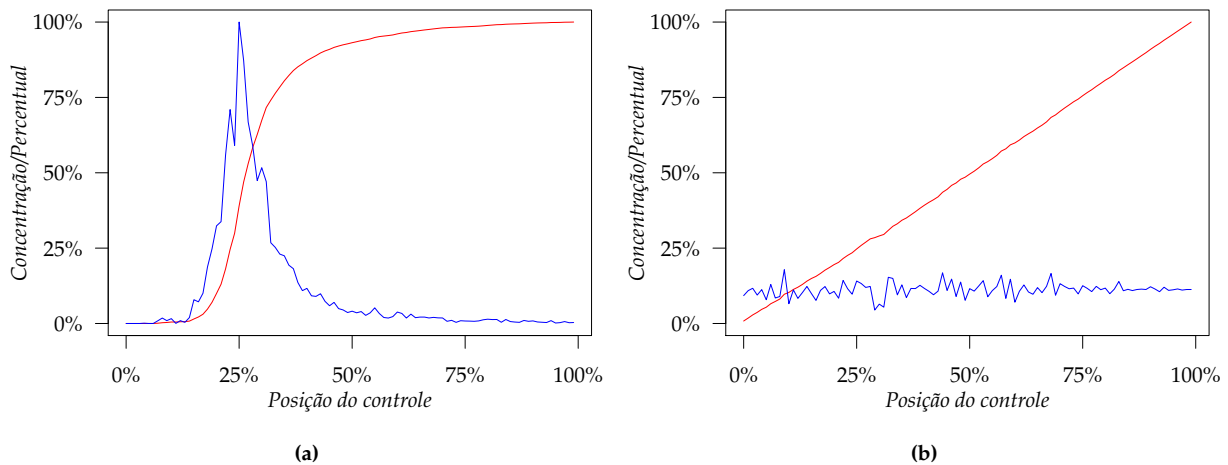
### 5.3.2 Equalização dos parâmetros

O uso do paradigma de atualidade permanente traz à tona questões que antes não tinham razão de ser. Uma delas refere-se aos critérios de parada do algoritmo: quando o usuário arrasta os controles para ajustá-los, logo percebe que a quantidade e o tamanho das trajetórias exibidas na tela não aumentam e diminuem linearmente conforme o valor dos critérios de parada. Trata-se de uma consequência natural do fato de que os valores dos índices não estão uniformemente distribuídos.

Tome-se, por exemplo, a anisotropia fracional. Em um conjunto de dados típico existem valores baixos e altos de anisotropia fracional, mas alguns intervalos de valores são mais frequentes do que outros. Quando o controle de ajuste do valor mínimo de anisotropia fracional passa por esse intervalo, portanto, um número maior de trajetórias aparecerá ou desaparecerá (conforme a direção do movimento do usuário) em comparação a outras regiões.

Para que os parâmetros possam ser explorados com facilidade nas suas regiões mais sensíveis, foi criado um método que permite que incrementos (ou decrementos) iguais nos controles de ajuste produzam impactos visuais iguais na tela. Isso significa que, se o usuário arrastar o controle de valor mínimo de anisotropia fracional por 10% de sua extensão, sempre o mesmo número de pontos de trajetória aparecerão ou desaparecerão da tela, independentemente de outros fatores.

O método que realiza essa tarefa é baseado na equalização de histograma de imagens digitais. Ao invés de aplicar a equalização ao histograma de uma imagem, esse método aplica-o ao histograma de impacto visual



**Figura 5.4:** Histogramas de impacto visual do critério de parada de valor mínimo de difusividade média, (a) antes e (b) depois da equalização.

da imagem. Esse histograma indica o número de pontos que aparecem (ou desaparecem) conforme diferentes valores dos parâmetros são utilizados. A equalização desse histograma produz exatamente o efeito desejado: o impacto visual na tela varia linearmente com o movimento aplicado aos controles de ajuste dos critérios de parada.

A Figura 5.4 contém dois histogramas de impacto visual do parâmetro de difusividade média, um antes e outro após a equalização. Percebe-se que, antes da equalização, grande parte dos pontos de trajetória aparecem ou desaparecem quando o controle é deslizado entre 20% e 30% de sua extensão. No histograma equalizado, por outro lado, o aparecimento e desaparecimento de pontos de trajetória espalha-se uniformemente ao longo de toda a extensão do controle.

O usuário pode solicitar a equalização do histograma de impacto visual dos parâmetros de anisotropia fracional e difusividade média. Para tal, basta que pressione o botão situado à direita dos respectivos controles, que podem ser vistos na Figura 5.3. Atualmente, a equalização é executada exclusivamente na GPU.

### 5.3.3 Estratégia de renderização

A velocidade com que os resultados tractográficos em si são calculados é o fator determinante no desempenho da ferramenta de tractografia, mas a maneira como as trajetórias são desenhadas na tela também tem um impacto significativo no tempo de resposta. Por esse motivo, a ferramenta de tractografia emprega duas estratégias de renderização distintas: uma para os momentos em que os resultados são calculados com muita frequência e outra para exibir resultados que permanecerão na tela por um período considerável de tempo. As duas estratégias estão ilustradas na Figura 5.5.

A primeira estratégia de renderização mostra as trajetórias na tela como linhas simples. Essa estratégia é executada muito rapidamente porque os resultados tractográficos oriundos da GPU podem ser repassados diretamente às rotinas gráficas, sem necessidade de manipulação alguma. A velocidade obtida através dessa estratégia a torna útil para os casos em que muitos resultados tractográficos precisam ser exibidos em um curto intervalo de tempo. Ela é utilizada, portanto, quando o usuário está arrastando o volume de interesse ou





**Figura 5.5:** Trajetórias renderizadas por meio de (a) linhas e (b) tubos. As linhas são usadas quando muitos resultados tractográficos precisam ser exibidos em um curto intervalo de tempo, e os tubos são usados nas outras situações.

quando está deslizando um dos controles dos critérios de parada.

A segunda estratégia mostra as trajetórias como minúsculos tubos. Ela é mais lenta que a primeira porque cada segmento de reta precisa ser transformado em um cilindro e individualmente desenhado. Essa estratégia é utilizada para exibir as trajetórias assim que o usuário pára de arrastar o volume de interesse ou deslizar os controles, o que acontece quando o usuário solta o botão do mouse que foi pressionado para iniciar a operação.

Exibir as trajetórias como tubos melhora sua visualização de várias maneiras. As trajetórias tornam-se mais visíveis que as linhas, pois essas últimas sempre tem a largura de um pixel na tela. Os cilindros também passam a interagir melhor com as luzes do ambiente tridimensional, refletindo a luz que se lhes incide. Por fim, é mais fácil visualizar grupos de trajetórias quando elas são exibidas como tubos.

## 5.4 Experimentos

Os objetivos dos experimentos deste capítulo são, no contexto da ferramenta interativa de tractografia:

- Determinar a velocidade com que as GPUs e a CPU produzem resultados tractográficos;
- Averiguar a influência dos conjuntos de dados sobre o desempenho;
- Averiguar como o tempo de execução se comporta à medida que o número de pontos semente dentro do volume de interesse aumenta;
- Determinar em que situações o tempo de execução se enquadra no conceito de tempo real, definido na introdução.

Para atingir esses objetivos, é preciso que o volume de interesse seja arrastado na tela. Uma estratégia seria que usuários arrastassem, através do mouse, o volume de interesse, enquanto a ferramenta registraria os tempos de execução de cada chamada à biblioteca tractográfica. Solicitar-se-ia do usuário, nessa estratégia, que ele abrisse diferentes conjuntos de dados e que usasse diferentes quantidades de pontos semente. Dessa forma, após o usuário ter usado a ferramenta por um determinado tempo, os dados coletados poderiam ser analisados e os objetivos descritos acima atingidos.

Essa estratégia, entretanto, não é sistemática o suficiente para gerar resultados confiáveis e reproduzíveis. Não se pode esperar, por exemplo, que o usuário arraste o volume de interesse exatamente pelos mesmos locais mais de uma vez, ou que o faça movimentando o mouse com a mesma velocidade. Por esse motivo, foi definido um procedimento automático que faz as vezes do usuário.

### 5.4.1 O procedimento automático

O procedimento automático consiste nos seguintes passos:

1. O volume de interesse tem seu tamanho ajustado para 10% do tamanho total do volume. Dessa forma, o volume de interesse passa a ter as mesmas proporções dos volumes DT-MRI originais, porém com uma fração (0,1%) de seu volume físico.
2. O volume de interesse é posicionado na parte inferior do eixo  $z$  e no centro dos eixos  $x$  e  $y$ .
3. Os resultados tractográficos são calculados de acordo com os parâmetros atuais, e os resultados são exibidos na tela.
4. O volume de interesse é movido para cima, ao longo do eixo  $z$ , na proporção de 0,9% da altura do volume.
5. Os passos 3–4 são executados outras 99 vezes, totalizando 100 execuções da tractografia e levando o volume de interesse do início ao fim do eixo  $z$ .

A execução desse procedimento é denominada aqui varredura. Ainda que as varreduras não imitem o comportamento de um usuário, elas podem ser usadas para atingir os objetivos definidos acima de uma maneira consistente e reproduzível.

### 5.4.2 Estrutura dos experimentos

Os experimentos deste capítulo consistem na execução de múltiplas varreduras, explorando a diversidade de configurações e conjuntos de dados disponíveis, sendo variado também o número de pontos semente. As varreduras foram feitas combinando-se:

- 4 conjuntos de dados, CD1, CD2, CD3 e CD4;
- 2 tratamentos, CPU e GPU;
- 4 unidades experimentais, CFG1, CFG2, CFG3 e CFG4;
- 40 densidades de pontos semente para GPU e 20 para CPU;

**Tabela 5.1:** Desempenho das CPUs nas diferentes configurações. O tempo está expresso em segundos. Os números em negrito indicam a configuração mais rápida para cada conjunto de dados.

	CFG1	CFG2	CFG3	CFG4	Média	Mín.	Máx.	Máx./Mín.
CD1	794,48	790,61	<b>790,34</b>	800,13	793,89	790,34	800,13	1,24%
CD2	2683,64	<b>2642,78</b>	2644,46	2666,03	2659,23	2642,78	2683,64	1,55%
CD3	3014,11	<b>2983,50</b>	2989,85	3012,77	3000,06	2983,50	3014,11	1,03%
CD4	1854,84	1819,41	<b>1818,05</b>	1848,94	1835,31	1818,05	1854,84	2,02%
Total	8347,07	<b>8236,30</b>	8242,69	8327,87	8288,48	8236,30	8347,07	1,34%

- 8 replicações.

Todos equipamentos e conjuntos de dados utilizados nos experimentos estão descritos no Apêndice A. Um número menor de densidades foi empregado com a CPU para que o tempo de execução dos experimentos não fosse excessivamente longo.

## 5.5 Resultados

### 5.5.1 Similaridade dos resultados das CPUs

Apesar de as CPUs de todas as configurações serem idênticas, os resultados dos experimentos desse capítulo dependem da placa de vídeo das configurações. Ao contrário dos experimentos do capítulo anterior, em que a tractografia executada na CPU não tinha relação alguma com a GPU instalada, neste capítulo a GPU torna-se relevante porque os resultados tractográficos, sejam eles calculados pela CPU ou pela GPU, são exibidos na tela. Como a exibição de gráficos na tela é função desempenhada pela GPU, os resultados dos experimentos com CPU dependem da GPU utilizada.

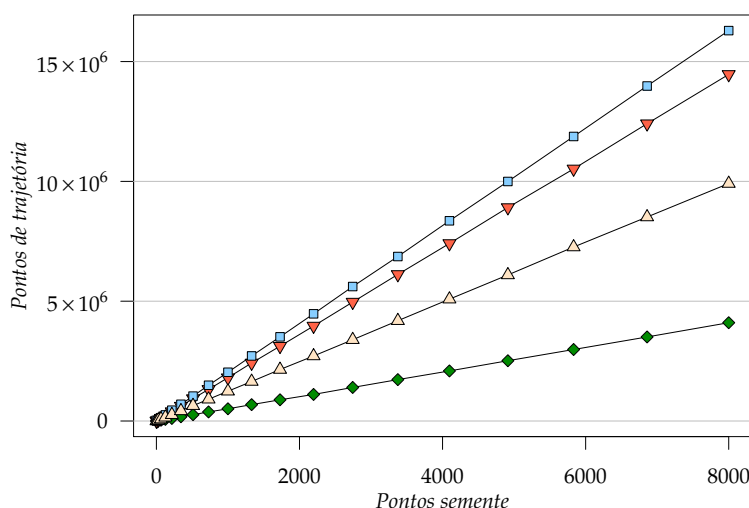
O tempo total de execução das CPUs das diferentes configurações para cada um dos conjuntos de dados é dado na Tabela 5.1. A diferença entre os tempos de execução de CPUs de configurações diferentes é, no caso mais divergente, de 2,02%. Por se tratar de uma diferença pequena quando comparada com as diferenças de desempenho entre CPUs e GPUs, não se justifica incluir individualmente nas demais tabelas e gráficos deste capítulo cada uma das CPUs. Relatar os resultados das CPUs das quatro configurações como um único resultado torna a apresentação e a interpretação dos gráficos e tabelas mais simples, sem perda substancial de conteúdo.

Para obter um resultado de CPU único a partir das quatro configurações disponíveis, duas alternativas são possíveis: combinar os resultados através de média aritmética ou apontar um dos resultados como o representante das CPUs de todas as configurações. Como as comparações entre GPU e CPU deste trabalho como um todo apontam vantagem significativa para as GPUs, optou-se por escolher a configuração que obteve o melhor resultado. Como mostra a Tabela 5.1, a configuração com o melhor tempo de execução é a CFG2, que na média foi 1,34% mais rápida que a mais lenta das configurações, a CFG1.

Os resultados deste capítulo, portanto, levam em consideração cinco plataformas: quatro GPUs (GPU1, GPU2, GPU3 e GPU4) e uma única CPU, que corresponde à configuração CFG2.

**Tabela 5.2:** Cores, símbolos e dimensões dos conjuntos de dados.

Nome	Cor	Símbolo	Dimensões
CD1	■	◆	$128 \times 128 \times 19$
CD2	■	▼	$128 \times 128 \times 33$
CD3	■	□	$128 \times 128 \times 40$
CD4	■	△	$128 \times 128 \times 70$



**Figura 5.6:** Pontos de trajetória de acordo com o número de pontos semente. O número de pontos de trajetória é o total encontrado por uma varredura. O número de pontos de trajetória encontrados pelo software de tractografia depende do conjunto de dados.

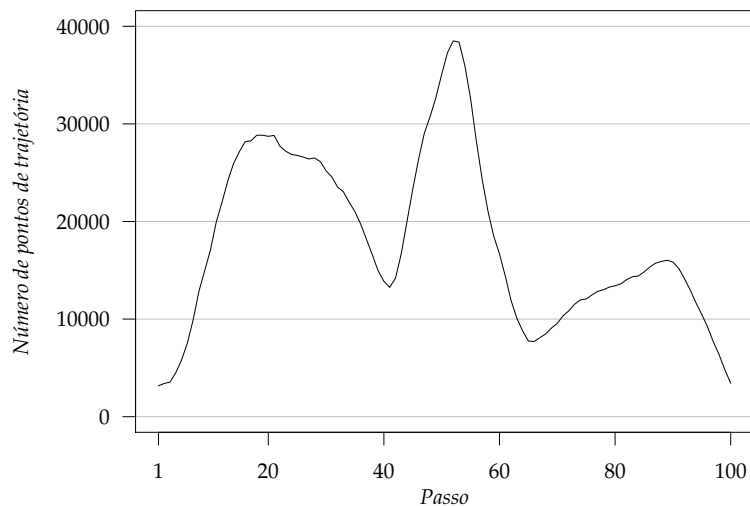
### 5.5.2 Conjuntos de dados

Os conjuntos de dados utilizados nos experimentos deste capítulo são os mesmos do capítulo anterior e também influenciam de maneira similar o desempenho do software de tractografia. As dimensões dos conjuntos de dados e as cores e símbolos usados para representá-los nos gráficos são dados pela Tabela 5.2.

A Figura 5.6 mostra a quantidade de pontos de trajetória encontrados pelo software de tractografia de acordo com o número de pontos semente, até o máximo de  $20 \times 20 \times 20 = 8.000$  pontos semente. O número de pontos de trajetória é o total para a varredura de 100 passos. Observa-se que, para qualquer número de pontos semente, alguns conjuntos de dados sempre produzem mais pontos de trajetória do que outros. O CD1, por exemplo, sempre gera aproximadamente 4 vezes menos pontos de trajetória do que o CD3.

O número de pontos de trajetória dos resultados tractográficos não varia apenas de um conjunto de dados para outro, mas também de uma região para outra dentro de um mesmo volume. Naturalmente, um volume de interesse posicionado fora do corpo do paciente não produzirá nenhum resultado (à exceção de resultados induzidos por ruído), ao passo que um volume posicionado no corpo caloso produzirá muitos resultados.

Essa variação da quantidade de resultados em diferentes regiões do volume está ilustrada na Figura 5.7. Esta figura mostra o número de pontos encontrados em cada um dos 100 passos da varredura do CD2 com  $10 \times 10 \times 10 = 1.000$  pontos semente. O pico logo após o passo 50, em que em único passo quase 40.000 pontos



**Figura 5.7:** Número de pontos de trajetória encontrados na varredura do CD2 com 1.000 pontos semente de acordo com o passo da varredura.

são encontrados, corresponde à região do corpo caloso.

É importante, em suma, levar em consideração essa variação dentro de cada conjunto de dados ao interpretar os resultados deste capítulo, pois apenas valores totais ou índices derivados dos valores totais são apresentados. Quando se diz, por exemplo, que uma varredura completa levou 100 segundos, ficam convenientemente abstraídas as diferenças entre os passos da varredura.

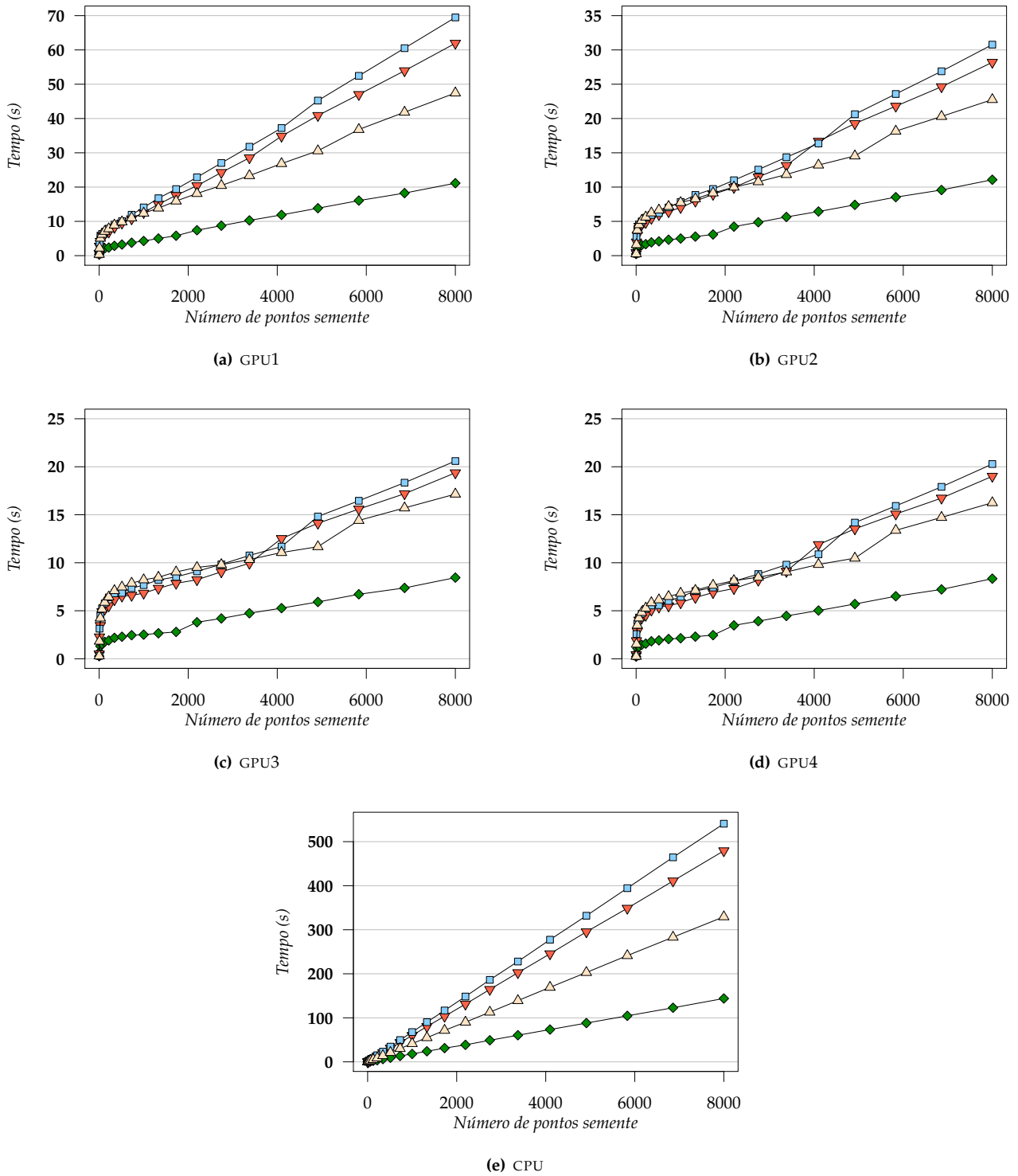
### 5.5.3 Tempo de execução

Os resultados dos experimentos com varreduras de até 8.000 pontos semente são exibidos na Figura 5.8. Cada gráfico corresponde a uma das cinco plataformas. Atente-se para o fato de que as escalas do eixo  $y$  não são as mesmas para todos os gráficos — de outro modo os dados dos gráficos das GPUs ficariam muito concentrados e pouco informativos, já que a escala da CPU passa de 500 s e nenhuma das GPUs chega sequer a 100 s.

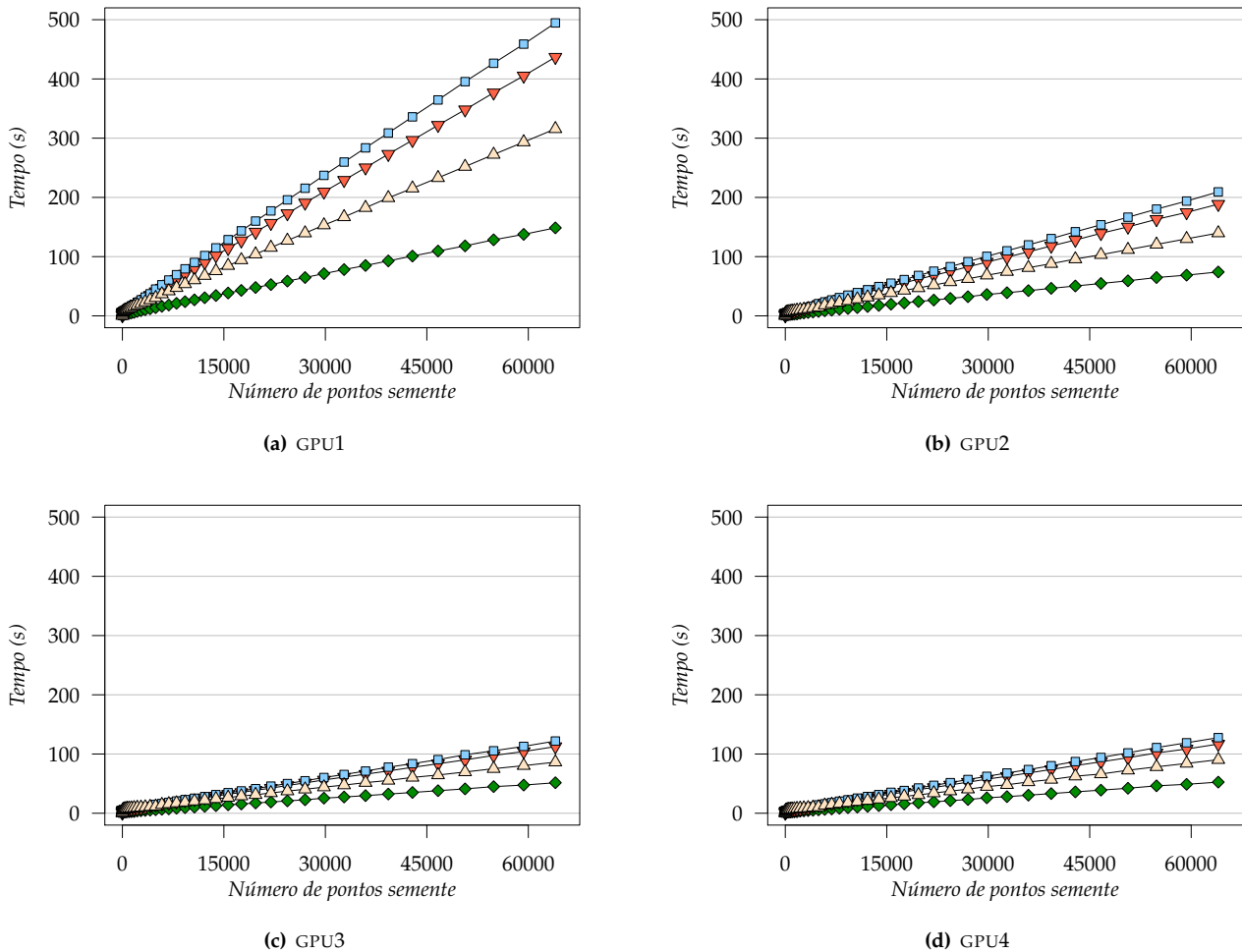
O gráfico da CPU na Figura 5.8 exibe um comportamento altamente linear; o tempo de execução da tractografia é diretamente proporcional ao número de pontos semente. Os gráficos das GPUs, contudo, exibem apenas uma tendência linear e, ainda assim, somente com mais de 1.000 pontos semente. Com menos de 1.000 pontos semente, o tempo de execução das GPUs cresce muito rápido antes de assumir um comportamento mais linear.

Essa não-linearidade das GPUs é produzida pela arquitetura que elas utilizam. A execução da tractografia nas GPUs é baseada em uma estrutura de gerenciamento complexa que influencia o tempo de execução de maneiras difíceis de serem previstas. A CPU, por outro lado, não precisa lidar com uma grande quantidade de threads executando paralelamente e, por essa razão, exibe um comportamento mais previsível.

Mais irregular que essa não-linearidade inicial nos gráficos das GPUs é o pequeno salto no tempo de execução que acontece, também com as GPUs, em todos os conjuntos de dados. O salto acontece em uma região diferente do número de pontos semente para cada um dos conjuntos de dados, ainda que essa região não varie em função das GPUs. Esse comportamento esdrúxulo é menos visível na GPU1. Ainda que não haja nesse



**Figura 5.8:** Tempo de execução das varreduras com até 8.000 pontos semente para as cinco plataformas em função do número de pontos semente. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

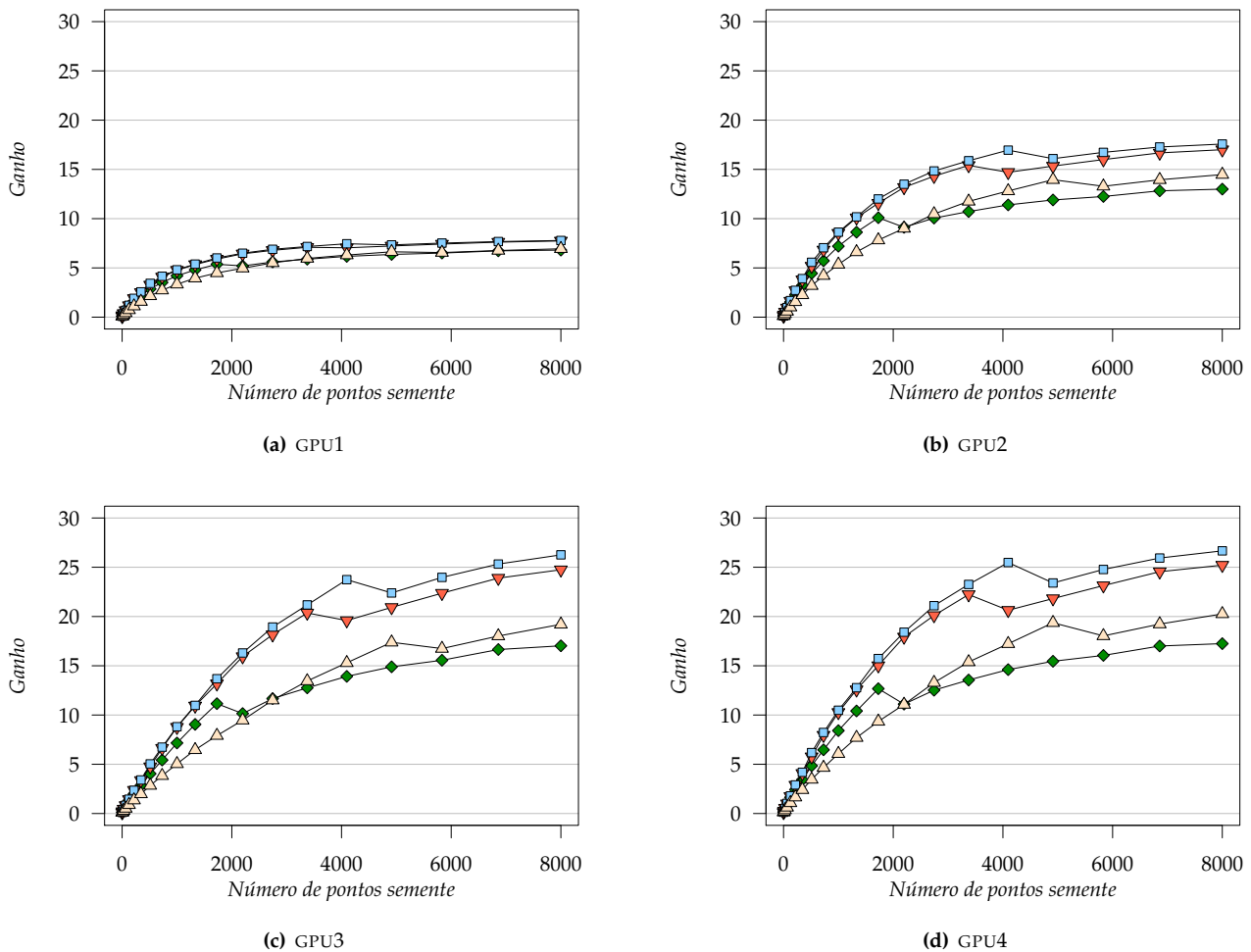


**Figura 5.9:** Tempo de execução das varreduras com até 64.000 pontos semente para as quatro GPUs em função do número de pontos semente. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

momento uma explicação convincente para o fenômeno, observa-se que sua ocorrência parece estar ligada às dimensões do conjunto de dados: quanto maiores suas dimensões, mais tarde o fenômeno manifesta-se. Desse modo, o salto acontece primeiro com o CD1 e em seguida com o CD2, CD3 e CD4.

As varreduras com mais de 8.000 sementes têm seus resultados exibidos na Figura 5.9. Apenas experimentos com GPUs foram conduzidos para essas varreduras, pois eles levariam muito tempo para serem executados com a CPU. De qualquer forma, o comportamento bastante linear da CPU observado na Figura 5.8 permite que sejam estimados tempos de execução para grandes quantidades de pontos através de regressão linear. Nessa figura, as escalas do eixo  $y$  são idênticas.

A Figura 5.9 mostra que, apesar da não-linearidade observada com um número reduzido de pontos semente e do pequeno salto, as GPUs apresentam um comportamento bastante linear quando são levadas em consideração grandes quantidades de pontos semente. Nestes gráficos fica clara a superioridade das GPUs 3 e 4, que conseguiram realizar uma varredura com 64.000 pontos semente em pouco mais de dois minutos.



**Figura 5.10:** Ganho de desempenho obtido pelas GPUs em comparação com a CPU. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

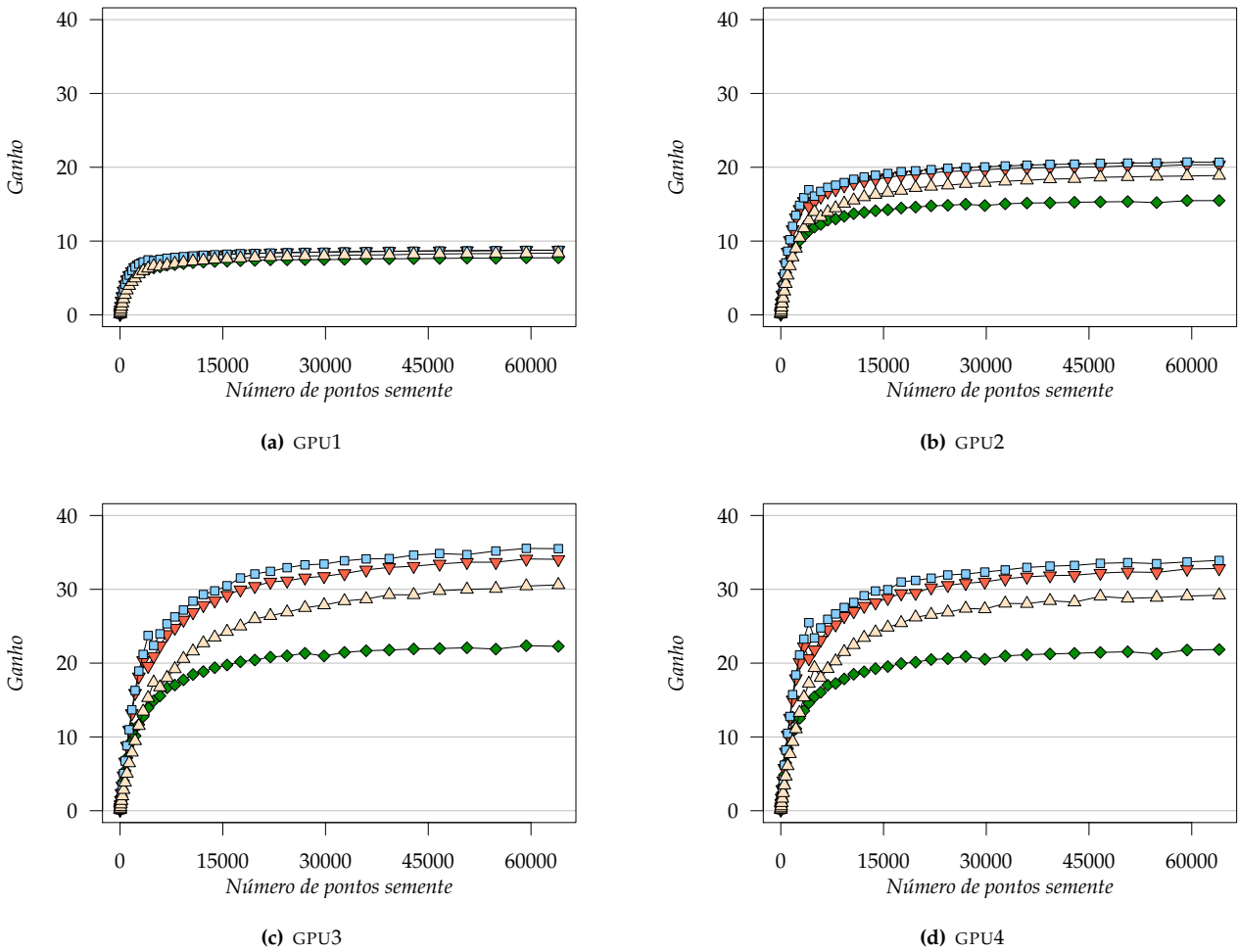
### 5.5.4 Ganho de desempenho

A relação entre os tempos de execução da CPU e das quatro GPUs é exibida nos gráficos da Figura 5.10. O valor exibido nos gráficos é o tempo de execução das varreduras da CPU dividido pelo tempo de execução das varreduras correspondentes das GPUs. Desta forma, os gráficos dessa figura representam o ganho de desempenho obtido pelas GPUs.

Percebe-se que, à exceção da GPU1, as demais GPUs não chegam a um ponto em que o ganho de desempenho fica fixo em um valor. Ao invés disso, o ganho continua crescendo lentamente. Isso indica que, até 8.000 pontos sementes, as GPUs 2, 3 e 4 ainda não alcançam seu potencial máximo de processamento. A GPU1, ao contrário, já tem um ganho de desempenho sobre a CPU mais bem definido que as outras ao atingirem os 8.000 pontos.

Como a CPU apresenta um comportamento suficientemente linear, pode-se extrapolar seu tempo de execução para além dos 8.000 pontos e aproximar o ganho de desempenho das GPUs. O resultado desse novo cálculo está nos gráficos da Figura 5.11. Assumindo que a CPU se comportaria de acordo com a regressão linear, vê-se que o ganho de desempenho de todas as GPUs já estariam praticamente estabilizados ao atingirem os





**Figura 5.11:** Ganho de desempenho obtido pelas GPUs em comparação com a CPU, incluídos os valores aproximados por regressão linear para varreduras com mais de 8.000 pontos no caso da CPU. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

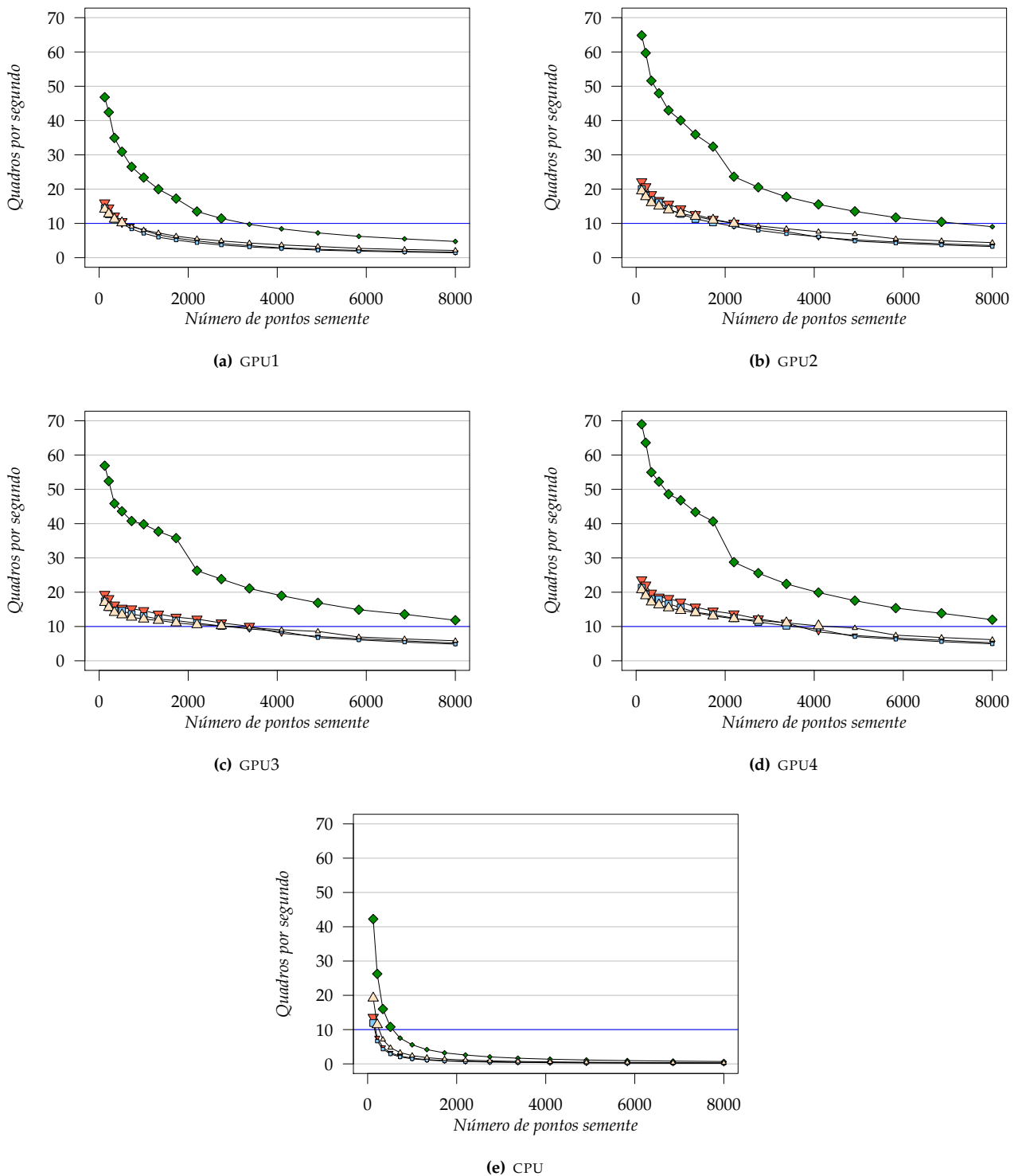
64.000 pontos.

O ganho de desempenho das GPUs, com 8.000 e 64.000 pontos semente (com regressão linear nesse último caso), pode ser visto na Tabela 5.3.

### 5.5.5 Taxa de quadros por segundo

A Figura 5.12 mostra o número de quadros por segundo obtido por cada uma das plataformas com todos os conjuntos de dados. Somente varreduras com  $5 \times 5 \times 5 = 125$  ou mais pontos semente são consideradas, pois quantidades menores são processadas muito rapidamente e a alta taxa resultante de quadros por segundo tornaria os gráficos pouco informativos a respeito das demais. Por exemplo, fazendo uma varredura com apenas 1 ponto semente a CPU atinge uma taxa de quase 3.500 quadros por segundo. Ajustar a escala dos gráficos a um valor tão elevado lançaria os outros valores, bem menores, muito para baixo, o que os tornaria indistintos.

Adotando-se a definição de tempo real apresentada na introdução, as plataformas precisam sustentar uma



**Figura 5.12:** Número de quadros por segundo obtido por cada uma das plataformas com todos os conjuntos de dados a partir de  $5 \times 5 \times 5 = 125$  pontos semente. A linha horizontal marca o limiar de 10 quadros por segundo. Os marcadores de tamanho reduzido indicam valores abaixo desse limiar. Os conjuntos de dados são  $\blacklozenge$ =CD1,  $\blacktriangledown$ =CD2,  $\blacksquare$ =CD3 e  $\blacktriangle$ =CD4.

**Tabela 5.3:** Ganho de desempenho obtido pelas GPUs em comparação com a CPU nas varreduras de 8.000 pontos semente e 64.000 pontos semente. Os valores da CPU utilizados para o cômputo do ganho com 64.000 pontos semente foram obtidos através de regressão linear.

	8.000 pontos semente				64.000 pontos semente			
	GPU1	GPU2	GPU3	GPU4	GPU1	GPU2	GPU3	GPU4
CD1	6,81	13,00	17,03	<b>17,25</b>	7,73	15,49	<b>22,28</b>	21,84
CD2	7,74	17,00	24,75	<b>25,21</b>	8,78	20,33	<b>34,06</b>	32,82
CD3	7,78	17,57	26,26	<b>26,66</b>	8,75	20,68	<b>35,50</b>	33,91
CD4	6,94	14,47	19,21	<b>20,27</b>	8,36	18,88	<b>30,60</b>	29,20

**Tabela 5.4:** Número máximo de pontos semente com a qual cada plataforma é capaz de manter uma taxa de no mínimo 10 quadros por segundo em cada um dos conjuntos de dados.

	CPU	GPU1	GPU2	GPU3	GPU4
CD1	512	2744	6859	<b>9261</b>	<b>9261</b>
CD2	125	512	2197	<b>3375</b>	<b>3375</b>
CD3	125	343	1728	2744	<b>3375</b>
CD4	216	512	2197	2744	<b>4096</b>

taxa de no mínimo 10 quadros por segundo para fornecer resultados tractográficos em tempo real. A linha horizontal marca esse limiar nos gráficos da Figura 5.12. Para facilitar a identificação dos valores que se encontram abaixo desse limiar, seus marcadores tiveram o tamanho reduzido.

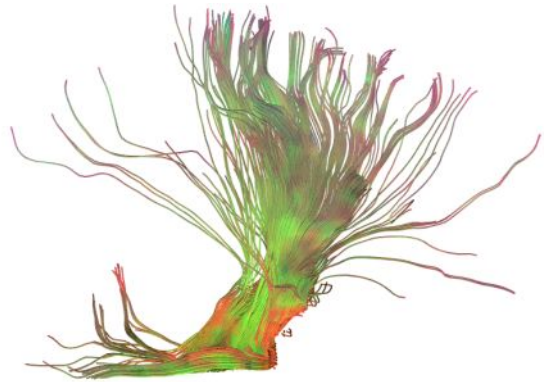
Os gráficos da Figura 5.12 mostram, essencialmente, até qual quantidade de pontos semente uma plataforma é capaz de manter uma taxa de quadros por segundo igual ou superior a 10. Dessa forma, percebe-se que a CPU é capaz de fazê-lo somente para pequenas quantidades de pontos semente, enquanto a GPU3 e a GPU4 fazem-no para quantidades significativamente maiores.

A Tabela 5.4 mostra a máxima quantidade de pontos semente em que cada plataforma ainda é capaz de manter uma taxa de quadros por segundo superior a 10. Percebe-se que a GPU4, para todos os conjuntos de dados, conseguiu sustentar uma taxa de pelo menos 10 quadros por segundo com  $15 \times 15 \times 15 = 3.375$  pontos semente. Em contrapartida, a CPU manteve essa taxa com no máximo  $5 \times 5 \times 5 = 125$  pontos semente para os conjuntos de dados 2 e 3.

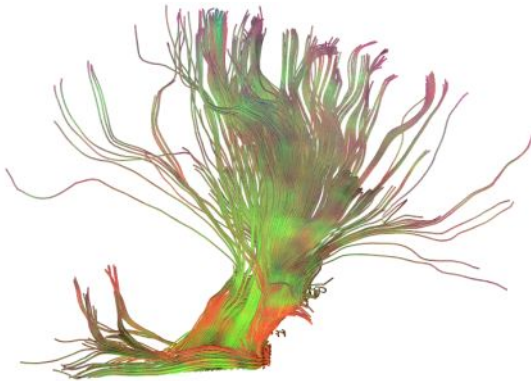
A Figura 5.13 mostra resultados tractográficos obtidos com a quantidade máxima de pontos semente com a qual cada uma das plataforma ainda conseguiu sustentar a taxa de no mínimo 10 quadros por segundo no CD3. Esse conjunto de dados foi escolhido por ser o conjunto que impõe o maior trabalho às plataformas, de acordo com a Tabela 5.4.



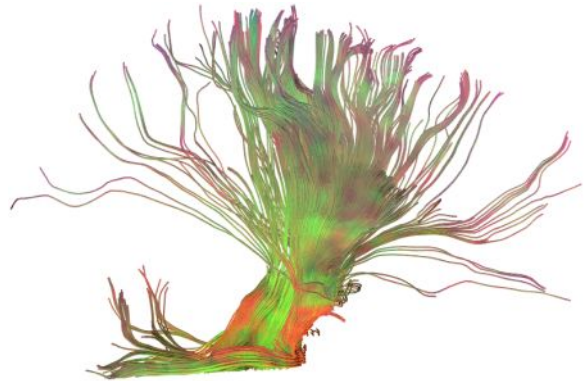
(a)  $7 \times 7 \times 7 = 343$  pontos semente, máximo para a GPU1



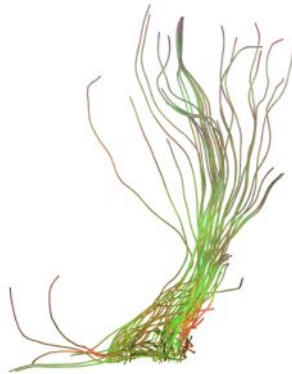
(b)  $12 \times 12 \times 12 = 1.728$  pontos semente, máximo para a GPU2



(c)  $14 \times 14 \times 14 = 2.744$  pontos semente, máximo para a GPU3



(d)  $15 \times 15 \times 15 = 3.375$  pontos semente, máximo para a GPU4



(e)  $5 \times 5 \times 5 = 125$  pontos semente, máximo para a CPU

**Figura 5.13:** Resultados tractográficos obtidos com a quantidade máxima de pontos semente com a qual cada plataforma é capaz de manter uma taxa de no mínimo 10 quadros por segundo no CD3. Todos resultados foram obtidos com o mesmo volume de interesse.

# Capítulo 6

## Conclusão

As unidades de processamento gráfico oferecem um ambiente de execução paralela eficiente e de baixo custo. Este trabalho mostra que a tractografia por propagação de linhas, ao ser executada nesse ambiente, pode obter um ganho significativo de desempenho e, em muitas circunstâncias, ser executada em tempo real.

Nas seções a seguir, as principais contribuições do trabalho são apresentadas, seguidas de uma discussão estimulada pelos resultados dos experimentos. As limitações da pesquisa realizada e propostas de como superá-las e de trabalhos futuros também são relacionadas.

### 6.1 Contribuições

Apesar de a tractografia por propagação de linhas ter uma unidade básica de paralelismo facilmente identificável, isto é, as trajetórias calculadas a partir dos pontos semente, é necessária uma estratégia que explore esse paralelismo inerente de forma eficiente, tanto no que tange ao desempenho quanto ao uso da memória. Esse trabalho apresenta uma estratégia em que a GPU e a CPU cooperam de maneira estreita: a GPU calcula um único ponto de cada trajetória e a CPU volta a chamar a GPU apenas para as trajetórias não finalizadas. O uso da memória fica, dessa forma, reduzido ao mínimo necessário. Os experimentos demonstram, ademais, que empregando essa estratégia simples as GPUs conseguem executar a tractografia em um tempo significativamente breve, especialmente quando esse tempo é comparado àquele obtido pelas CPUs.

Os experimentos do Capítulo 5 trazem resultados que quantificam o tempo de execução da tractografia em quatro conjuntos de dados diferentes. O ganho de desempenho obtido pelas GPUs com relação à CPU depende, é evidente, da CPU e das GPUs específicas que foram utilizadas nos experimentos. Interpretando-se os resultados, portanto, não se pode afirmar de maneira absoluta que as GPUs são  $n$  vezes mais rápidas que as CPUs, nem tampouco é possível dizer que os valores obtidos representam uma média significativa. É possível, contudo, afirmar que as GPUs atuais são significativamente mais rápidas que as CPUs atuais. Não se pode ignorar, com efeito, o fato de que a adição de uma placa de vídeo que custa menos de 100 dólares a uma configuração com um processador mediano, tipicamente encontrado nas estações de trabalho, resulta em um ganho de desempenho de mais de 30 vezes. Uma das contribuições deste trabalho é, desse modo, dar uma idéia bastante concreta do ganho de desempenho que se obtém ao utilizar as GPUs para a tractografia.

Outra contribuição do trabalho é a definição e implementação de um novo paradigma de interatividade para a tractografia por propagação de linhas. Não foi encontrada na literatura uma proposta de interatividade em que os resultados tractográficos sejam computados em tempo real em resposta a qualquer ação do usuário. No paradigma de atualidade permanente, proposto neste trabalho, as ações do usuário na interface gráfica — ajustar o valor de um parâmetro do método de tractografia, definir o tamanho do volume de interesse ou arrastá-lo na tela — fazem com que novos resultados sejam calculados e exibidos na tela em tempo real. É importante ressaltar que, em contraste com outras abordagens, nessa proposta os resultados são sempre calculados a partir do zero, isto é, não há necessidade de resultados pré-computados estarem disponíveis. Também digno de nota é o fato de que no paradigma de atualidade permanente os resultados são atualizados não apenas *após* a solicitação do usuário, mas também *enquanto* ela acontece: ao arrastar o volume de interesse na tela, os resultados são recalculados para cada posição nova que o ponteiro do mouse assume na tela; ao deslizar os controles que ajustam os valores mínimos de anisotropia fracional e difusividade média, os resultados são atualizados para cada valor intermediário que o controle assume na interface gráfica.

A qualificação de “tempo real” requer que o número de quadros por segundo que a ferramenta tractográfica atinge enquanto o usuário realiza uma ação na interface gráfica seja medido. Não faria sentido, pois, dizer que a tractografia está sendo executada em tempo real se ela leva vários segundos para produzir seus resultados. Para medir o número de quadros por segundo, este trabalho propõe um procedimento automático que move o volume de interesse na tela, como se o usuário o fizesse. Apesar de o movimento ser bastante simples (percorrer o volume seguindo um eixo vertical) e não reproduzir o comportamento de um usuário real, esse procedimento tem a vantagem de ser reproduzível, o que permite a comparação de tempos de execução entre plataformas diferentes.

Os resultados dos experimentos do Capítulo 5 são especialmente relevantes, pois mostram como o tempo de execução, e portanto o ganho de desempenho e o número de quadros por segundo, varia conforme o número de pontos semente dentro do volume interesse cresce. A partir desses resultados, fica evidente que o comportamento da CPU é bastante linear e previsível: o tempo de execução é função linear do número de pontos semente. As GPUs por outro lado, só apresentam um comportamento linear para um número elevado de pontos semente. Com menos de 1.000 pontos, o tempo de execução cresce rapidamente e, antes de atingir os 8.000 pontos, passa por um sobressalto de origem desconhecida. Assim, além de mostrar como a CPU e as GPUs se comportam conforme a carga de trabalho aumenta, este trabalho também fornece uma boa estimativa para a maior carga de trabalho com que a CPU e a GPU ainda conseguem produzir resultados em tempo real, isto é, em uma taxa superior a 10 quadros por segundo. Os resultados tractográficos obtidos com essa carga máxima de trabalho, exibidos no Capítulo 6, mostram que a melhor GPU utilizada é capaz de realizar a tractografia em tempo real com uma carga de trabalho cujos resultados são abundantes.

## 6.2 Discussão

O ganho de desempenho obtido com a implementação em GPU da tractografia por propagação de linhas é fruto de algumas características do problema. Primeiramente, a tractografia é facilmente paralelizável. Os pontos de cada trajetória podem ser calculados de maneira totalmente independente dos demais, o que permite que

várias trajetórias sejam calculadas simultaneamente. Como o número de pontos semente frequentemente está na ordem dos milhares, os processadores disponíveis na GPU em geral são bem aproveitados. Em segundo lugar, a tractografia demanda grande quantidade de computação. Para cada ponto de cada trajetória, várias diagonalizações de matrizes e cálculos com vetores precisam ser executados. Essas duas características — o paralelismo e a computação intensiva — são típicas de problemas que via de regra obtêm bons ganhos de desempenho na GPU.

Apesar dos ganhos significativos de desempenho, é comum que outras aplicações obtenham ganhos de desempenho ainda maiores. Um dos fatores que provavelmente impedem a tractografia de ser executada com um tempo inferior na GPU é a maneira como a memória é acessada. Programas de computador que utilizam a tecnologia CUDA conseguem acessar a memória com maior velocidade quando o fazem de maneira ordenada (coalescida). O principal acesso à memória feito pela tractografia, contudo, é ao campo discreto de tensores. Como as trajetórias seguem caminhos que o programa não pode prever, o acesso ao campo de tensores é feito de maneira bastante arbitrária, e por esse motivo dificilmente implementações futuras conseguirão fazer acessos à memória com um alto grau de ordenação.

A maneira como a pesquisa que originou esse trabalho se desenrolou mostra que as GPUs são uma tecnologia *habilitadora*. Inicialmente, a tractografia em GPU foi escrita para se obter resultados mais rapidamente. Porém, uma vez que a tractografia passa a ser executada mais de 30 vezes mais rápido que previamente, novas possibilidades são reveladas. Desta forma, o paradigma de atualidade permanente surgiu somente após a tractografia ter sido implementada em GPU — antes disso, a implementação em CPU era muito lenta para sugerir que uma abordagem assim seria vantajosa. De maneira similar, a equalização dos parâmetros do método de tractografia só passou a ter razão de ser quando os resultados tractográficos eram atualizados na tela rápido o suficiente para que se percebesse que umas regiões do espaço dos parâmetros eram mais sensíveis que outras.

Os bons ganhos de desempenho que o uso de GPUs de baixo custo possibilitam, todavia, exigem um esforço significativo de implementação. Não se trata apenas de um esforço para a familiarização com conceitos diferentes daqueles empregados tradicionalmente; é preciso, de fato, enfrentar vários problemas de ordem técnica. A compilação de programas frequentemente leva um longo tempo para ser concluída, atualizações de driver podem ter um impacto grande sobre o desempenho dos programas (diferenças de até 13% foram observadas nesta pesquisa), os erros são difíceis de serem localizados, resultados inesperados e excêntricos são produzidos. Esses problemas todos são consequência, provavelmente, das condições impostas pelo mercado: o fabricante que lança primeiro uma tecnologia assegura a liderança, mesmo que essa tecnologia ainda não esteja bem desenvolvida. O contraste entre escrever um programa para GPUs e um programa para clusters, por exemplo, é grande, como argumenta-se alhures (Mittmann et al., 2009).

### 6.3 Limitações e trabalhos futuros

Todas as contribuições deste trabalho foram feitas no contexto da tractografia *por propagação de linhas*. Esse é o método mais tradicional de se localizar as fibras nervosas no cérebro humano, mas não é de maneira alguma o único. Existem outras propostas que tentam ser imunes às limitações do método de propagação de linhas, em particular à sua incapacidade de seguir múltiplas direções de difusão, uma consequência direta

do modelo de tensor adotado. Uma estrutura de software para GPUs que generalize mais de um método de tractografia ou que ao menos forneça os elementos fundamentais para sua coexistência tornaria possível ferramentas de tractografia altamente flexíveis, além de possibilitar comparações quantitativas e qualitativas entre diferentes abordagens à tractografia. Também relevante seria o modelo de interatividade para uma tal ferramenta unificada de tractografia.

Ainda na tractografia por propagação de linhas, a estratégia de paralelização proposta neste trabalho, apesar de obter êxito no que se propõe a fazer, pode ser otimizada. Um mecanismo que permita o cálculo e exibição dos resultados sem necessidade de intermédio da CPU eliminaria o tempo de comunicação e, portanto, melhoraria o desempenho. Um trabalho publicado recentemente (Köhn et al., 2009) traz novas idéias nesse contexto e pode ser um bom caminho para evitar a comunicação desnecessária entre CPU e GPU.

O procedimento automático definido para medir o desempenho da tractografia na ferramenta gráfica cobre apenas um eixo vertical do volume. Tipicamente, o procedimento passa por regiões importantes do cérebro e produz medições significativas, mas ele passa por apenas 1% do volume. Por outro lado, medir a taxa de quadros por segundo em 100% do volume faria menos sentido, já que em exames de ressonância magnética o corpo do paciente ocupa apenas uma fração do volume total. Não é interessante saber se uma implementação de tractografia gera resultados em tempo real em uma região vazia do espaço. Uma estratégia melhor seria, portanto, percorrer somente as regiões relevantes do volume para estimar a taxa de quadros por segundo. Uma segmentação do cérebro poderia revelar quais regiões são relevantes.

Uma limitação importante deste trabalho é a ausência de avaliações e critérios qualitativos para as ferramentas e métodos apresentados. Um estudo que mostre como os softwares de tractografia são utilizados por usuários poderia, por exemplo, guiar melhor a exploração de novos meios de interatividade. Conforme o poder computacional disponível nas estações de trabalhos aumentam, seja através da CPU ou da GPU, o tempo de resposta passa a ser suficientemente pequeno e as questões qualitativas passam a merecer maior destaque.

Uma avaliação qualitativa também seria uma maneira factível de serem comparadas diferentes ferramentas interativas de tractografia. As medições dos experimentos deste trabalho podem ser feitas em outras ferramentas, *contanto que* utilizem paradigmas de interatividade suficientemente semelhantes. Esse, no entanto, não é o caso para as ferramentas atuais. Uma comparação entre diferentes ferramentas de tractografia tem de ser, neste momento, de natureza qualitativa.

Uma das premissas dos experimentos do Capítulo 5, sobre a tractografia em tempo real, é que um número de pontos semente maior produz uma visualização melhor, mais informativa para o usuário. Deve existir um limite, entretanto, além do qual aumentar o número de pontos semente não gera resultados melhores para um dado volume de interesse. Determinar esse limite significa poder afirmar de uma maneira mais absoluta que uma ferramenta de tractografia opera em tempo real ou não. Se, por exemplo, esse limite for 10.000 pontos, seria possível mostrar que uma ferramenta executa a tractografia com 10.000 pontos em todas as regiões do cérebro de vários volumes em tempo real, o que equivaleria a dizer que na prática ela funciona sempre em tempo real. Novamente, experimentos qualitativos poderiam revelar se esse limite existe de fato e qual é o seu valor.

A implementação da tractografia por propagação de linhas apresentada neste trabalho utiliza a tecnologia CUDA para obter acesso aos recursos computacionais da placa de vídeo. Essa tecnologia, entretanto, é espe-



cífica de um fabricante de placas de vídeo, a NVIDIA. Apesar de ser a mais madura disponível atualmente, outros concorrentes têm propostas similares e em diferentes estágios de desenvolvimento. Neste momento, não está claro se é possível uma implementação de um método complexo como a tractografia que possa ser executada em placas de vídeo de fabricantes distintos. A situação é especialmente complicada por envolver uma área em que propostas de linguagens e tecnologias surgem com bastante frequência.

Neste trabalho não foram abordadas as questões relativas à diferença numérica dos resultados produzidos pela CPU e pela GPU. Essa diferença existe por dois motivos: a maior parte das GPUs atuais não produzem resultados de acordo com o padrão relevante (IEEE, 1985) em 100% das operações matemáticas, enquanto as CPUs o fazem; e mesmo se as operações de ponto flutuante tivessem sempre o mesmo resultado em ambas plataformas, os programas de tractografia como um todo só gerariam precisamente o mesmo resultados se executassem todas as operações precisamente na mesma ordem. Como as implementações de GPU e CPU deste trabalho utilizam códigos distintos, isso dificilmente aconteceria. A questão da diferença entre os resultados, porém, foi discutida em um dos artigos publicados ao longo da pesquisa que originou este trabalho (Mittmann et al., 2008).

# Apêndice A

## Descrição de materiais

Dois computadores foram utilizados para a execução dos experimentos, ambos com configurações idênticas, à exceção das placas de vídeo, que foram instaladas conforme o experimento a ser realizado. A configuração básica e invariável de cada computador é dada na Tabela A.1. Computadores idênticos foram usados nos experimentos para minimizar o número de variáveis envolvidas na análise dos resultados.

Quatro placas de vídeos diferentes foram usadas, todas da fabricante NVIDIA e da série GeForce. As principais características dessas placas estão na Tabela A.2. As datas de lançamento foram obtidas diretamente da sala de imprensa da NVIDIA na internet (NVIDIA, 2008c, 2007a, 2006). As especificações técnicas também provêm da NVIDIA (NVIDIA, 2009a,b,c,f). Os preços atuais foram obtidos a partir da loja Newegg (Newegg, 2009c,b,a) por meio de seleção do menor preço disponível, considerando-se, contudo, apenas os produtos fabricados pelos chamados “parceiros de placa autorizados” da NVIDIA (NVIDIA, 2009e). A placa GeForce 8800 GTS, entretanto, está saindo de linha e não está disponível na loja Newegg. Cabe também lembrar que os valores de frequência da memória e dos processadores listados na tabela são os valores nominais divulgados pela NVIDIA; os fabricantes podem impor frequências diferentes (tipicamente mais altas) das nominais.

Muitas das razões que levaram à escolha dos computadores e das placas de vídeo foram de ordem prática. Nenhum equipamento foi comprado especialmente para a realização destes experimentos; ao contrário, computadores normalmente utilizados para outras tarefas foram temporariamente alocados para esse fim.

Também tendo em mente o objetivo de minimizar as variáveis que podem influenciar os experimentos, as mesmas versões dos mesmos softwares foram instalados em ambos computadores, inclusive — e principalmente — a mesma versão dos softwares desenvolvidos no Grupo Cyclops (Grupo Cyclops, 2009). Os principais softwares e as versões correspondentes constam da Tabela A.3, em ordem alfabética. Alguns dos softwares (wxWidgets e Estação Radiológica) são relevantes apenas para os experimentos do Capítulo 5.

Apesar de haver disponíveis versões mais recentes do driver de vídeo (no momento da escrita deste trabalho, a versão 180.29 era a última), a versão 173.08 foi utilizada por mostrar-se empiricamente mais apropriada que as outras: averiguou-se que o desempenho do software de tractografia era até 13% melhor com essa versão específica. O uso dessa versão também determinou a versão do CUDA Toolkit, pois versões mais recentes desse Toolkit exigem drivers também mais recentes; e o uso dessa versão do Toolkit requereu, por sua vez, o emprego de uma versão mais antiga (4.1.2) do compilador GCC em um dos passos da compilação.

**Tabela A.1:** Características de hardware dos dois computadores usados nos experimentos.

Componente	Especificação
Placa-mãe	ASUS M2N-E
Processador	Athlon XP 64 X2 Dual Core 4400+
Frequência	2300 MHz
Cache L1	128 KB × 2
Cache L2	512 KB × 2
Memória	2 GB
Tecnologia	DDR2
Frequência	800 MHz
Configuração	Single channel
Disco rígido	Seagate Barracuda 7200.10
Capacidade	320 GB

**Tabela A.2:** Principais características das placas de vídeo usadas nos experimentos. Todas são da série GeForce da NVIDIA.

	8500 GT	8600 GT	8800 GTS	9600 GT
Lançamento	17/04/2007	17/04/2007	08/11/2006	21/02/2008
Preço atual	US\$ 45,99	US\$ 74,99	–	US\$ 95,99
Memória	512 MB	256 MB	320 MB	512 MB
Frequência	400 MHz	700 MHz	800 MHz	900 MHz
Tecnologia	DDR2	GDDR3	GDDR3	GDDR3
Interface	128 bits	128 bits	320 bits	256 bits
Banda	12,8 GB/s	22,4 GB/S	64 GB/S	57,6 GB/S
Processadores	16	32	96	64
Frequência	900 MHz	1180 MHz	1200 MHz	1625 MHz

**Tabela A.3:** Versões dos softwares instalados nos dois computadores para os experimentos.

Software	Função	Versão
Software de DT-MRI	Biblioteca de tractografia	r13602
Estação radiológica	Estação radiológica	r2529
CUDA Toolkit	Compilador CUDA	1.1
Driver de vídeo	Interface para a placa de vídeo	173.08
Slamd64	Sistema operacional	12.1
GCC	Compilador C/C++	4.2.3
Linux	Núcleo	2.6.24.5
GSL	Biblioteca numérica	1.12
CMake	Construtor de software	2.6.2
wxWidgets	Biblioteca gráfica	2.8.9

Um total de quatro conjuntos de dados foram utilizados nos experimentos deste trabalho; suas principais características estão descritas na Tabela A.4. O conjunto 1 foi aleatoriamente escolhido dentre um total de 57 com as mesmas características.

**Tabela A.4:** Principais características dos conjuntos de dados DT-MRI usados nos experimentos.

	Conjunto 1	Conjunto 2
Equipamento	1.5 T MAGNETOM Vision	1.5 T MAGNETOM Sonata
Voxel	$1,80 \times 1,80 \times 5,00$ mm	$1,80 \times 1,80 \times 3,00$ mm
Dimensões	$128 \times 128 \times 19$	$128 \times 128 \times 33$
Espaçamento	1,50 mm	0
TE/TR	100/8.000 ms	105/8.000 ms
Valor de $b$	$900 \text{ s/mm}^2$	$1.000 \text{ s/mm}^2$
Direções	6	6
	Conjunto 3	Conjunto 4
Equipamento	1.5 T MAGNETOM Sonata	3 T MAGNETOM Trio
Voxel	$1,80 \times 1,80 \times 3,00$ mm	$1,90 \times 1,90 \times 1,90$ mm
Dimensões	$128 \times 128 \times 40$	$128 \times 128 \times 70$
Espaçamento	0	0
TE/TR	105/8.000 ms	118/11.800 ms
Valor de $b$	$1.000 \text{ s/mm}^2$	$1.000 \text{ s/mm}^2$
Direções	6	20

# Referências

- AMD, 2009a. AMD delivers and submits for certification industry's first OpenCL™ software development platform for x86 CPUs.  
Disponível em: <http://www.amd.com/us/press-releases/Pages/amd-delivers-and-submits-2009aug04.aspx>
- AMD, 2009b. ATI Stream technology.  
Disponível em: <http://www.amd.com/stream>
- Bammer, R., Acar, B., Moseley, M. E., 2003. In vivo MR tractography using diffusion imaging. *European Journal of Radiology* 45, 223–234.
- Basser, P. J., Mattiello, J., Le Bihan, D., 1994a. Estimation of the effective self-diffusion tensor from the NMR spin echo. *Journal of Magnetic Resonance B* 103, 247–254.
- Basser, P. J., Mattiello, J., LeBihan, D., 1994b. MR diffusion tensor spectroscopy and imaging. *Biophysics Journal* 66, 259–267.
- Basser, P. J., Pajevic, S., Pierpaoli, C., Aldroubi, A., 2002. Fiber tract following in the human brain using DT-MRI data. *IEICE Trans. Inf. & Syst* E85-D, 15–21.
- Basser, P. J., Pajevic, S., Pierpaoli, C., Duda, J., Aldroubi, A., 2000. In vivo fiber tractography using DT-MRI data. *Magnetic Resonance in Medicine* 44, 625–632.
- Basser, P. J., Pierpaoli, C., 1996. Microstructural and physiological features of tissues elucidated by quantitative-diffusion-tensor MRI. *Journal of Magnetic Resonance B* 111, 209–219.
- Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P., 2004. Brook for gpus: stream computing on graphics hardware. In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*. ACM, New York, NY, USA, pp. 777–786.
- Carvalho, D. D. B., Nobrega, T. H. C., von Wangenheim, A., 2008. Estação radiológica tridimensional multi-propósito. In: *XI Congresso Brasileiro de Informática em Saúde*.
- Clark, C. A., Barrick, T. R., Murphy, M. M., Bell, B. A., 2003. White matter fiber tracking in patients with space-occupying lesions of the brain: a new technique for neurosurgical planning? *NeuroImage* 20, 1601–1608.

- Conturo, T. E., Lori, N. F., Cull, T. S., Akbudak, E., Synder, A. Z., Shimony, J. S., McKinstry, R. C., Burton, H., Raichle, M. E., 1999. Tracking neuronal fiber pathways in the living human brain. In: Proc. Natl. Acad. Sci. USA. pp. 10422–10427.
- Coulon, O., Alexander, D. C., Arridge, S., 2004. Diffusion tensor magnetic resonance image regularization. *Medical Image Analysis* 8, 47–67.
- Fernando, R., Kilgard, M. J., 2003. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley.
- Foster, I., 1995. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley.
- Grupo Cyclops, 2009. Página do Grupo Cyclops.  
Disponível em: <http://www.cyclops.ufsc.br/>
- Harris, M. J., Coombe, G., Scheuermann, T., Lastra, A., 2002. Physically-based visual simulation on graphics hardware. In: SIGGRAPH/Eurographics Workshop on Graphics Hardware.
- IEEE, 1985. IEEE Standard 754 for Binary Floating-Point Arithmetic. IEEE.
- Intel, 2009. Larrabee microarchitecture.  
Disponível em: <http://www.intel.com/technology/visual/microarch.htm>
- Jeong, W.-K., Fletcher, P. T., Tao, R., Whitaker, R. T., 2007. Interactive visualization of volumetric white matter connectivity in DT-MRI using a parallel-hardware hamilton-jacobi solver. *IEEE Transactions on Visualization and Computer Graphics* 13, 1480–1487.
- Kehtarnavaz, N., Gamadia, M., 2006. *Real-Time Image and Video Processing: From Research to Reality*. Morgan & Claypool.
- Khronos Group, 2009. OpenCL.  
Disponível em: <http://www.khronos.org/opencv/>
- Köhn, A., Klein, J., Weiler, F., Peitgen, H.-O., 2009. A GPU-based fiber tracking framework using geometry shaders. In: *Proceedings of SPIE Medical Imaging 2009*. Vol. 7261. pp. 72611J1–72611J10.
- Lehmann, T. M., Gönner, C., Spitzer, K., 1999. Survey: interpolation methods in medical image processing. *IEEE Transactions on Medical Imaging* 18, 1049–1075.
- Lorensen, W. E., Cline, H. E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. Vol. 21. ACM Press, New York, NY, USA, pp. 163–169.
- Luebke, D., Humphreys, G., 2007. How GPUs work. *Computer* 40, 96–100.
- Macedonia, M., 2003. The GPU enters computing's mainstream. *Computer* 36, 106–108.

- McGraw, T., Nadar, M., 2007. Stochastic DT-MRI connectivity mapping on the GPU. *IEEE Transactions on Visualization and Computer Graphics* 13, 1504–1511.
- Miljanic, Z., Kaeli, D. R., 1992. A study of 80x86/80x87 floating-point execution. *SIGSMALL/PC Notes* 18, 13–17.
- Mishra, A., Lu, Y., Choe, A. S., Aldroubi, A., Gore, J. C., Anderson, A. W., Ding, Z., 2007. An image-processing toolset for diffusion tensor tractography. *Magnetic Resonance Imaging* 25, 365–376.
- Mishra, A., Lu, Y., Meng, J., Anderson, A. W., Ding, Z., 2006. Unified framework for anisotropic interpolation and smoothing of diffusion tensor images. *NeuroImage* 31, 1525–1535.
- Mittmann, A., Comunello, E., von Wangenheim, A., 2008. Diffusion tensor fiber tracking on graphics processing units. *Computerized Medical Imaging and Graphics* 32, 521–530.
- Mittmann, A., Dantas, M., von Wangenheim, A., 2009. Design and implementation of brain fiber tracking for GPUs and PC clusters. In: *Proceedings of the 21st Symposium on Computer Architecture and High Performance Computing*. pp. 101–108.
- Mori, S., 2007. *Introduction to Diffusion Tensor Imaging*. Elsevier.
- Mori, S., Barker, P. B., 1999. Diffusion magnetic resonance imaging: Its principles and applications. *The Anatomical Record* 257, 102–109.
- Mori, S., Zhang, J., 2006. Principles of diffusion tensor imaging and its applications to basic neuroscience research. *Neuron* 51, 527–539.
- Munshi, A., 2009. OpenCL: Parallel computing on the GPU and CPU. *SIGGRAPH 2008 Course*.
- Neil, J. J., 1997. Measurement of water motion (apparent diffusion) in biological systems. *Concepts in Magnetic Resonance* 9, 385–401.
- Newegg, 2009a. MSI N9600GT-T2D512-OC GeForce 9600 GT 512MB 256-bit GDDR3 PCI Express 2.0 x16 HDCP Ready SLI Supported Video Card - Retail.  
Disponível em: <http://www.newegg.com/Product/Product.aspx?Item=N82E16814127382>
- Newegg, 2009b. PALiT NE/860TS+T321 GeForce 8600GT SONIC 256MB 128-bit GDDR3 PCI Express x16 HDCP Ready SLI Supported Video Card - Retail.  
Disponível em: <http://www.newegg.com/Product/Product.aspx?Item=N82E16814260057>
- Newegg, 2009c. XFX PVT86JYAHG GeForce 8500 GT 512MB 128-bit GDDR2 PCI Express x16 HDCP Ready SLI Supported Video Card - Retail.  
Disponível em: <http://www.newegg.com/Product/Product.aspx?Item=N82E16814150245>
- Nimsky, C., Ganslandt, O., Hastreiter, P., Wang, R., Benner, T., Sorenson, A. G., Fahlbusch, R., 2005. Preoperative and intraoperative diffusion tensor imaging-based fiber tracking in glioma surgery. *Neurosurgery* 56, 130–138.



- NVIDIA, 2006. New NVIDIA products transform the PC into the definitive gaming platform.  
Disponível em: [http://www.nvidia.com/object/I0\\_37234.html](http://www.nvidia.com/object/I0_37234.html)
- NVIDIA, 2007a. NVIDIA brings cutting-edge DirectX 10 graphics and HD video to all PC users.  
Disponível em: [http://www.nvidia.com/object/I0\\_41448.html](http://www.nvidia.com/object/I0_41448.html)
- NVIDIA, 2007b. NVIDIA® CUDA™ unleashes power of GPU computing.  
Disponível em: [http://www.nvidia.com/object/I0\\_39918.html](http://www.nvidia.com/object/I0_39918.html)
- NVIDIA, Julho 2008a. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. 2nd Edition.
- NVIDIA, Agosto 2008b. NVIDIA CUDA™ Programming Guide. 2nd Edition.
- NVIDIA, 2008c. NVIDIA reveals one of the largest generational performance leaps in company history.  
Disponível em: [http://www.nvidia.com/object/io\\_1203588081917.html](http://www.nvidia.com/object/io_1203588081917.html)
- NVIDIA, 2009a. GeForce 8500.  
Disponível em: [http://www.nvidia.com/object/geforce\\_8500.html](http://www.nvidia.com/object/geforce_8500.html)
- NVIDIA, 2009b. GeForce 8600.  
Disponível em: [http://www.nvidia.com/object/geforce\\_8600.html](http://www.nvidia.com/object/geforce_8600.html)
- NVIDIA, 2009c. GeForce 8800.  
Disponível em: [http://www.nvidia.com/page/geforce\\_8800.html](http://www.nvidia.com/page/geforce_8800.html)
- NVIDIA, 2009d. GeForce GTX 275.
- NVIDIA, 2009e. NVIDIA authorized board partners.  
Disponível em: [http://www.nvidia.com/object/pf\\_boardpartners.html](http://www.nvidia.com/object/pf_boardpartners.html)
- NVIDIA, 2009f. NVIDIA GeForce 9600 GT.  
Disponível em: [http://www.nvidia.com/object/product\\_geforce\\_9600gt\\_us.html](http://www.nvidia.com/object/product_geforce_9600gt_us.html)
- NVIDIA, 2009g. NVIDIA submits OpenCL 1.0 driver to Khronos for conformance certification for Windows and Linux.  
Disponível em: [http://www.nvidia.com/object/io\\_1242238985095](http://www.nvidia.com/object/io_1242238985095)
- OpenGL, 2009. OpenGL - the industry standard for high performance graphics.  
Disponível em: <http://www.opengl.org/>
- Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., Phillips, J. C., 2008. GPU computing. Proc. IEEE 96, 879–899.
- Pajevic, S., Aldroubi, A., Basser, P. J., 2002. A continuous tensor field approximation of discrete DT-MRI data for extracting microstructural and architectural features of tissue. Journal of Magnetic Resonance 154, 85–100.
- Petrovic, V., Fallon, J., Kuester, F., 2007. Visualizing whole-brain DTI tractography with GPU-based tuboids and LoD management. IEEE Transactions on Visualization and Computer Graphics 13, 1488–1495.

- Poupon, C., Clark, C. A., Frouin, V., Régis, J., Bloch, I., Bihan, D. L., Mangin, J., 2000. Regularization of diffusion-based direction maps for the tracking of brain white matter fascicles. *NeuroImage* 12, 184–195.  
Disponível em: <http://dx.doi.org/10.1006/nimg.2000.0607>
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., 1992. *Numerical recipes in C: The art of scientific computing*. Cambridge University Press.
- Santo, B., Adee, S., 2009. Multicore made simple. *IEEE Spectrum* 46, 32–36.
- Shimony, J. S., McKinstry, R. C., Akbudak, E., Aronovitz, J. A., Snyder, A. Z., Lori, N. F., Cull, T. S., Conturo, T. E., 1999. Quantitative diffusion-tensor anisotropy brain MR imaging: Normative human data and anatomic analysis. *Radiology* 212, 770–84.
- Silva, A. F. B., Carvalho, D. D. B., Nobrega, T. H. C., Inácio, R. T., von Wangenheim, A., 2009. A multi-layered development framework for medical imaging applications. In: *22nd IEEE International Symposium on Computer-Based Medical Systems*.
- Singh, M., Kwatra, A., Wong, C.-W., Prasanna, V., 2006. Acceleration of fiber tracking in DTI tractography by reconfigurable computer hardware. In: *Proc. 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society EMBS '06*. pp. 4819–4822.
- Stejskal, E., Tanner, J., 1964. Spin diffusion measurements: Spin echoes in the presence of a time-dependent field gradient. *Journal of Chemical Physics* 42, 288–292.
- Thompson, C. J., Hahn, S., Oskin, M., 2002. Using modern graphics architectures for general-purpose computing: a framework and analysis. In: *Proc. 35th Annual IEEE/ACM International Symposium on (MICRO-35) Microarchitecture*. pp. 306–317.
- Westin, C.-F., Maier, S., Mamata, H., Nabavi, A., Jolesz, F., Kikinis, R., 2002. Processing and visualization for diffusion tensor MRI. *Medical Image Analysis* 6, 98–108.
- Westin, C.-F., Maier, S. E., Khidhir, B., Everett, P., Jolesz, F. A., Kikinis, R., 1999. Image processing for diffusion tensor magnetic resonance imaging. In: *Medical Image Computing and Computer-Assisted Intervention. Lecture Notes in Computer Science*. pp. 441–452.
- wxWidgets, 2009. wxWidgets.  
Disponível em: <http://www.wxwidgets.org/>