View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio da Producao Cientifica e Intelectual da Unicamp

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Segmentação e Visualização Tridimensional Interativa de Imagens de Ressonância Magnética

Dissertação de Mestrado

Autor: **Romaric Matthias Michel Audigier**

Orientador: **Prof. Dr. Roberto de Alencar Lotufo (FEEC/Unicamp)**

Co-orientador: **Prof. Dr. Alexandre Xavier Falcão (IC/Unicamp)**

Banca Examinadora: **Prof. Dr. Luiz Velho (IMPA/Rio de Janeiro)**
Prof. Dr. Clésio Luís Tozzi (FEEC/Unicamp)

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Fevereiro de 2004

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

Au24s

Audigier, Romaric Matthias Michel
Segmentação e visualização tridimensional interativa de imagens de ressonância magnética / Romaric Matthias Michel Audigier. --Campinas, SP: [s.n.], 2004.

Orientador: Roberto de Alencar Lotufo e Alexandre Xavier Falcão.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Imagem tridimensional. 2. Visualização. 3. Morfologia matemática. 4. Processamento de imagens. I. Lotufo, Roberto de Alencar. II. Falcão, Alexandre Xavier. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Resumo

Este trabalho propõe métodos rápidos e iterativos que integram segmentação e visualização tridimensional, enquanto a abordagem clássica separa totalmente esses dois processos. A segmentação interativa, com auxílio de usuário, baseia-se no *watershed* iterativo, implementado de forma eficiente via Transformada Imagem-Floresta (IFT). Estudaram-se várias estratégias de visualização para interligá-las à segmentação. O primeiro algoritmo proposto consiste em extrair a borda das estruturas segmentadas durante sua segmentação e atualizá-la em cada iteração. Visualizam-se as estruturas por projeção da borda. O segundo algoritmo atualiza diretamente a imagem em função das mudanças ocorridas na cena segmentada a cada passo. Os dois métodos, implementados em C e aplicados a diversas Imagens de Ressonância Magnética entre outras, deram resultados satisfatórios comparados à abordagem clássica. O algoritmo de atualização da imagem mostra-se um pouco mais rápido que o método clássico, enquanto que o algoritmo de atualização da borda, apesar de ser um pouco mais lento na segmentação, possibilita um *rendering* muito rápido adequado à manipulação do objeto segmentado. Os métodos propostos proporcionam, portanto, uma boa realimentação ao usuário.

Palavras-Chave: segmentação e visualização, *watershed* via IFT, *rendering* tridimensional interativo.

Abstract

This work presents fast and iterative methods that integrate segmentation and three-dimensional visualization, while the classic approach separates these two processes. The user-aided segmentation is based on iterative watershed, implemented with the efficient Image Foresting Transform (IFT). Several techniques of visualization were analyzed. The first proposed algorithm consists of extracting segmented structure borders during the segmentation and updating them at each iteration. Structure visualization is achieved by border projection. The second algorithm updates the image directly from the changes of the segmented scene occured in each step. The two methods were implemented in C and tested with various Magnetic Resonance Images. The results are satisfatory compared to the classical approach. The image update algorithm is a little faster than the classical one while the border update algorithm, although a little slower in segmenting, allows a very fast rendering, very suitable for object manipulation. Thus, the proposed methods provide a good feedback to user.

Keywords: segmentation and visualization, IFT-watershed, three-dimensional interactive rendering.

Agradecimentos

Gostaria de agradecer a todos que, de longe ou de perto, me ajudaram neste trabalho:

Ao meu orientador, Roberto Lotufo, por ter confiado em mim, por seu incentivo nos momentos em que eu não via o fim do túnel, pelas reuniões aliviadoras e enriquecedoras, pela lucidez, paciência e compreensão com as quais sempre soube enxergar este trabalho e pelo exemplo de humildade que deu.

Ao meu co-orientador, Alexandre Falcão, pelo incentivo, pela colaboração e por ter me ajudado a botar as mãos na massa... dos programas.

Ao Prof. Dr. Luiz Velho por ter aceitado, com presteza, participar da banca de defesa. Ao Prof. Dr. Clésio Luis Tozzi por ter aceitado ser membro das bancas de defesa e qualificação e pelas observações sobre a primeira versão do trabalho. À Profa. Dra. Wu Shin Ting por ter participado também da banca do exame de qualificação.

À CAPES pelo apoio financeiro.

Ao laboratório de Neuroimagem pelas imagens de ressonância magnética fornecidas para os testes.

Aos colegas do Instituto de Computação, Felipe Bergo, que me cedeu vários conjuntos de sementes para a segmentação assistida das cenas de ressonância magnética, além das dicas em programação; e ao Leonardo Rocha pela simpatia e pelos valiosos conselhos em computação.

Ao Helder, fiel companheiro das noites passadas no laboratório, pelo incentivo, pela preciosa ajuda e amizade. Ao amigo e camarada de turma Leandro que também esteve nessa mesma aventura chamada mestrado e que me ajudou em domar o ferroz LaTeX.

Aos amigos e às amigas do LCA, Alexandre, Célio, Cláudio, Gabriela, Gonzaga, Ivana, Leandrinho, Marco Antônio, Marina, Rangel, Rodrigo, Rubens, Tiago e os outros, pela simpatia e acolhimento.

Aos amigos encontrados no Brasil e aos meus saudosos amigos da França, pela felicidade que eles trazem.

Aos meus pais Jean-Claude e Anne-Marie por me deixarem a liberdade de escolher os meus próprios caminhos, se importando sobretudo com a minha felicidade, e também pelo seu apoio e incentivo em todos os momentos de alegria ou tristeza. Às minhas queridas irmãs Emilie e Laure pela alegria que nos liga. À minha doce e corajosa avó pela paciência com a qual espera minha volta.

À minha namorada, Patrícia, por seu carinho, sua confiança e seu apoio moral e também pela atenção com que corrigiu minha tese

Finalmente agradeço a Deus por ter me dado força e perseverência nesses dois anos.

A meus pais, Jean-Claude e Anne-Marie.

“Les choses visibles me sont comme invisibles, et les invisibles comme visibles ; et les unes et les autres me servent en cette manière comme de défense contre la tentation de tous les biens et de tous les maux de ce monde.”

Abbé de Saint Cyran (1581-1643)

Sumário

RESUMO i

AGRADECIMENTOS iii

SUMÁRIO vii

LISTA DE FIGURAS xi

LISTA DE TABELAS xiii

1 Introdução 1

1.1 Motivação 1

1.2 Objetivos 3

1.3 Contribuições 4

1.4 Organização do documento 4

2 Segmentação por *watershed* 5

2.1 Segmentação 5

2.2 A transformada de *watershed* 7

2.2.1 Um processo de inundação 7

2.2.2 *Watershed* por marcadores 8

2.3 A Transformada Imagem-Floresta (IFT) 9

2.3.1 Generalidades sobre a IFT 10

2.3.2 Algoritmo 11

2.3.3 Implementação da fila hierárquica 13

2.3.4 *Watershed* via IFT 14

2.4 *Watershed* iterativo via DIFT . . . 15
2.4.1 Generalidades sobre a DIFT . . . 16
2.4.2 Algoritmo . . . 18
2.4.3 Observações . . . 21

3 Visualização tridimensional 23

3.1 Visão geral das abordagens em visualização tridimensional . . . 23
3.1.1 Pré-processamento: Da fatia à cena . . . 23
3.1.2 Da cena à imagem . . . 24
3.1.3 Classificação das abordagens em visualização de dados tridimensionais . . . 25
3.2 *Rendering* de superfície . . . 26
3.2.1 Extração de superfície . . . 27
3.2.1.1 Superfície baseada em contornos 2D . . . 27
3.2.1.2 Superfície baseada em contornos 3D . . . 27
3.2.1.3 Comparação dos métodos de extração de superfície . . . 28
3.2.2 *Rendering* . . . 29
3.3 *Rendering* de volume . . . 29
3.3.1 Métodos com precisão de objeto . . . 30
3.3.1.1 Projeção de voxels . . . 30
3.3.1.2 Otimizações em qualidade e eficiência . . . 31
3.3.2 Métodos com precisão de imagem . . . 32
3.3.2.1 *Raycasting* num volume semitransparente . . . 32
3.3.2.2 *Raycastings* otimizados . . . 33
3.3.3 Métodos no domínio da freqüência . . . 35
3.3.4 Observações . . . 35
3.4 Tonalização . . . 37
3.4.1 *Depth-Shading* . . . 37
3.4.2 *Phong-Shading* . . . 37
3.4.3 Gouraud-*Shading* ou *Smooth Shading* . . . 39

4 Segmentação e visualização interativas **41**

4.1 Estratégias de segmentação e visualização interativas 41

4.2 Borda incremental 44

4.2.1 Extração e atualização da borda 44

4.2.2 Algoritmo 46

4.3 Imagem incremental 49

4.3.1 Análise das mudanças 49

4.3.2 Algoritmo 53

5 Resultados experimentais **55**

5.1 Experimentos 55

5.1.1 Implementação 55

5.1.2 Testes 56

5.1.3 Material 56

5.1.4 Imagens de teste 57

5.1.5 Roteiros de interatividade 58

5.2 Discussão dos resultados 60

5.2.1 Comparação das técnicas de *rendering* 60

5.2.2 Comparação das estratégias em relação à segmentação 63

5.2.3 Comparação das estratégias em relação à visualização 65

5.2.4 Comparação geral das estratégias de segmentação e visualização 66

6 Conclusão **71**

Referências Bibliográficas **73**

A Convenções adotadas no ambiente de visualização **79**

A.1 Definições 79

A.2 Convenções adotadas nos programas 81

A.2.1 Coordenadas homogêneas 81

A.2.2 Matrizes de transformação 81

A.2.3 Composição de transformações 81

A.3 Outras convenções possíveis 83

A.3.1 Coordenadas homogêneas 83
A.3.2 Matrizes de transformação 83
A.3.3 Composição de transformações 84
A.4 Ambiente de visualização 84

B Algoritmo de Bresenham **85**
B.1 Introdução 85
B.2 Algoritmo de Bresenham em 2D 86
B.3 Algoritmo de Bresenham em 3D 88

C Avaliação da normal **91**
C.1 Avaliação de normal no espaço imagem 91
C.2 Avaliação de normal no espaço voxel 92
C.3 Avaliação da normal no espaço objeto a partir do *d-buffer* 94

D Fatoração *shear-warp* **97**
D.1 Generalidades 97
D.2 Matrizes *shear* e *warp* 98
D.2.1 No caso de um eixo principal em z 98
D.2.2 Generalização para qualquer eixo principal 98

E Imagens obtidas **101**

Lista de Figuras

2.1 Transformada de *watershed*: um processo de inundação. 8

2.2 Vizinhanças-4 e -8 10

2.3 Algoritmo da DIFT: Etapas intermediárias dentro de uma iteração. 17

3.1 Representação esquemática das transformações aplicadas a uma cena para se obter uma imagem e suas características. (adaptada de Udupa e Herman) 25

3.2 Classificação das abordagens em visualização de dados tridimensionais. 26

3.3 Modelo de iluminação de Phong 39

4.1 Segmentação não incremental com visualização de cena 41

4.2 Segmentação iterativa com visualização de cena 42

4.3 Segmentação iterativa com visualização de borda 43

4.4 Segmentação iterativa com visualização incremental 43

4.5 Imagem Incremental: O voxel apareceu 50

4.6 Imagem Incremental: O voxel mudou de rótulo 51

4.7 Imagem Incremental: O voxel desapareceu 52

5.1 Roteiro genérico das interações do usuário durante uma segmentação semi-automática por DIFT. 58

5.2 Tempos relativos médios de *rendering*. 61

5.3 Tempos de segmentação de "*brain*" para cada iteração usando: a IFT, a DIFT ou a BDIFT (experimentos 1 a 3 respectivamente) 64

5.4 Duração relativa média de um passo de segmentação (passos posteriores ao primeiro) usando: a IFT, a DIFT ou a BDIFT (experimentos 1 a 3 respectivamente). 64

5.5 Tempos de visualização para cada passo da segmentação de "*brain3*" usando: o *splatting* de borda (**S-Borda**), a atualização da imagem (**A-Img**) ou o *splatting* de cena (**S**) (experimentos 3, 5 e 2 respectivamente) 66

5.6 Duração relativa média da visualização (em cada passo posteriores ao primeiro) usando: a projeção de borda (**S-Borda**), a atualização da imagem (**A-Img**) ou a projeção de cena (**S**) (experimentos 3, 5 e 2 respectivamente). 66

5.7 Tempo total de segmentação e visualização em cada passo (cena "*brain*"). 67

5.8 Duração relativa média de um passo de segmentação e visualização. 68

A.1 Ambiente de visualização: convenções e sistemas adotados. 84

B.1 Algoritmo de Bresenham 2D. 87

B.2 Algoritmo de Bresenham 3D. 90

C.1 Estimação da normal a partir do *d-buffer* (espaço imagem). 92

C.2 Estimação da normal a partir dos níveis de cinza (espaço voxel). 94

C.3 Estimação da normal a partir das coordenadas de voxels (espaço objeto). 96

D.1 Fatoração *shear-warp*. 98

E.1 Cena de teste "*knee*" segmentada 101

E.2 Cenas de teste segmentadas: "*geometric*", "*cena1*", "*cena2*", "*cena3*", "*cena4*" e "*cena5*" . . 102

E.3 Diferentes técnicas de *rendering* aplicadas à cena "*geometric*" segmentada 103

E.4 Segmentação da cena "*brain*" passo a passo (visualização segundo o plano axial) 104

E.5 Segmentação da cena "*brain*" passo a passo (visualização segundo o plano coronal) 105

E.6 Segmentação da cena "*brain*" passo a passo (visualização segundo o plano sagital) 106

E.7 Segmentação da cena "*brain3*" passo a passo (visualização segundo o plano axial) 107

E.8 Segmentação da cena "*brain3*" passo a passo (visualização segundo o plano sagital) 108

Lista de Tabelas

2.1 Operações sobre os conjuntos. 11

5.1 Cenas de teste. O nome das cenas interpoladas comporta o sufixo $_interpXYZ$ onde X, Y e Z são os coeficientes de interpolação segundo os eixos x, y e z respectivamente. 57

5.2 Roteiro para as cenas "*geometric*", suas 7 interpolações e "*knee*". 59

5.3 Roteiro para as cenas de RM. 60

5.4 Tempos de *rendering* (em ms) da cena "*brain*" nas máquinas M1 e M2. 61

5.5 Tempos de segmentação (em s) de *"brain"* para cada iteração nos experimentos 1, 2 e 3 . . 64

5.6 Tempos de visualização (em ms) para cada passo da segmentação de "*brain3*" nos experimentos 3, 5 e 2. 66

5.7 Tempos de $\{$segmentação + visualização$\}$ (em s) para cada passo da segmentação de "*brain*" nos 5 experimentos. 67

Capítulo 1

Introdução

1.1 Motivação

Imagens e medicina

Hoje em dia, as imagens são cada vez mais indispensáveis à medicina. Elas são usadas tanto para a diagnose quanto para o planejamento e acompanhamento de cirurgias ou tratamentos por radiação, além de servirem de ferramentas para a pesquisa e o ensino nas ciências médicas. A familiaridade com os diversos tipos de imagens é uma parte importante do conhecimento do médico (Nisbet et al., 1987). Essas imagens, que permitem a observação não-invasiva de estruturas, podem ser obtidas por radiografia X clássica ou por modalidades mais especializadas como a tomografia computadorizada por raio-X (*Computerized Tomography*, CT), a ultra-sonografia, a tomografia por emissão de pósitron (*Positron Emission Tomography*, PET), a tomografia por emissão de fóton (*Single-Photon Emission Computed Tomography*, SPECT) ou a Ressonância Magnética (RM). Cada uma dessas técnicas de aquisição de imagem é baseada em um fenômeno físico (absorção dos raios-X, eco das ondas ultra-sonoras, radioatividade de nuclídeos, ressonância do spin magnético) que é explorado a fim de obter imagens que refletem características anatômicas ou funcionais das estruturas de interesse (Shung et al., 1992). Hoje em dia, além das imagens clássicas obtidas em filmes (radiografia X ou ultra-sonografia) e analisadas diretamente pelo médico, existem muitos exames feitos através de sistemas digitais que criam imagens digitais, exploráveis por computadores. Este trabalho refere-se unicamente a esta categoria de imagens.

Em certas aplicações médicas, a visualização bidimensional (2D) de "fatias" de estruturas de interesse é insuficiente, pois perde a informação da terceira dimensão. Frente às numerosas fatias 2D geradas pelos equipamentos de aquisição de imagens, o médico sente falta de uma visão global. Ele precisa então ter uma visão tridimensional, mais representativa da realidade. Depois da aparição, nos anos 70, dos tomógrafos que dão imagens 2D de tecidos ou órgãos sob planos predeterminados, geralmente ortogonais a um eixo principal do paciente, desenvolveu-se a área de visualização tridimensional (Udupa e Herman, 1991). A partir do empilhamento das diversas fatias (imagens bidimensionais) obtidas por esses equipamentos e respeitando-se

o espaçamento original entre elas, pode-se constituir um volume de dados, chamado cena, que representa uma região tridimensional (3D) do paciente. Desta maneira, uma boa visualização tridimensional permite a observação em qualquer direção do objeto de interesse. Porém, apenas a observação da cena e de seus objetos de interesse pode mostrar-se insuficiente para o médico que deseja analisar quantitativamente estruturas particulares. Em certos casos, a diagnose apóia-se não só sobre a forma, disposição das estruturas, mas também sobre medidas quantitativas dessas, como volume, superfície, perímetro, centro de gravidade, momentos de inércia e posição relativa.

Surge então o problema seguinte: Como fazer medidas sobre estruturas cujos limites são desconhecidos? A resolução desse problema é um dos propósitos da segmentação. A segmentação é um processo que particiona os elementos de uma imagem, formando vários componentes, segmentos ou objetos (Gonzalez e Woods, 1993). A cada componente pode-se atribuir um rótulo característico para designá-lo. Uma vez essa partição feita, efetuam-se as medidas relativas a cada componente.

Segmentação e visualização tridimensional

O problema da segmentação, no entanto, é um dos mais difíceis na área de processamento de imagens. Existem várias abordagens para resolvê-lo. Primeiro, pode-se realizar uma segmentação manualmente. No caso de cenas (3D), o usuário delineia a borda dos objetos ou "pinta" regiões de interesse em cada uma das fatias (2D). Essa abordagem apresenta claramente várias desvantagens. A segmentação torna-se um processo tedioso que, em geral, requer um alto consumo de tempo, mesmo que seja auxiliada por sistemas gráficos interativos. Além disso, os resultados obtidos manualmente são pouco reprodutíveis e, portanto, pouco confiáveis; comumente ocorrendo variações intra e interoperadores. Apesar dessas dificuldades (falta de repetitibilidade e duração), o processo manual é ainda hoje muito utilizado, pois a segmentação automática nem sempre consegue dar resultados corretos. Com efeito, um processo automático funciona bem quando ele é especializado em segmentar um certo tipo de estruturas numa certa modalidade de imagens. As cenas obtidas por CT com raios-X, por exemplo, possuem geralmente um bom contraste o que facilita a segmentação de partes ósseas. Já, para as Imagens de Ressonância Magnética (IRM), os métodos que segmentam órgãos e tecidos tornam-se bem mais complexos. A segmentação automática tem a grande vantagem de não envolver tempo e interações do usuário. Mas quando se ganha em generalidade ou multiplicidade quanto às estruturas segmentadas, perde-se na precisão e corretude das segmentações.

A terceira abordagem é um meio-termo entre as duas primeiras. A segmentação semi-automática ou supervisionada consiste em minimizar as interações do usuário com o sistema, dando-lhe as condições de controlar e influir sobre o resultado da segmentação. A partir de algumas ações do usuário, o sistema processa a imagem para obter uma imagem segmentada modificável. Permitindo-se assim a supervisão do usuário, tem-se um compromisso entre tempo gasto por este e corretude do resultado. Esse fato tem motivado a pesquisa e o desenvolvimento de métodos interativos de segmentação que minimizem o envolvimento e o tempo total do usuário (Falcão et al., 1998; Falcão et al., 2000; Falcão e Udupa, 2000). Esta também foi a abordagem adotada neste trabalho. Os trabalhos de Rondina (Rondina et al., 2000; Rond-

ina, 2001) propuseram interface e método interativos em sistemas usados hoje pela equipe de pesquisadores do laboratório de Neuroimagem da FCM-Unicamp, para a segmentação em IRM de estruturas do cérebro relacionadas às epilepsias (Carnevalle et al., 2003). Uma dificuldade observada em segmentação interativa de imagens médicas é que os usuários conduzem o processo em imagens bidimensionais (2D) sem nenhuma realimentação visual da anatomia tridimensional (3D) dos objetos que estão sendo segmentados. Por isso, muitos erros cometidos pelos usuários durante a segmentação só são percebidos após reconstrução e visualização 3D. Isto porque um processamento considerável é normalmente utilizado entre a segmentação e a visualização. Geralmente, os trabalhos que apresentam um método de segmentação e visualização 3D simultâneas e interativas propõem uma visualização que não é obtida em tempo real, muito embora esteja sendo utilizada para auxiliar o usuário na segmentação interativa.

Tradicionalmente, o processamento digital de imagens pode ser composto das seguintes etapas: *aquisição* da imagem por um sistema digital, *pré-processamento* para corrigir ruídos ou realçar características, *segmentação*, *representação e descrição* para extrair as características de interesse, e enfim, *reconhecimento e interpretação* (Gonzalez e Woods, 1993). Já, a visualização dos resultados (imagem, imagem segmentada, medidas) é geralmente um problema que pertence à área da computação gráfica (Gomes e Velho, 1994). No nosso conhecimento, existem sistemas que integram os processos de segmentação e de visualização, mas nenhum os mistura. São geralmente dois blocos de tratamento separados e totalmente independentes. A idéia do presente trabalho é reduzir boa parte do processamento entre a segmentação e a visualização, combinando esses dois processos e, portanto, possibilitando uma realimentação visual mais rápida e adequada para o usuário durante a segmentação.

1.2 Objetivos

O objetivo principal deste trabalho foi propor métodos rápidos e iterativos que integram segmentação e visualização tridimensional, processos habitualmente independentes. Para isto, foram alcançados os objetivos secundários seguintes:

- Estudar os métodos de visualização que existem e analisar suas características (qualidade de imagem, eficiência);
- Investigar possíveis métodos que interliguem os processos de segmentação e de visualização para se aproveitar dos mecanismos de atualização iterativa da segmentação e visualização;
- Comparar as diferentes estratégias para tentar ter uma realimentação mais rápida possível após segmentação ou após rotações da cena, no caso do usuário querer examinar a cena sob vários pontos de vista antes de continuar ou parar o processo de segmentação interativa.

1.3 Contribuições

As principais contribuições deste trabalho foram:

- A proposta de dois novos algoritmos: o primeiro de atualização de borda dos objetos, o segundo de atualização de imagem.
- As implementações em C dos métodos de segmentação e visualização comparados neste trabalho, compatíveis com os sistemas operacionais Windows e Unix.
- Teste e comparação dos algoritmos aplicados a diversas imagens e, em particular, imagens médicas de ressonância magnética, provindo do laboratório de Neuroimagem, FCM-Unicamp.
- As implementações dos métodos clássicos de visualização tridimensional no ambiente Adesso, sob forma de uma toolbox “v3dtools”. Este ambiente foi desenvolvido por Machado (Machado, 2002) e permite a criação de caixas de ferramentas, compostas de funções de base compiladas. Essas funções podem ser chamadas por scripts em linguagem Python ou Matlab (disponíveis nas plataformas Unix, Linux e Windows), e agregadas para formar aplicações particulares.

1.4 Organização do documento

Primeiro, apresentamos e explicitamos no capítulo 2 o método de segmentação iterativa usado, baseado em *watershed*. Em seguida, o capítulo 3 faz uma revisão dos principais conceitos em visualização tridimensional e descreve brevemente diferentes tipos de visualizações existentes. Dois algoritmos que ligam os processos de segmentação e visualização são propostos no capítulo 4. Enfim, os experimentos feitos e uma análise dos resultados obtidos encontram-se no capítulo 5, enquanto o capítulo 6 traz conclusões sobre o trabalho e sugere trabalhos futuros.

Capítulo 2

Segmentação por *watershed*

A segmentação interativa baseada na transformada de *watershed* (Beucher e Meyer, 1993; Lotufo e Falcão, 2000) tem se mostrado uma técnica multidimensional simples, eficiente e eficaz em muitas situações. O método de *watershed*, que simula uma inundação em um relevo, baseia-se na escolha de marcadores rotulados na imagem, a partir dos quais são definidas as regiões que pertencem aos objetos de interesse e ao fundo da imagem. Pela escolha de rótulos distintos, vários objetos são obtidos ao mesmo tempo. Nas seções seguintes (seções 2.1 e 2.2), apresentamos alguns conceitos sobre a segmentação e a transformada de *watershed*. Seguem dois algoritmos que podem processar, entre outras coisas, essa transformada de maneira iterativa (seção 2.4) ou não (seção 2.3).

2.1 Segmentação

Em processamento de imagens, a segmentação ocupa um lugar capital. Dessa etapa geralmente dependem os processos de representação e descrição de imagens, assim como o reconhecimento e a extração de características, logo, a interpretação da imagem. Para uma boa compreensão da imagem, é necessário segmentá-la. A segmentação é definida em termos gerais como a operação que "divide uma imagem de entrada em partes ou objetos constituintes" (Gonzalez e Woods, 1993). Mais precisamente, neste trabalho, segmentar uma imagem I é associar a cada elemento p desta, um rótulo i especificando a pertença do elemento a um objeto O_i. A segmentação pode ser guiada por critérios variados: apenas o julgamento do usuário na segmentação manual; o valor $I(p)$ (nível de cinza ou valores de cores) do pixel, visto individualmente em relação a um ou vários limiares (limiarização - ou *thresholding*); os valores dos elementos q contidos numa certa vizinhança de p, formando características de textura e contraste. A segmentação também pode ser vista como a obtenção de uma partição dos elementos de uma imagem em vários conjuntos (objetos), segundo os valores de seus atributos (níveis de cinza).

A segmentação é geralmente precedida por pré-processamentos para melhorar a imagem de forma a aumentar as chances para o sucesso da segmentação. É neste estágio que se tenta, por exemplo, reduzir os

ruídos da imagem (imperfeições devidas ao método de aquisição da imagem), realçar algumas características importantes na etapa de segmentação, tais como a presença de bordas de interesse. Para tanto, diversos filtros podem ser utilizados: gradientes, filtros lineares de média, gaussiano, filtro de mediana, etc. (Gomes e Velho, 1994).

Neste trabalho, a segmentação tem como objetivo separar determinadas estruturas de interesse presentes em uma imagem médica tridimensional, para possibilitar a análise e a caracterização qualitativa e quantitativa destas. Com efeito, uma vez a imagem segmentada, é possível efetuar cálculos de características geométricas como área e volume de estruturas, ajudando a análise e a diagnose de doenças, como são feitas, por exemplo, nas pesquisas desenvolvidas pela equipe do laboratório de Neuroimagem da FCM-Unicamp (Bonilha et al., 2003; Demarco et al., 2003; Cendes, 1997; Cendes et al., 1993).

O processo de segmentação inclui técnicas manuais, semi-automáticas e automáticas. No caso de cenas (dados tridimensionais), a técnica manual dominante é a edição de fatias de imagem. Nela, um operador habilidoso, utilizando alguma ferramenta digital, traça as regiões de interesse em cada fatia do volume. Desvantagens óbvias desta técnica são o tempo necessário para segmentar, a dificuldade de reprodução de resultados, dificuldade de comparação de resultados obtidos por diferentes operadores ou pelo mesmo em momentos diferentes, e dificuldade de se fazer inferências de estruturas tridimensionais a partir de fatias bidimensionais. Ao oposto, a segmentação automática tridimensional oferece excelente repetitibilidade, mas pouca robustez diante da variedade de imagens a segmentar. Por isso, em geral, a segmentação automática é uma das tarefas mais difíceis no processamento de imagens digitais (Gonzalez e Woods, 1993). Um meio-termo freqüentemente adotado é a segmentação semi-automática, onde o usuário perde um tempo menor, tendo um número de interações reduzido, mas controla a veracidade do resultado. A técnica adotada neste trabalho é semi-automática.

Existem inúmeros métodos de segmentação com estratégias variadas (Haralick e Shapiro, 1985): detecção de descontinuidades, limiarização, crescimento de regiões, divisão e fusão de regiões (Gonzalez e Woods, 1993), contorno ativo, *snakes* (Kass et al., 1988), *fuzzy connectedness* (Saha et al., 2000), etc. Neste trabalho utilizamos o método das “Linhas Divisoras de Águas” (LDA) ou ainda *watershed*. O *watershed* é uma das ferramentas mais poderosas da Morfologia Matemática para a segmentação.

A escolha desse método foi guiada pelo tipo de imagens de interesse. Com efeito, segundo a modalidade utilizada para gerar imagens, alguns métodos são mais indicados do que outros. Por exemplo, uma simples limiarização pode ser suficiente em certas imagens médicas obtidas por raio-X, pois elas oferecem um bom contraste entre partes de interesse rígidas (ossos) e fundo (tessidos, pele). Já em imagens de ressonância magnética (IRM), uma simples limiarização para segmentar estruturas geralmente fracassa.

2.2 A transformada de *watershed*

Como já dissemos, o método de *watershed* pertence ao campo da Morfologia Matemática. Essa vasta área propõe diversos processamentos de imagem, que são considerados tratamentos não lineares. A particularidade da Morfologia Matemática é interessar-se principalmente às formas dos elementos da imagem e transformar essas formas segundo um elemento estruturante: pequena imagem de forma escolhida que, aplicada na imagem, transforma-a segundo a adequação ou encaixamento das suas formas respectivas.

Usualmente em Morfologia Matemática, consideram-se imagens em níveis de cinza de maneira topográfica. Faz-se uma analogia entre valor de pixel (nível de cinza) e altitude. Valores de intensidade de pixel elevados formam, portanto, relevos (picos), valores menores são associados a vales ou planícies, enquanto regiões com valores constantes são chamadas de platôs. Essa analogia é facilmente visualisável com uma imagem 2D em níveis de cinza, onde se acrescenta uma terceira dimensão: a altitude. A imagem 2D descreve, portanto, uma superfície topográfica 3D.

As operações fundamentais de Morfologia são a dilatação e a erosão, a partir das quais se define a maioria das operações (Dougherty e Lotufo, 2003). Em linhas gerais, os tratamentos morfológicos visam simplificar a imagem reduzindo dados irrelevantes, mas buscando preservar suas características essenciais de forma (Haralick et al., 1987). Neste campo, um dos principais métodos de segmentação está relacionado a uma transformação que define as linhas divisoras de bacias hidrográficas - ou *watershed*. Este é o método de segmentação utilizado neste trabalho.

A transformada de *watershed* pode ser vista de várias maneiras como: um algoritmo de crescimento de regiões, um algoritmo de grafo buscando floresta ótima de árvores de custo mínimo, um processo de inundação. Essa última visão é particularmente prática para se entender o processo do *watershed*.

2.2.1 Um processo de inundação

O funcionamento do método *watershed* baseia-se no princípio físico da inundação de vales. Para melhor entender, considera-se uma imagem bidimensional em níveis de cinza, representada por uma superfície topográfica. Definem-se os mínimos regionais da imagem como as regiões conexas na superfície topológica que apresentam intensidade constante (zonas planas) e menor do que a intensidade dos pontos vizinhos destas regiões (Dougherty e Lotufo, 2003).

Imaginemos agora que os mínimos regionais desta imagem em níveis de cinza sejam perfurados e que a superfície seja imersa na água. A água vai penetrar regularmente pelos furos e encher as bacias hidrográficas até que fluxos provenientes de mínimos regionais diferentes possam se unir. Constroem-se então diques, para evitar esta mistura de fluxos. À medida que a água continua penetrando, diques crescem. Uma vez a superfície totalmente inundada, apenas os diques emergem (vide figura 2.1). Estes são então considerados como linhas divisoras que contornam as bacias hidrográficas, cada uma contendo apenas um

dos mínimos regionais iniciais (Beucher e Meyer, 1993).

O método de *watershed* clássico é aplicado a todos os mínimos regionais de uma imagem. Esta abordagem, em geral, leva a um efeito de supersegmentação, pois muitos mínimos correspondem a ruídos, estruturas menores ou descontinuidades na imagem, que não devem dar origem a uma bacia.

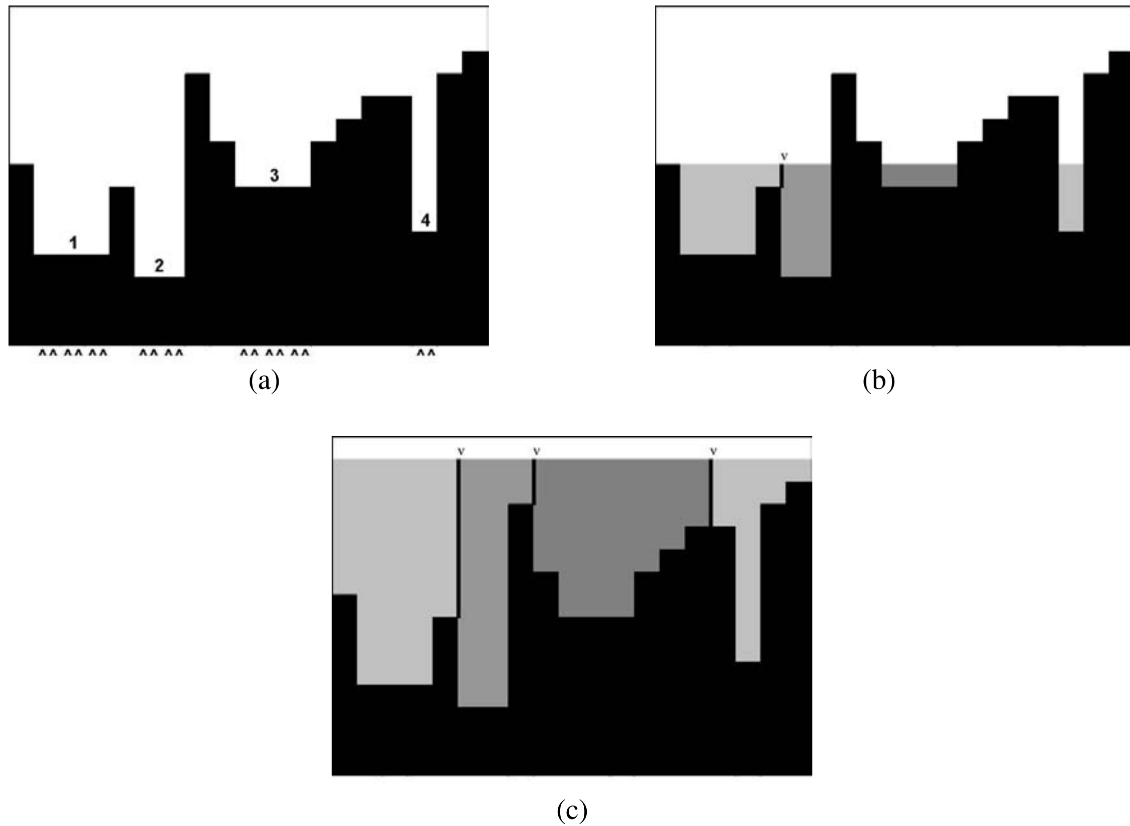


Figura 2.1: Transformada de *watershed*: um processo de inundação. **(a)** O perfil da superfície topográfica contém 4 mínimos regionais, onde são efetuados furos (localizados por setinhas). **(b)** Águas penetram nos diferentes vales e constrói-se o primeiro dique (ver setinha) para separar as águas. **(c)** As linhas de *watershed* situam-se em cima dos diques (setinhas) e separam as bacias hidrográficas.

2.2.2 *Watershed* por marcadores

Uma das abordagens para evitar o problema da supersegmentação é a realização do método *watershed* a partir de marcadores, elementos que são utilizados para fornecer uma informação adicional a respeito das estruturas que serão segmentadas e a partir dos quais as linhas divisoras serão inferidas. A única diferença em relação ao processo de inundação descrito acima é que somente as regiões dos marcadores são perfuradas. As demais sofrem, portanto, invasão das águas provindo dos marcadores, conforme a superfície imerge. Ao final do processo emergem as Linhas Divisoras de Águas para separar apenas fluxos provenientes de bacias selecionadas por marcadores.

Para reforçar a idéia do *watershed* por marcadores, onde, geralmente, são inseridos marcadores rotulados para especificar objetos diferentes, pode-se associar um "corante" por marcador rotulado. Assim, nesta abordagem, os marcadores rotulados localizados na imagem são equivalentes a perfurações na superfície munidas de corante. A superfície inteira é mergulhada na água que pode entrar, com fluxo contínuo, através dos buracos, sendo colorida de acordo com o rótulo do marcador. As Linhas Divisoras de Águas separam então lagos coloridos correspondentes a objetos diferentes.

Qualquer região pode ser selecionada como um marcador. Não precisa pertencer a um mínimo regional. Também não é necessário que o marcador seja uma região conexa. Pode ser formado por várias partes, desde que estas possuam o mesmo rótulo (Meyer e Beucher, 1990). A obtenção dos marcadores pode ser feita automaticamente ou por intervenção humana. No primeiro caso, os marcadores são detectados diretamente da imagem original, normalmente realizando-se uma filtragem intensa desta por meio de recursos morfológicos seguida de uma limiarização resultando em uma imagem binária de marcadores (Meyer e Beucher, 1990). No segundo caso em que o usuário fornece os marcadores, a segmentação torna-se semi-automática, e também denominada de segmentação assistida, supervisionada ou interativa. Esta é a abordagem utilizada neste trabalho.

Para extrair n regiões da imagem, são necessários no mínimo n marcadores. O fundo, mesmo que não diretamente objeto de interesse, é considerado como uma região que tem que ser segmentada atribuindo-lhe no mínimo um marcador. Marcadores adicionais podem ser fornecidos para corrigir falhas ou aumentar a precisão da segmentação.

Apesar de podermos calcular o *watershed* em uma imagem qualquer, as partições resultantes são mais significativas quando calculadas em uma imagem de gradientes, uma vez que os diques tenderão a ser erguidos em pontos onde o gradiente é alto, isto é, prováveis bordas de região. Para tanto, é comum a utilização de gradientes morfológicos para imagens em níveis de cinza ou coloridas (Soille, 1999).

Para mais detalhes sobre a Morfologia Matemática e o método *watershed* recomendam-se as referências (Dougherty, 1992), (Goutsias e Batman, 2000) e (Dougherty e Lotufo, 2003). Na seção seguinte, abordamos uma maneira de se implementar eficientemente o algoritmo de *watershed*: a Transformada Imagem-Floresta.

2.3 A Transformada Imagem-Floresta (IFT)

A Transformada Imagem-Floresta (ou *Image Foresting Transform* - IFT) é um algoritmo de grafo que trata o problema geral de floresta de caminhos mínimos. Desenvolvida por Lotufo e Falcão (Falcão et al., 2002), ela serve de ambiente genérico para a implementação eficiente de vários operadores de processamento de imagens (da Cunha, 2001). Com efeito, diversos problemas em processamento de imagens podem ser abordados como um problema de partição ótima da imagem a partir de um conjunto de pixels sementes. Assim, podem-se implementar transformadas de distância, segmentação baseada em conexidade

fuzzy, geração de esqueletos multiescala e operadores conexos. Mostrou-se que o *watershed* é um caso particular da IFT e que, usá-la para implementar a transformada de *watershed* garante eficiência e otimalidade do algoritmo (Lotufo e Falcão, 2000). Nota-se que existem diversas implementações que dão soluções ótimas de *watershed*. Porém, em certos casos (inundação de um platô por exemplo), ocorre um fenômeno de empate entre águas concorrentes e as implementações podem dar soluções diferentes, pois existem várias soluções ótimas, favorecendo a invasão da região de empate por uma água ou por outra. No caso da IFT, veremos que se usam filas FIFO (*First-In-First-Out*) que dão uma solução atrativa nesse tipo de empate por deixar as águas invadirem a região de empate equitativamente (segundo a relação de vizinhança utilizada).

2.3.1 Generalidades sobre a IFT

A IFT trata a imagem como um grafo onde todos os pixels são nós e as arestas são definidas pela relação de adjacência entre os pixels vizinhos. Usa-se geralmente a vizinhança-4 (ou vizinhança de quarteirão - *city-block*) onde cada pixel é considerado adjacente aos pixels pertencentes ao disco euclidiano de raio unitário centrado nele, ou a vizinhança-8 (ou vizinhança de xadrez - *chessboard*) onde o pixel-nó compartilha arestas com os pixels-nós situados numa distância inferior ou igual a $\sqrt{2}$ (vide figura 2.2). Associa-se um peso a cada arco. Segundo o processamento desejado, o peso dos arcos pode ser fixo ou variar de acordo com os elementos que o constituem.

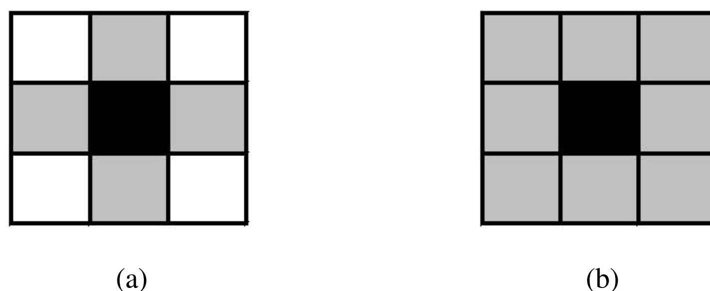


Figura 2.2: Vizinhanças-4 e -8: O pixel central (em preto) tem 4 vizinhos (em cinza) segundo a vizinhança-4 **(a)** ou 8 vizinhos (em cinza) segundo a vizinhança-8 **(b)**

A IFT é um algoritmo de floresta de caminhos mínimos que calcula uma partição ótima dos pixels-nós em árvores de custo mínimo. Portanto, cada pixel-nó recebe um rótulo específico da árvore à qual ele pertence, aponta de alguma maneira para seu predecessor na árvore e tem um custo de caminho.

A IFT requer também um conjunto de pixels sementes, de onde nascem as árvores e que podem ser obtidas de maneira automática ou através de interações com usuário. Cada semente recebe um rótulo que identifica a árvore.

Em função do caminho percorrido de uma semente até um nó, e segundo a função de custo de caminho escolhida, atribui-se um custo para cada nó. Como o algoritmo calcula uma floresta ótima com

árvores de custo mínimo, cada nó tem, no final, o rótulo da semente a partir da qual existe um caminho até ele, que seja de custo mínimo.

2.3.2 Algoritmo

O pseudocódigo A descreve o algoritmo da IFT. A tabela 2.1 apresenta as notações para as principais operações sobre os conjuntos utilizadas nos pseudocódigos desta dissertação.

Nome	**Notação**	**Definição**
Complemento	X^c	$X^c = \{x \mid x \notin X\}$
União	$X \cup Y$	$X \cup Y = \{x \mid x \in X$ ou $x \in Y\}$
Intersecção	$X \cap Y$	$X \cap Y = \{x \mid x \in X$ e $x \in Y\}$
Diferença	X/Y	$X/Y = X \cap Y^c$
Cardinal	$CARD(X)$	Número de elementos contidos em X.
Remoção de elemento	$X \leftarrow X/\{x\}$	Remove o elemento x do conjunto X
Inserção de elemento	$X \leftarrow X \cup \{x\}$	Insere o elemento x no conjunto X

Sejam X e Y dois conjuntos de elementos e x um elemento.

Tabela 2.1: Operações sobre os conjuntos.

A IFT tem como variáveis de entrada a cena [1] I e o conjunto S das sementes rotuladas pela função de rotulação λ. Além destas, deve-se especificar o tipo de vizinhança V utilizada e a função de custo de caminho f. A função f de custo de caminho tem que ser "suave" (da Cunha, 2001). Geralmente, usa-se a função de máximo ou função de soma, satisfazendo essa propriedade.

Retornam-se três mapas (ou cenas) de mesmo tamanho que a cena de entrada. O primeiro representa o mapa dos rótulos, o segundo o mapa de custos e o terceiro o mapa dos predecessores (nos caminhos mínimos).

Utilizam-se duas estruturas de dados auxiliares: a fila de prioridades Q e a função e de estado dos nós. Detalhes sobre a fila de prioridades encontram-se na seção seguinte (2.3.3). A função (ou mapa) de estado e é utilizada para informar o estado corrente de cada nó (pixel ou voxel). Assim, $e(p)$ é nulo no início do programa, quando o nó p não pertence à fila hierárquica Q; $e(p)$ assume o valor 1, quando o nó p está na fila hierárquica; e quando p foi removido de vez da fila Q (está definitivamente processado), ele assume o valor 2.

[1] No decorrer deste trabalho, usaremos o termo *cena* no lugar de imagem digital tridimensional para designar os dados provenientes dos sistemas de imageamento. Reservaremos o termo *imagem* para as imagens digitais bidimensionais para não gerar confusões. Essas são as vistas (2D) que um observador poder ter da cena original (3D) ou são fatias do volume tridimensional de dados. Imagem e cena podem ser vistas como funções discretas bi- e tridimensionais respectivamente, que associam a cada elemento do domínio de definição (*pixel* e *voxel* respectivamente) um valor discreto.

A estrutura do algoritmo é a seguinte (ver pseudocódigo A):
- Inicialização dos mapas de estado, de custo, de rótulos e de predecessores, e da fila de prioridades;
- Laço principal de esvaziamento da fila.

Inicialmente, todos os elementos da cena assumem um custo infinito, não têm nem rótulo, nem predecessor de árvore, e estão fora da fila (ver linha 1). Para cada elemento semente, a inicialização é outra: o custo é nulo, o rótulo dado pela função de rotulação λ, e o nó é inserido na fila Q com seu custo nulo (linha 2).

Em seguida, entra-se no laço principal da IFT. Enquanto a fila de prioridade não estiver vazia, remove-se o elemento de mais alta prioridade, isto é, de custo mínimo (l. 3-4), atualizando-se a função de estado e. O elemento saindo da fila de prioridade tenta, primeiro, propagar seu rótulo em seus vizinhos que ainda não receberam rótulo permanente (l. 5). Para poder propagar o rótulo para o vizinho q, o nó p correspondente tem que oferecer um custo de caminho menor do que o atualmente associado a q. O custo de caminho oferecido é dado pela função de custo de caminho f, aplicada ao caminho oferecido [2] , isto é, ao caminho até p concatenado com o arco de p para q (ver l. 6). Se for possível a propagação do rótulo de p (l. 7), atualizam-se tanto o rótulo do vizinho q com o rótulo de p, como o custo de caminho de q recentemente calculado. O nó p torna-se, conseqüentemente, o predecessor do nó q (l. 10). Como o nó q assume novos valores, ele tem que ser inserido na fila de prioridades com seu novo custo (l. 9). Caso ele já esteja na fila, deve ser previamente removido da fila (l. 8), para não constar duas vezes na fila com dois custos diferentes.

[2]No pseudocódigo, π_p representa o caminho da semente do elemento p até p, enquanto $\pi_p \cdot \langle p, q \rangle$ representa o caminho oferecido a q por p. Este vai da semente de p até q passando por p: É a concatenação do caminho mínimo chegando em p com o arco $\langle p, q \rangle$.

Pseudocódigo A ***IFT***

Variáveis de entrada:

** a cena I (conjunto de voxels),*

** a vizinhança simétrica V: $V(p)$ designa o conjunto dos vizinhos do elemento p,*

** o conjunto S de voxels sementes rotuladas,*

** a função λ que associa um rótulo a cada semente de S,*

** uma função suave de custo de caminho f que associa ao caminho π_p o custo de percurso $f(\pi_p)$.*

Variáveis de saída:

** o mapa de custos C (custos acumulados dos caminhos até cada voxel, em um dado instante),*

** o mapa de rótulos R,*

** o mapa de predecessores P.*

Variáveis auxiliares:

** a fila de prioridade Q vazia, gerenciada com as seguintes funções:*

** $RemoveMin(Q)$ remove da fila Q, o elemento de custo mínimo,*

** $Remove(p, Q)$ remove o elemento p da fila Q,*

** $Insere(p, Q, c)$ insere o elemento p na fila Q com o custo c,*

** $EstáVazia(Q)$ retorna um booleano indicando se a fila Q está vazia;*

** a tabela e de estado dos voxels associando a cada voxel p da imagem I o valor $e(p)$ igual a:*

** 0 se o voxel p não foi tratado e não está na fila Q,*

** 1 se p está em Q,*

** 2 se p já foi tratado definitivamente (portanto, está fora da fila);*

** a variável c para armazenar um custo (geralmente inteiro).*

1. $\forall p \in I$, $C[p] \leftarrow \infty$; $R[p] \leftarrow nil$; $P[p] \leftarrow nil$; $e(p) \leftarrow 0$;
2. $\forall s \in S$, $C[s] \leftarrow 0$; $R[s] \leftarrow \lambda(s)$; $Insere(s, Q, 0)$; $e(s) \leftarrow 1$;
3. ***Enquanto*** $EstáVazia(Q) = FALSO$,
4. $\quad p \leftarrow RemoveMin(Q)$; $e(p) \leftarrow 2$;
5. $\quad \forall q \in V(p)$, ***Se*** $e(q) \neq 2$,
6. $\quad\quad c \leftarrow f(\pi_p \cdot \langle p, q\rangle)$;
7. $\quad\quad$ ***Se*** $c < C[q]$,
8. $\quad\quad\quad$ ***Se*** $e(q) = 1$, $Remove(q, Q)$;
9. $\quad\quad\quad Insere(q, Q, c)$; $e(q) \leftarrow 1$;
10. $\quad\quad\quad C[q] \leftarrow c$; $R[q] \leftarrow R[p]$; $P[q] \leftarrow p$;

2.3.3 Implementação da fila hierárquica

A fila hierárquica ou fila de prioridade com *buckets* organiza seus elementos em níveis de prioridade, que correspondem a custos de caminho, no nosso caso. O *bucket* de prioridade mais alta corresponde ao *bucket* contendo os nós de custo menor. Em cada *bucket*, os nós são ordenados segundo uma política FIFO, respeitando-se, portanto, a ordem na qual são inseridos os nós.

Um elemento é inserido na fila, especificando-se o *bucket* ao qual ele deve pertencer, isto é, seu custo. Se o *bucket* já contiver elementos, insere-se aquele elemento no final da fila FIFO que constitui o *bucket*.

O elemento que é definitivamente removido da fila em cada passo do laço principal corresponde sempre ao primeiro elemento do *bucket* de custo menor que não esteja vazio. Pela hierarquia imposta pela fila e pelo caráter monotonamente crescente da função de custo de caminho, esse custo menor também é monotonamente crescente. Inicialmente nulo, ele é incrementado quando um *bucket* torna-se vazio até tomar o valor do próximo *bucket* não vazio. As filas FIFO internas a cada *bucket* cumprem implicitamente o papel do custo lexicográfico, descrito em (Lotufo e Falcão, 2000). Esta formulação do custo leva em conta o fator "tempo de propagação" em casos de empate entre árvores concorrentes, resultando numa propagação equitativa entre concorrentes, como evocado na introdução da seção 2.3.

A fila hierárquica tem que permitir também o deslocamento de um elemento, de um *bucket* para outro, o que equivale a poder remover um elemento de qualquer *bucket*.

Cunha descreve uma maneira eficiente de se implementar esta fila hierárquica (da Cunha, 2001), que permite para diversos operadores baseados na IFT uma execução linear em relação ao tamanho da cena. Resumidamente, a fila tem um tamanho minimizado, encontrando-se o maior peso de arco possível na cena; ela é utilizada de maneira circular; filas FIFO duplamente ligadas materializam os *buckets*.

2.3.4 *Watershed* via IFT

A implementação do *watershed* via IFT é uma das mais eficientes e, portanto, foi a abordagem investigada neste trabalho. Para adequar o algoritmo genérico da IFT ao problema do *watershed*, basta determinar a função f de custo de caminho, assim como o peso dos arcos.

Define-se o peso do arco $\langle p, q \rangle$ como a intensidade $I(q)$ do voxel associado ao nó q. A função de custo de caminho é a função de máximo. Portanto, o custo $C[p]$ do elemento p, ou seja $f(\pi_p)$, é o máximo das intensidades dos voxels associados aos nós constituindo o caminho π_p até p. Logo, no pseudocódigo (ver pseudocódigo B), $f(\pi_p \cdot \langle p, q \rangle)$ torna-se $MAX(C[p], I[q])$.

Pseudocódigo B ***Watershed via IFT***

Variáveis de entrada:

** a cena I (conjunto de voxels),*

** a vizinhança simétrica V: $V(p)$ designa o conjunto dos vizinhos do elemento p,*

** o conjunto S de voxels sementes rotuladas,*

** a função λ que associa um rótulo a cada semente de S.*

Variáveis de saída:

** o mapa de custos C (custos acumulados dos caminhos até cada voxel, em um dado instante),*

** o mapa de rótulos R,*

** o mapa de predecessores P.*

Variáveis auxiliares:

** a fila de prioridade Q vazia, gerenciada com as seguintes funções:*

** $RemoveMin(Q)$ remove da fila Q, o elemento de custo mínimo,*

** $Remove(p, Q)$ remove o elemento p da fila Q,*

** $Insere(p, Q, c)$ insere o elemento p na fila Q com o custo c,*

** $EstáVazia(Q)$ retorna um booleano indicando se a fila Q está vazia;*

** a tabela e de estado dos voxels associando a cada voxel p da imagem I o valor $e(p)$ igual a:*

** 0 se o voxel p não foi tratado e não está na fila Q,*

** 1 se p está em Q,*

** 2 se p já foi tratado definitivamente (portanto, está fora da fila);*

** a variável c para armazenar um custo (geralmente inteiro).*

1. $\forall p \in I$, $C[p] \leftarrow \infty$; $R[p] \leftarrow nil$; $P[p] \leftarrow nil$; $e(p) \leftarrow 0$;
2. $\forall s \in S$, $C[s] \leftarrow 0$; $R[s] \leftarrow \lambda(s)$; $Insere(s, Q, 0)$; $e(s) \leftarrow 1$;
3. ***Enquanto*** $EstáVazia(Q) = FALSO$,
4. $\quad p \leftarrow RemoveMin(Q)$; $e(p) \leftarrow 2$;
5. $\quad \forall q \in V(p)$, ***Se*** $e(q) \neq 2$,
6. $\quad\quad c \leftarrow MAX(C[p], I[q])$;
7. $\quad\quad$ ***Se*** $c < C[q]$,
8. $\quad\quad\quad$ ***Se*** $e(q) = 1$, $Remove(q, Q)$;
9. $\quad\quad\quad Insere(q, Q, c)$; $e(q) \leftarrow 1$;
10. $\quad\quad\quad C[q] \leftarrow c$; $R[q] \leftarrow R[p]$; $P[q] \leftarrow p$;

2.4 *Watershed* iterativo via DIFT

Para fazer frente ao problema da segmentação assistida, é preciso ter um algoritmo que seja iterativo, onde se possa corrigir um resultado de segmentação, voltando para o passo anterior, e onde se possa remover as partes mal segmentadas ou refinar certas partes ainda grosseiras demais. A IFT diferencial (*Differential Image Foresting Transform* - DIFT), proposta por Bergo e Falcão (Bergo e Falcão, 2003; Bergo, 2004), é

a versão iterativa da IFT. Com ela, pode-se implementar uma versão iterativa do *watershed*. O *watershed* iterativo é um método adequado para um processo interativo, guiado pelo usuário. Com efeito, ele permite aproveitar antigas sementes e resultados, assim como voltar para trás, para segmentar iterativamente e rapidamente uma cena.

2.4.1 Generalidades sobre a DIFT

O algoritmo da DIFT é apresentado no pseudocódigo D. Como para a IFT, a DIFT tem como variáveis de entrada a cena I, o conjunto S das sementes, a vizinhança V utilizada e a função suave de custo de caminho f. Além destas, para o algoritmo da DIFT poder aproveitar-se do resultado do passo anterior, são passados em entrada os três mapas, resultados do passo anterior: custos C, raízes R e predecessores P. Também se deve especificar o conjunto E de "eliminadores de árvore" escolhidos pelo usuário. Um voxel é escolhido como eliminador de árvore para notificar que a árvore à qual ele pertence tem que ser removida. Os eliminadores são muito úteis para corrigir regiões da segmentação.

Em saída do algoritmo, retornam-se igualmente três mapas do tamanho da cena de entrada: o mapa de custos C, o mapa das raízes R e o mapa dos predecessores P. Pode-se notar que se trabalha com o mapa das raízes R e não com o mapa dos rótulos, como no caso da IFT. Isto foi feito para simplificar a notação do pseudocódigo e agilizar o processo na segmentação, ao procurar raiz para remover árvores, como vamos ver no pseudocódigo C. No entanto, poder-se-ia muito bem trabalhar com o mapa dos rótulos.

Utilizam-se também a fila de prioridades Q e a função e de estado dos nós. Além destas, um conjunto auxiliar F contém os voxels-nós sementes e fronteiras, como mostra o pseudocógido C.

A estrutura geral do algoritmo é a seguinte (ver pseudocódigos C e D):

- Inicializações: a função "*DIFT_TreeRemoval*" (pseucódigo C) remove as árvores correspondentes a regiões a serem corrigidas, designadas pelo usuário e atualizam-se ou inicializam-se os diversos mapas (estado, custos, raízes, predecessores), assim como a fila de prioridades (vide figuras 2.3(a) e 2.3 (b));

- Laço principal de esvaziamento da fila: As novas sementes e os voxels de fronteira vão conquistar voxels da cena, propagando seu rótulo de vizinho para vizinho, segundo o critério de custo (vide figura 2.3(c)).

As diferenças fundamentais entre a DIFT e a IFT são a reutilização dos mapas obtidos no passo anterior e a reinicialização local (remoção de árvores) de voxels-nós, que disponibiliza-os para serem segmentados em outro objeto de interesse.

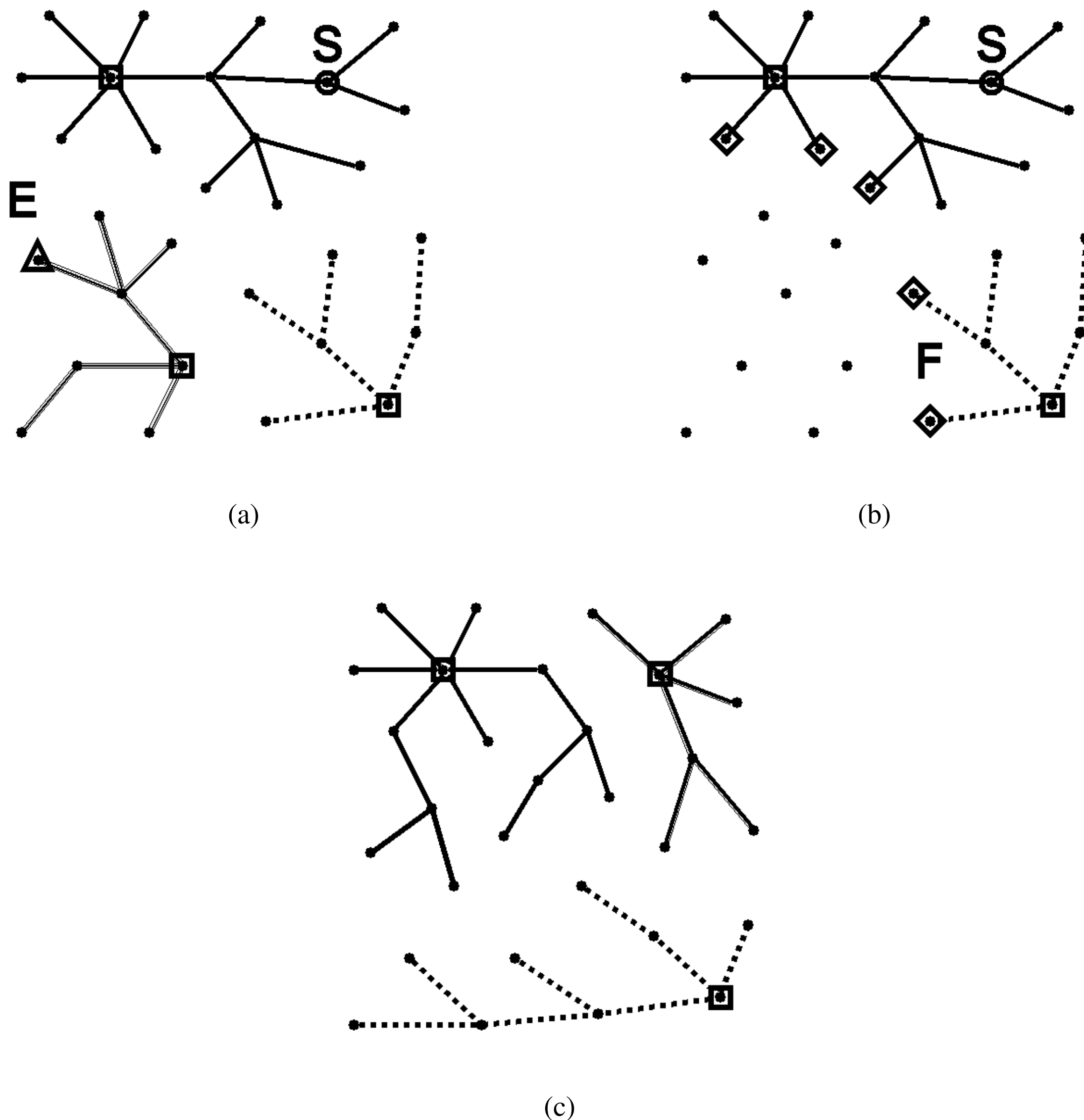


Figura 2.3: Algoritmo da DIFT: Etapas intermediárias dentro de uma iteração. **(a)** Situação inicial da iteração. A cena segmentada pode ser vista como um conjunto de árvores rotuladas. Os nós (voxels) são representados por pontos ligados por segmentos a seu predecessor. Os nós raízes aparecem com quadrado. O usuário especificou uma semente (S) com círculo e um eliminador (E) com triângulo. **(b)** Após remoção da árvore, nós são liberados para ser reconquistados. Semente (S) e nós de fronteira (F) sinalizados com losangos constituem as frentes de reconquista. **(c)** Situação final da iteração: Os nós livres foram invadidos pelas árvores vizinhas, enquanto outros já ocupados mudaram de árvore após reconquista.

2.4.2 Algoritmo

Após inicialização da tabela de estado dos voxels (pseudocódigo D linha 1), começa-se a tratar as árvores a serem removidas, selecionadas pelos eliminadores (linha 2). Essa tarefa é efetuada pela função "*DIFT_TreeRemoval*".

A partir dos conjuntos de sementes S e de eliminadores E, forma-se o conjunto F dos voxels sementes e voxels fronteiras (pseudocódigo C). Primeiro, todos os voxels pertencentes às árvores designadas por um eliminador de E são reinicializados com custo infinito e com nenhum predecessor, de maneira a poder ser reconquistados por outra árvore que se propaga. Esta reinicialização é feita por propagação, da raiz da árvore a ser removida até as folhas (l. 5-9). Encontram-se os voxels "de fronteira" que são vizinhos das folhas e nós reinicializados, porém, não pertencentes a árvores marcadas. Eles são, como as sementes, pontos de partida para uma nova propagação de rótulos e, portanto, são inseridos no conjunto F (l. 9). Enfim, inserem-se as sementes do conjunto S no conjunto F se seu *handicap* for inferior ao custo atual (l. 10). Cada semente recebe então o custo do caminho trivial (com um elemento só), chamado de *handicap* e geralmente nulo. Não tem predecessor e é sua própria raiz (l. 11).

Continuando na função principal "*DIFT*" (pseudocódigo D), inserem-se os voxels sementes e fronteiras na fila de prioridade (l. 3).

Em seguida, entra-se no laço central da DIFT. Enquanto a fila de prioridade não estiver vazia, remove-se o elemento de mais alta prioridade, isto é, de custo mínimo (l. 4-5). O elemento saindo da fila de prioridade tenta propagar seu rótulo em seus vizinhos que ainda não receberam rótulo permanente (l. 6). Para poder propagar o rótulo para o vizinho t, o nó s, que sai da fila, tem que oferecer um custo de caminho menor do que o atualmente associado a t, ou então, s tem que ser pai deste vizinho t (l. 7-8). Aqui está outra grande diferença com o algoritmo da IFT. Com efeito, o elemento s está saindo da fila porque ele assumiu novos valores de rótulo, custo e predecessor: foi reconquistado. Logo, o seu vizinho t, que era nó filho no passo anterior, tinha um custo relacionado ao seu caminho e portanto, aos antigos valores do nó pai. Agora que o pai foi reconquistado, nada garante que este mesmo ofereça ainda o menor custo; e mesmo que oferecesse, o nó pai pode ter mudado de rótulo e, portanto, este rótulo tem que ser atualizado para seus nós filhos. Por isso, no algoritmo da DIFT, cada elemento saindo da fila vai forçar a liberação dos seus elementos descendentes para uma futura reconquista.

Se for possível a propagação do rótulo de s, atualizam-se tanto o rótulo do vizinho t com o rótulo de s, como o custo de caminho de t recentemente calculado. O nó s torna-se, conseqüentemente, o predecessor (pai) do nó t (l. 11). Como o nó t assume novos valores, ele tem que ser inserido na fila de prioridades com seu novo custo (l. 10). Caso ele já esteja na fila, deve ser previamente removido da fila (l. 9), para não constar duas vezes na fila com dois custos diferentes.

O segredo do algoritmo da DIFT é esse reaproveitamento do passo anterior, ou seja, conservação da maior parte dos cálculos de segmentação feitos, junto à liberação para uma reconquista dos elementos mal

segmentados e dos que devem sofrer mudança de segmentação para a corretude do algoritmo iterativo. As iterações do algoritmo da DIFT processam apenas esses elementos liberados para a reconquista, que são, a princípio, bem menos numerosos que os elementos da cena total. Logo, espera-se um ganho significativo em tempo de execução nos passos seguindo a primeira iteração.

Pseudocódigo C ***DIFT_TreeRemoval***

Variáveis de entrada:

** o mapa de custos C,*

** o mapa de raízes R,*

** o mapa de predecessores P,*

** a vizinhança simétrica V: $V(p)$ designa o conjunto dos vizinhos do elemento p,*

** o conjunto S de voxels sementes,*

** o conjunto E de voxels eliminadores.*

Variáveis de saída:

** o mapa de custos C,*

** o mapa de predecessores P,*

** o conjunto F (inicialmente vazio) de voxels sementes ou de fronteira.*

Variáveis auxiliares:

** a fila FIFO T vazia, gerenciada com as seguintes funções:*

** $Insere(r, T)$ insere o elemento r na fila FIFO T,*

** $Remove(T)$ remove um elemento da fila T segundo a ordem FIFO e retorna este,*

** $EstáVazia(T)$ retorna um booleano indicando se a fila FIFO T está vazia;*

** o conjunto D (inicialmente vazio) das raízes das árvores a serem removidas.*

1. $\forall t \in E$,
2. $r \leftarrow R[t]$;
3. **Se** $C[r] \neq \infty$,
4. $D \leftarrow D \cup \{r\}$; $C[r] \leftarrow \infty$; $P[r] \leftarrow nil$; $Insere(r, T)$;
5. **Enquanto** $EstáVazia(T) = FALSO$,
6. $s \leftarrow Remove(T)$;
7. $\forall t \in V(s)$,
8. **Se** $P[t] = s$, $C[t] \leftarrow \infty$; $P[t] \leftarrow nil$; $Insere(t, T)$;
9. **Senão, Se** $R[t] \notin D$, $F \leftarrow F \cup \{t\}$;
10. $\forall t \in S$, ***Se*** $f(\langle t \rangle) < C[t]$,
11. $F \leftarrow F \cup \{t\}$; $C[t] \leftarrow f(\langle t \rangle)$; $P[t] \leftarrow nil$; $R[t] \leftarrow t$;

Pseudocódigo D ***DIFT***

Variáveis de entrada:

** a cena I (conjunto de voxels),*

** o mapa de custos C (custos acumulados dos caminhos até cada voxel, em um dado instante),*

** o mapa de raízes R,*

** o mapa de predecessores P,*

** a vizinhança simétrica V: $V(p)$ designa o conjunto dos vizinhos do elemento p,*

** o conjunto S de voxels sementes rotuladas,*

** o conjunto E de voxels eliminadores (para indicar as partes a serem removidas),*

** uma função suave de custo de caminho f que associa ao caminho π_p o custo de percurso $f(\pi_p)$.*

Variáveis de saída:

** o mapa de custos C,*

** o mapa de raízes R,*

** o mapa de predecessores P.*

Variáveis auxiliares:

** a fila de prioridade Q vazia, gerenciada com as seguintes funções:*

** $RemoveMin(Q)$ remove da fila Q, o elemento de custo mínimo,*

** $Remove(p, Q)$ remove o elemento p da fila Q,*

** $Insere(p, Q, c)$ insere o elemento p na fila Q com o custo c,*

** $EstáVazia(Q)$ retorna um booleano indicando se a fila Q está vazia;*

** a tabela e de estado dos voxels associando a cada voxel p da imagem I o valor $e(p)$ igual a:*

** 0 se o voxel p não foi tratado e não está na fila Q,*

** 1 se p está em Q,*

** 2 se p já foi tratado definitivamente (portanto, está fora da fila);*

** a variável c para armazenar um custo (geralmente inteiro),*

** o conjunto F vazio.*

1. $\forall t \in I$, $e(t) = 0$;

2. $(C, P, F) \leftarrow DIFT_TreeRemoval(C, R, P, V, S, E)$;

3. $\forall t \in F$, $Insere(t, Q, C[t])$; $e(t) \leftarrow 1$;

4. **Enquanto** $EstáVazia(Q) = FALSO$,

5. $s \leftarrow RemoveMin(Q)$; $e(s) \leftarrow 2$;

6. $\forall t \in V(s)$, ***Se*** $e(t) \neq 2$,

7. $c \leftarrow f(\pi_s \cdot \langle s, t \rangle)$;

8. **Se** $c < C[t]$ **ou** $P[t] = s$,

9. **Se** $e(t) = 1$, $Remove(t, Q)$;

10. $Insere(t, Q, c)$; $e(t) \leftarrow 1$;

11. $C[t] \leftarrow c$; $R[t] \leftarrow R[s]$; $P[t] \leftarrow s$;

2.4.3 Observações

Vale a pena ressaltar aqui o problema de empate já evocado na introdução da seção 2.3. Vimos que o problema de empate (quando árvores concorrentes tentam conquistar com um mesmo custo os mesmos nós) não é resolvido pela IFT, pois existem ainda várias soluções ótimas; mas é contornado, pois a IFT propõe uma solução ótima atrativa, onde a linha de *watershed* é centrada entre os concurrentes. A solução dada pela IFT será sempre a mesma, contando que as entradas (cena e sementes) estejam as mesmas.

No caso da DIFT, a situação é mais complicada, pois o resultado final da segmentação depende de várias iterações e não só de um passo como ocorre com a IFT. Pode acontecer o que chamamos do caso de "semente infértil". Este acontece quando uma semente não consegue se propagar, isto é, o custo proposto por esta é sempre maior que o custo já armazenado pelos elementos vizinhos. Ela é uma semente "infértil".

Geralmente, o custo escolhido para a semente inserida (*handicap*) é nulo. Portanto, o único caso para o qual a semente não se propaga, é quando ela tem vizinhos oferecendo um custo nulo igualmente e é folha de árvore (vide linha 8 do pseudocódigo D).

Se o usuário quiser que a semente infértil reconquiste nós, ele pode forçá-la a crescer removendo a árvore dos voxels vizinhos. Desta maneira, os voxels serão reinicializados, o que autorizará reconquistas. O usuário tem que ter isto em mente. Se, ao acrescentar sementes, ele observar que estas não cresceram, ele deverá escolher eliminadores nas regiões que não se conseguem reconquistar: As próprias sementes podem ser também eliminadoras, por exemplo.

Essa incapacidade de crescer é um fenômeno mais generalizado que acontece quando a árvore, apesar de ter crescido, não consegue reconquistar nós, pois propõe um custo igual ao custo dos nós, atribuído na iteração anterior. Esse caso geral de empate é um pouco diferente do empate na IFT, pois ele não ocorre entre árvores concorrentes em crescimento, mas sim, entre uma em crescimento e outra já enraizada numa iteração anterior da DIFT. O empate desse tipo, "assíncrono" (que inclui o caso das sementes inférteis), sempre será resolvido pela DIFT, dando prioridade à árvore mais antiga, o que pode ser contrário à vontade do usuário. Se um usuário acrescenta uma semente depois de várias iterações, é para corrigir a segmentação de uma região particular. Portanto, seria desejável que a árvore correspondente crescesse de maneira equitativa com árvores antigas. A idéia de um trabalho futuro seria corrigir esses casos de empates assíncronos, reinserindo automaticamente as árvores antigas participando de empates. Obviamente isso teria um custo adicional de processamento e, portanto, um tempo de execução da DIFT mais lento.

Por isso, não foi corrigida a DIFT neste trabalho, mas lembrando que na prática os empates assíncronos podem ocorrer e ser resolvidos acrescentando eliminadores. A conseqüência desses empates assíncronos é que a ordem de inserção das sementes e eliminadores influencia o resultado da segmentação. Portanto, diferentemente da IFT, a DIFT não segmenta necessariamente uma cena da mesma maneira, dado um conjunto de sementes e eliminadores. Por isso, entra em conta a noção de "roteiro", especificando quais sementes e eliminadores inserir em cada passo, quando se comparam resultados da DIFT (ver seção 5.1.5).

Capítulo 3

Visualização tridimensional

As seções seguintes fazem uma revisão das noções de base em visualização e apresentam e classificam as principais abordagens existentes em visualização tridimensional (3D). Discute-se apenas a visualização tridimensional para tela de computador, isto é, a criação de uma imagem digital bidimensional (2D) que representa uma cena tridimensional (3D) e dá apenas a ilusão da terceira dimensão. Nota-se que existe, no entanto, tecnologias como a holografia e os espelhos vibrantes que constituem "telas tridimensionais", permitindo a criação de uma "imagem 3D da cena".

3.1 Visão geral das abordagens em visualização tridimensional

A visualização tridimensional pode ser vista como uma série de transformações aplicadas a uma cena para se obter uma imagem 2D na tela do computador. Antes de detalhar essas transformações, são relembrados alguns pré-processamentos a serem feitos para obter a cena.

3.1.1 Pré-processamento: Da fatia à cena

Os equipamentos de CT e RM amostram propriedades de tecidos do corpo humano e reprentam-nas sob formas de imagens seccionais retangulares. Essas são compostas de elementos mínimos rectangulares, os pixels (*picture elements*). Cada pixel tem um valor chamado densidade, nível de cinza ou tom de cinza, correspondente à medida da propriedade amostrada. Empilhando-se essas imagens de fatias paralelas, pode-se formar uma cena. O elemento mínimo da cena, pequeno paralelepípedo, é chamado voxel (*volume element*). Os pixels empilhados tornam-se voxels e conservam seu valor de cinza. Portanto, Z imagens de dimensões $X \times Y$ pixels formam uma cena de dimensão $X \times Y \times Z$ voxels, se forem colocadas lado ao lado.

Como nem toda a cena de origem é interessante para o usuário, extrai-se geralmente um volume de interesse (*Volume Of Interest*, VOI): Seleciona-se uma região da cena (comumente paralelepipédica) e

descarta-se o resto dos voxels. Obtém-se uma cena menor que ocupa menos espaço de memória e será processada posteriormente em menos tempo.

Pode ser aplicada em seguida uma série de transformações para obter melhores resultados nos processamentos posteriores. Podem-se transformar os níveis de cinza sem olhar a vizinhança dos voxels: alargamento de contraste, compressão da escala dinâmica, processamento do histograma da cena (normalização, equalização). Também pode-se olhar a vizinhança dos voxels no caso dos filtros espaciais (filtros lineares ou não, como os passa-baixas, os passa-altas, os passa-banda, o filtro por mediana) ou dos filtros morfológicos (fechamento, abertura, por exemplo). A cena é quase sempre filtrada, a fim de diminuir os ruídos devidos à aquisição e realçar ou suavizar certas partes da cena. Maiores informações sobre filtragem podem encontrar-se em (Gomes e Velho, 1994; Gonzalez e Woods, 1993).

Outra operação importante freqüentemente efetuada sobre as cenas é a interpolação. A amostragem feita pelos tomógrafos é geralmente anisotrópica, isto é, o espaço entre as diferentes fatias é maior que as dimensões dos pixels de cada imagem. Por razões de exatidão e precisão, é desejável, porém, ter uma cena com voxels cúbicos e não paralelepipédicos. Por isso, interpola-se a cena, criando-se voxels intermediários por exemplo, com densidade dada por uma certa regra e calculada a partir das densidades dos vizinhos. As interpolações mais utilizadas são a interpolação do vizinho mais próximo e a interpolação trilinear.

Enfim, a segmentação, como explicado no capítulo anterior (seção 2.1), seja baseada em borda, seja em regiões, é muitas vezes aplicada antes de visualizar a cena, para identificar as estruturas de interesse. Nota-se que filtragem, interpolação e segmentação não são necessariamente aplicadas nessa ordem.

3.1.2 Da cena à imagem

Depois desses pré-processamentos, a visualização consiste de várias transformações, antes de se obter a imagem na tela do computador. O diagrama da figura 3.1 mostra as possíveis transformações aplicadas na cena. É chamado de *espaço da cena* o espaço 3D no qual é definido a cena. Como visto na seção anterior, podem ser feitos vários pré-processamentos na cena. A segmentação, no entanto, tem a particularidade de extrair estruturas da cena, o que resulta numa cena segmentada, explicitada no *espaço dos objetos*. Transformações geométricas tais rotações, translações ou escalamentos nos objetos segmentados (regiões ou bordas) permitem passar no *espaço da imagem* (vide apêndice A para mais detalhes sobre a matemática dessas transformações). Enfim, por transformações de projeção, obtém-se uma imagem bidimensional, definida no *espaço de visualização* (*view-space*), ele mesmo ligado ao espaço físico da tela. A projeção é responsável da perda da terceira dimensão. Para ter a impressão de profundidade, várias ilusões são então criadas: tonalização, remoção de superfícies ou partes escondidas, transparência, visão estéreo, animação. Nesse caso, alguns autores falam de imagem 2D e $\frac{1}{2}$ para lembrar que a imagem permanece bidimensional, apesar de dar a impressão de uma terceira dimensão. Pode ser preciso fazer medidas e extrair alguns parâmetros da cena, a partir da imagem 2D. Formalmente, pode-se dizer que os parâmetros são então definidos no *espaço dos parâmetros*.

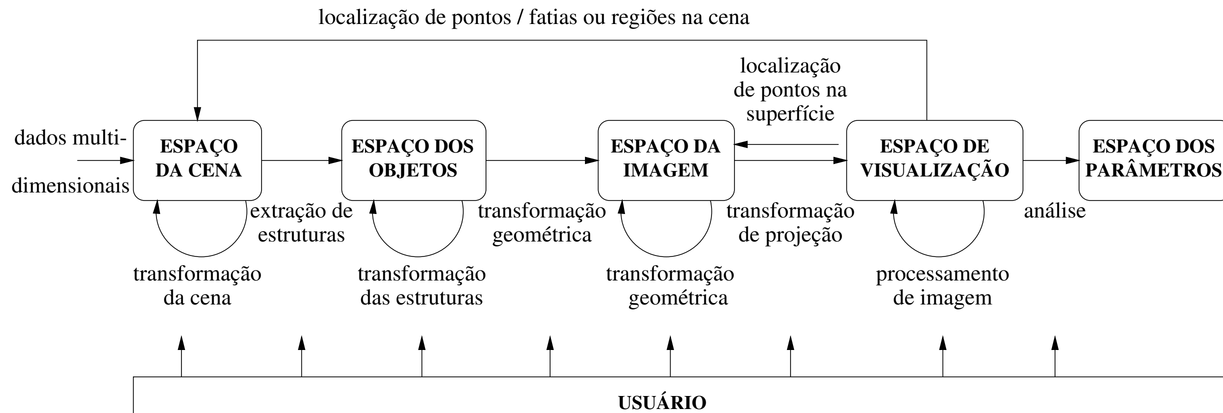


Figura 3.1: Representação esquemática das transformações aplicadas a uma cena para se obter uma imagem e suas características. (adaptada de Udupa e Herman)

3.1.3 Classificação das abordagens em visualização de dados tridimensionais

Nesta seção classificam-se, segundo suas estratégias, os diversos métodos de visualização de dados 3D conhecidos na literatura. A figura 3.2, adaptada de (Udupa e Herman, 1991) resume essa classsificação. Apesar de não existir padrão de nomenclatura na área de visualização, adoptamos os nomes que faziam mais lógica com a classificação e que não criavam confusões. Outros nomes freqüentemente usados são citados. A classificação seguinte é um consenso feito a partir de várias publicações (Udupa e Herman, 1991; Stytz et al., 1991; Elvins, 1992; Kaufman, 1996; Owen, 1999). Essa classificação é importante por apresentar a essência de cada método. O problema de nomenclatura é real nessa área, pois um nome pode designar métodos totalmente diferentes, além de não ter sempre uma boa tradução fixa em português.

A visualização de dados tridimensionais, comumente chamada de visualização 3D (*3D imaging* ou *volume visualization*), pode ser dividida em três grandes tipos de visualização, segundo a dimensionalidade do suporte físico da visualização e a do espaço de análise. Primeiro, a visualização de fatia (*slice imaging*) só permite análise bidimensional (2D) da cena, como se fosse extraída uma fatia da cena e vista no display 2D do computador. Esse tipo de visualização, embora não esteja dentro do assunto deste trabalho, é de grande utilidade nos sistemas atuais com interface, processando imagens médicas. Vistas de corte 2D são complementares à vista tridimensional. Em medicina, as vistas mais comuns são definidas sobre planos perpendiculares aos três eixos ortogonais principais do paciente: os cortes coronal (vista de frente), sagital (de perfil) e axial (transversal). Cortes oblíquos também são necessários para examinar certas estruturas segundo outra direção. Portanto, efetua-se um refatiamento (*reslicing*) da cena, ortogonal a essa direção. Existe, além disso, o refatiamento curvado, seguindo uma direção curvilínea. As referências (Rhodes et al., 1980; Maravilla, 1978; Rothman et al., 1984) são das mais antigas publicações sobre o problema de refatiamento oblíquo e curvilíneo.

O segundo tipo, de interesse neste trabalho, é aquele que é diretamente e implicitamente associado ao termo “visualização 3D”: a visualização por projeção (*projective imaging*). Desta vez, o espaço de análise

é 3D, enquanto o suporte do display permanece 2D. Como já foi dito, é a chamada visualização 2D $\frac{1}{2}$. Dentro deste tipo, podem-se distinguir duas famílias de métodos: o *rendering* de superfície (*surface rendering*) e o *rendering* de volume (*volume rendering*). Formalmente, ambas consistem em resolver os dois problemas do *rendering* (Yagel, 1997): o mapeamento dos voxels da cena nos pixels da imagem (projeção) e a remoção das partes ou objetos escondidos. Na primeira família de *rendering*, há formação explícita das superfícies das estruturas de interesse para poder renderizar a cena. Na segunda, as primitivas de *rendering* são os voxels do volume original. Essas duas famílias de *rendering* são detalhadas nas duas seções seguintes (3.2 e 3.3).

Enfim, para continuar na lógica desta classificação, citamos, como na introdução do presente capítulo, a visualização de volume por "imagem tridimensional" (*volume imaging*): Tanto a análise quanto o suporte físico da imagem são 3D. Seria a "verdadeira" visualização tridimensional. Mas, já que descartamos esse tipo de visualização, de agora em diante, falar-se-á apenas da visualização projetiva como a visualização 3D.

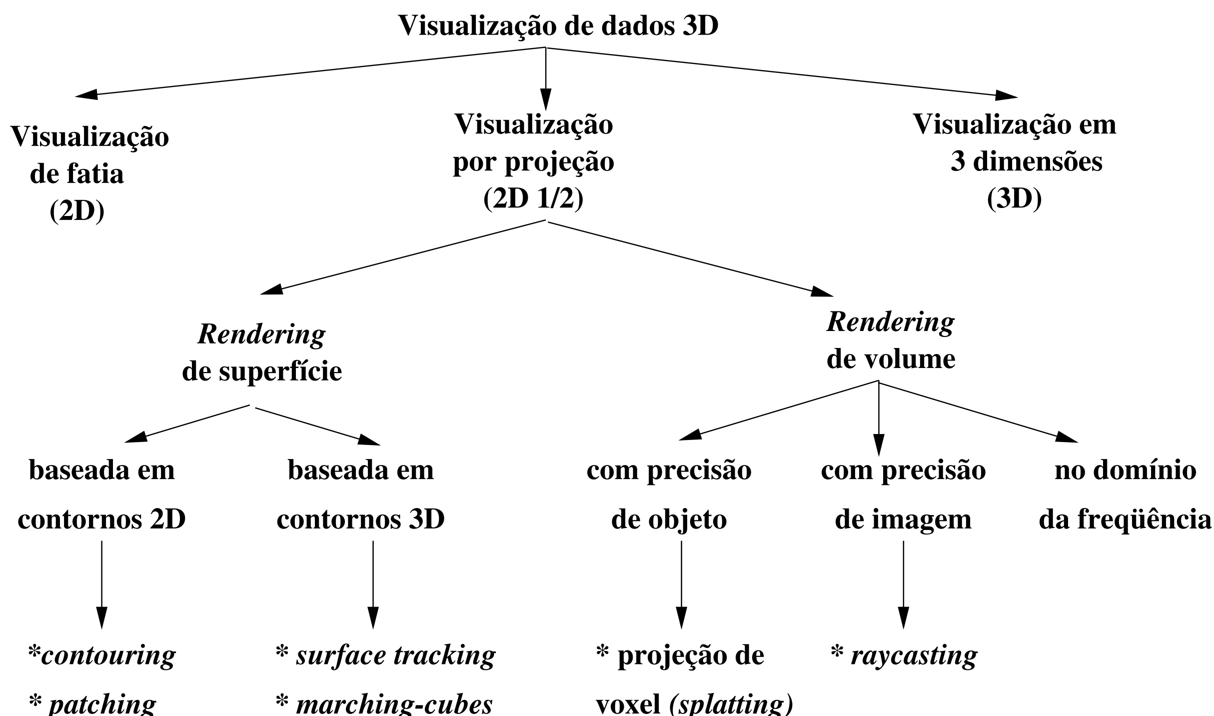


Figura 3.2: Classificação das abordagens em visualização de dados tridimensionais.

3.2 *Rendering* de superfície

O *rendering* de superfície (*surface rendering*) também é chamado de *rendering* indireto de volume (*Indirect Volume Rendering*, IVR), pois se extraem primeiro as superfícies das estruturas de interesse para poder renderizá-las em seguida. O *rendering* de superfície mostra *indiretamente* os dados originais,

modelando-os com a ajuda de um algoritmo de *surface fitting*. O *rendering* de superfície opõe-se ao *rendering* de volume que é um método de *rendering* direto (*Direct Volume Rendering*, DVR).

3.2.1 Extração de superfície

No primeiro passo do *rendering* de superfície (extração de superfície), cria-se um conjunto de primitivas geométricas que envolvam o objeto de interesse e sejam fáceis de renderizar. Essa transformação do espaço da cena para o espaço dos objetos pode ser abordada de várias maneiras. Um primeiro grupo de métodos baseia-se em contornos *2D* do objeto para formar a superfície de envoltura, enquanto um segundo já produz superfícies *3D* na borda dos objetos.

3.2.1.1 Superfície baseada em contornos 2D

3.2.1.1.1 ***Contouring*** Neste método por contornos 2D, linhas 2D contornando os pixels da estrutura são determinadas automática ou semi-automaticamente e individualmente em cada fatia 2D já binarizada (segmentação por região). Cada contorno fechado é estendido (ortogonalmente) na terceira dimensão, conhecendo-se o espaçamento entre as fatias. O empilhamento desses contornos estendidos forma o contorno 3D da estrutura.

3.2.1.1.2 ***Patching*** Também se usam as linhas de contorno 2D do método precedente mas essas, suficientemente suaves, são ligadas por retalhos (*patches*): pequenas superfícies elementares planas tridimensionais como os triângulos (Keppel, 1975; Fuchs et al., 1977; Ekoule et al., 1991). O ladrilho desses *patches* constitui a superfície 3D da estrutura. O método é similar ao anterior, só que ele interpola na ordem 1 (em vez da ordem 0) a ligação entre contornos 2D. Esse método, porém, não é muito utilizado com imagens médicas, pois a ligação entre contornos de duas fatias pode levar em falsas superfícies ou falsas topologias quando aparecem buracos na estrutura de interesse.

3.2.1.2 Superfície baseada em contornos 3D

3.2.1.2.1 ***Surface tracking*** O método também é chamado de método dos cubos opacos. A estratégia de perseguição de superfície consiste em percorrer os voxels (cubinhos) de borda da estrutura. A superfície da estrutura de interesse é o conjunto das faces dos voxels percorridas (Artzy et al., 1981; Morgenthaler e Rosenfeld, 1981; Udupa et al., 1982; Herman e Webster, 1983; Udupa, 1987; Gordon e Udupa, 1989). Este método tem a desvantagem de dar um aspecto denteado aos objetos visualizados.

3.2.1.2.2 ***Marching Cubes*** O algoritmo de *Marching Cubes*, introduzido por Lorensen et al. (Lorensen e Cline, 1987), é agora um método clássico de extração de iso-superfícies. Nesse algoritmo, define-se a

partir de um valor de limiarização dado pelo usuário, uma iso-superfície constituída de pequenos triângulos (*patches*). Em função de seu valor, os voxels são classificados como internos ou externos à iso-superfície. Se cada voxel for visto como um vértice, a cena forma então um conjunto de cubos delimitados por esses vértices. O valor dos vértices (voxels) de um cubo informa sobre a localização da interseção entre a iso-superfície e o cubo. Existe uma interseção com a aresta de um cubo se um dos vértices dessa aresta é externo e outro é interno. Como cada um dos 8 vértices de um cubo pode ser interno ou externo, existem $2^8 = 256$ maneiras possíveis da iso-superfície cruzar o cubo. Por simetria esses casos reduzem-se a 15. Uma tabela de tipo *LookUp Table* (LUT) dá rapidamente o tipo de interseção com as arestas de um cubo e os triângulos necessários para modelizar a iso-superfície em cada cubo. A interseção exata com as arestas são então determinadas por interpolação linear e calculam-se as normais em cada vértice para a tonalização posterior.

Observa-se que, no entanto, podem ocorrer problemas topológicos na definição da superfície. Para evitá-los, não se simplificam os 256 casos. A principal desvantagem para este método é o tempo de extração da superfície. Uma vez a superfície extraída, a visualização segundo qualquer ponto de observação torna-se rápida.

A partir deste método, desenvolveram-se outros métodos análogos, como o *Dividing Cubes* e o *Marching Tetrahedra*. O algoritmo de *Dividing Cubes* é similar ao *Marching Cubes*, exceto que quando um cubo é cruzado pela iso-superfície, ele é projetado no plano da imagem. Se sua projeção é contida num só pixel, ele é renderizado como um ponto. Senão, ele é subdividido em superfícies como no *Marching Cubes*. Há, portanto, perda da topologia dos objetos. O *Marching Tetrahedra* propõe trabalhar com tetraedros ao invés de triângulos e reduz, desta maneira, o tamanho da tabela LUT para 3 casos.

3.2.1.3 Comparação dos métodos de extração de superfície

A perseguição de borda tem uso limitado. Quando nenhum hardware especializado é disponível, o algoritmo de *dividing cubes* é mais aconselhado.

Uma vez as superfícies extraídas, aparecem claramente as vantagens dos métodos de *rendering* de superfície. O *rendering* das superfícies pode ser feito por hardware especializado, pois é um processo bem conhecido em computação gráfica: sombras, *depth cueing*, reflexões, etc. O uso de *display list* em memória acelera o processo (exemplo em OpenGL). A mudança de ponto de vista e luzes requer apenas um novo *rendering*. Além disso, é uma forma mais compacta para se armazenar ou transmitir a cena. Pode ser renderizado em precisão de objeto ou de imagem. Tem uma boa coerência espacial para *renderings* mais eficientes. As desvantagens são: a necessidade de ter segmentado ou classificado de maneira binária anteriormente como objeto ou fundo. Os dados contidos entre as superfícies são perdidos. Não trata bem os pequenos detalhes nem as ramificações. Os dados amorfos, por essência, não podem ser modelados por finas superfícies. A extração de superfícies é geralmente um processo bastante demorado, comparado com um *rendering* direto de volume.

3.2.2 *Rendering*

Após o passo de extração de superfície, é preciso fazer o *rendering* propriamente dito: mostrar essas superfícies, formando uma imagem 2D na tela. Essa operação depende da direção de observação e das condições de iluminação. Consiste essencialmente da projeção (geralmente ortogonal para os dados médicos) das superfícies numa imagem 2D, com remoção das superfícies não visíveis do ponto de observação, seguida, geralmente, pela tonalização. Tanto a remoção de superfícies escondidas quanto a tonalização ou outras técnicas (visão estéreo, animação) encarregam-se de compensar a perda, durante a projeção, da terceira dimensão. Elas determinam a qual ponto da superfície corresponde cada pixel da imagem e, portanto, qual valor de intensidade ele deve assumir em função das condições de iluminação e da orientação da superfície associada. Esses processos pertencem à área da computação gráfica.

Com efeito, o *rendering* das primitivas geométricas é bem conhecido em computação gráfica e hoje em dia, as diversas técnicas de projeção, remoção de superfícies e tonalização são disponíveis em hardwares especializados como as placas vídeo gerenciadas por interfaces tais como OpenGL. Maiores detalhes sobre essas técnicas encontram-se em (Foley et al., 1990; Woo et al., 1999). A seção 3.4 relembra alguns conceitos de tonalização, comuns aos *renderings* de superfície e de volume.

3.3 *Rendering* de volume

Ao contrário do *rendering* de superfície, o *rendering* de volume não extrai nenhuma superfície e não utiliza nenhuma modelagem de objeto. Os métodos de *rendering* de volume apoiam-se *diretamente* sobre os dados originais para formar a imagem. Por isso, são também chamados de *rendering* direto de volume (*Direct Volume Rendering*, DVR).

Podem ser aplicados em dois tipos de dados:

- os volumes binários (ou n-ários), depois de uma segmentação prévia do(s) objeto(s) de interesse;
- os volumes não-binários, em tons de cinza, sem segmentação prévia.

Neste último caso, uma classificação dos voxels ocorre no processo de visualização. Ela é, em essência, um tipo de segmentação, só que implícita. Não se separam objetos do fundo, mas associam-se os voxels com materiais específicos, via funções de tranferência que atribuem opacidade em função dos níveis de cinza dos voxels. A opacidade de um material é dado por um valor α entre 0 e 1. Um material de opacidade nula é totalmente transparente e, portanto, é invisível. Ao contrário um material de opacidade unitária é totalmente opaco e não permite ver o que estiver atrás dele. Nos outros casos, o material é visível e permite ver os objetos escondidos: É um material semi-transparente. A classificação permite transformar uma cena em níveis de cinza numa cena semi-transparente em níveis de opacidade. Durante o processo de *rendering* propriamente dito, uma composição das opacidades dos materiais visíveis é efetuada para obter a

imagem final. Nesse contexto, a visualização de volumes binários pode ser vista como um caso particular, onde os voxels de objeto têm opacidade unitária e os de fundo opacidade nula.

Existem três estratégias de *rendering* de volume:

- Os métodos com precisão de objeto (*object-order*), relativos à cena.
- Os métodos com precisão de imagem (*image-order*), relativos à imagem.
- Os métodos freqüenciais (*domain-order*), mudando de espaço para operar num espaço freqüencial.

3.3.1 Métodos com precisão de objeto

3.3.1.1 Projeção de voxels

Os métodos com precisão de objeto são simplesmente projeções de voxels, chamadas também de *forward projections*. Eles consistem em varrer os voxels da cena e calcular qual é a projeção destes no plano da imagem, cuja posição e orientação é definida pela posição do observador. Detalhes matemáticos sobre a projeção, uma rotação no caso ortogonal, são dados no apêndice A. O algoritmo de projeção de voxel utiliza um *z-buffer*: um mapa de pixels do tamanho da imagem, só que armazena as distâncias dos voxels projetados nos pixels, ao invés da intensidade luminosa. Ao projetar sobre o plano, determina-se em qual pixel mais próximo, o voxel se projeta. Caso vários voxels se projetem no mesmo pixel, temos que determinar qual é visível (no caso de um volume binário, opaco), isto é, qual é o voxel mais perto do observador. Por isso, temos três estratégias possíveis.

A primeira e mais clássica estratégia consiste em comparar a distância armazenada no *z-buffer* com a distância do voxel projetado, candidato a ser visível. Se esta última for menor, o *z-buffer* é atualizado, assim como o mapa que conserva as informações sobre os voxels projetados (índice de localização, índice do objeto por exemplo). Caso contrário, nenhuma atualização é efetuada (vide o pseudocódigo a seguir) .

Para todo voxel $V(x, y, z) \in Cena$,
  Projetar V: $(u, v, d) \longleftarrow M_{proj} \cdot (x, y, z)$;
  Se $d \leq ZBuffer(u, v)$,
    $ZBuffer(u, v) \longleftarrow d$;
    $MapaIndices(u, v) \longleftarrow V$;

Na segunda estratégia, varre-se a cena no sentido de trás para frente (*back-to-front*, BTF) em relação ao observador (Frieder et al., 1985). Cada novo voxel candidato a ser projetado num pixel é projetado,

pegando sistematicamente o lugar do antigo voxel projetado, pois com toda certeza, pela ordem de varredura, o último fica mais perto do observador. Essa abordagem fica geralmente mais eficiente por economizar as comparações com o *z-buffer*. No entanto, o acesso BTF para qualquer direção de observação é possível com um custo adicional em espaço memória (para armazenar o volume nas três direções principais).

Enfim, pode-se varrer a cena no sentido oposto (*front-to-back*, FTB). Cada novo voxel candidato a ser projetado num pixel é realmente projetado se o pixel ainda está livre (se nenhum voxel já foi projetado).

3.3.1.2 Otimizações em qualidade e eficiência

Esses métodos de precisão de objeto têm duas desvantagens do ponto de vista de eficiência e qualidade do resultado:
- ter que varrer a cena toda e projetar os n voxels de objeto, levando a uma complexidade de algoritmo $O(n)$.
- gerar uma imagem com buracos, quando o tamanho do voxel é maior que o do pixel.

Para reduzir essa segunda desvantagem, pode-se projetar cada voxel num pixel e sua vizinhança. Esse método é chamado de *splatting* (Westover, 1990). É como se o voxel fosse projetado e espalhado localmente no plano da imagem. Um voxel é representado por uma nuvem de pontos (*footprint*) na imagem. Observa-se que esse espalhamento dos voxels projetados pode ser tanto uma melhoria da imagem com buracos como uma degradação. Com efeito, quanto maior for a *footprint*, mais borrada parece a imagem. Um bom tamanho de *footprint* é relacionado à razão do tamanho do pixel sobre o do voxel. Usa-se uma *footprint* maior quando esta razão é menor. Mais recentemente, foram propostos métodos mais sofisticados de *splatting* com a preocupação da qualidade da imagem. Métodos corrigindo artefatos de *popping* e borrado são relatados em (Mueller e Crawfis, 1998; Mueller et al., 1999). Yagel et al., mais interessados na eficiência do *splatting*, representam cada voxel como um pequeno mapa de textura (do tamanho do *footprint*) perpendicular à direção de observação, e renderizam esses voxels “nebulosos” via hardware especializado em mapeamento de textura (Yagel et al., 1995).

Outras otimizações do algoritmo de base visando maior eficiência foram publicadas. O método de transformação incremental para a projeção evita multiplicações matriciais, calculando-se cada projeção em função da anterior (Machiraju e Yagel, 1993). Por voxel, é necessário apenas uma adição ao invés de duas adições mais três multiplicações de *floats*. Uma versão paralela do algoritmo incremental com *splatting* BTF é possível.

A fatoração *shear-warp* (SW), proposta por Lacroute (Lacroute e Levoy, 1994; Lacroute, 1995), equivale a uma fatoração da matriz de projeção (3D) em duas matrizes de transformação: um cisalhamento 3D (*shear*), bastante rápido que desloca as fatias do volume, seguido de uma transformação 2D (*warp*) para tirar as distorções inseridas na imagem intermediária e obter uma imagem final correta. A transformação *shear* faz com que as fatias se tornem perpendiculares à direção de visualização. Desta maneira, a projeção ortogonal que ocorre depois da operação *shear* não necessita cálculos extra para a projeção dos voxels. O

método proposto utiliza também estruturas de dados particulares para acelerar o processo de visualização. Este método tornou-se bastante popular por ser facilmente paralelizável. Schmidt et al. (Schmidt et al., 2000) propõem uma visualização baseada nele, misturando dados médicos reais com modelos poligonais. O apêndice D dá o detalhamento das operações necessárias à fatoração SW.

O *shell rendering* proposto por Udupa e Odhner (Udupa e Odhner, 1993) utiliza uma estrutura de dados muito compacta: *shell* (casca). Esta armazena atributos de visualização (opacidade, índice, normal...) para voxels da cena que são potencialmente visíveis, ou seja, os voxels das bordas dos objetos. No caso de volume binário, basta achar os voxels de objeto (opacos) que têm pelo menos um vizinho de fundo (transparente). Se o volume for em tons de cinza, são selecionados voxels com opacidade superior a um certo valor (para ser considerados de objeto) e cuja vizinhança contém pelo menos um voxel de opacidade que não satisfaz o critério de pertença a objeto. Este método faz parte dos métodos de *volume rendering*, pois renderiza os voxels (volumes elementares) de borda. Porém, ele não é um *rendering* direto, pois existe um passo de extração de bordas. Mas também não é um *rendering* de superfície, pois essa casca, agregação de voxels, tem uma certa espessura. Rocha propôs um método que alia essa estrutura compacta de *shell* com a fatoração SW (Rocha, 2002).

3.3.2 Métodos com precisão de imagem

Enquanto os métodos anteriores mapeavam cada voxel com um ou vários pixels, por projeção, os métodos com precisão de imagem mapeiam os pixels da imagem com os voxels da cena. Esse mapeamento inverso é comumente qualificado de *backward*. Ele consiste em jogar raios a partir de cada pixel do plano da imagem, na direção de visualização (dada pela posição do observador). Esse algoritmo, conhecido como *raycasting*, imita a trajetória de raios luminosos. Ao caminharem, esses raios entram na cena e encontram ou não, voxels pertencentes aos objetos de interesse. Os raios são interrompidos ao saírem da cena.

3.3.2.1 *Raycasting* num volume semitransparente

No caso particular de cena em níveis de cinza, efetua-se a classificação de cada voxel em um dado material em função do seu nível de cinza e de outras medidas mais complexas (gradiente por exemplo). Para isso, definem-se funções de transferência que associam às medidas v de um voxel V, uma cor $f_C(v)$ e uma opacidade $f_\alpha(v)$ que pode também ser vista como o poder de absorção do material ou a taxa de energia luminosa perdida ao atravessar um voxel feito deste material. Essa opacidade é um valor entre 0 e 1. A quantidade $(1 - \alpha)$ corresponde à transparência de um material. Uma opacidade nula (transparência unitária) corresponde a um material totalmente transparente, portanto, invisível. Ele deixa o raio de luz continuar na sua trajetória. Uma opacidade unitária (transparência nula) é característica de um material opaco que absorve totalmente a luz para ser bem visível (por reflexão da luz) e que interrompe o raio luminoso. A cor de um pixel é dada pelas contribuições dos voxels encontrados pelo raio.

Quando se joga um raio de um pixel, a opacidade acumulada na trajetória do raio α_i e a cor visível acumulada C_i são nulas. Ao encontrar um voxel V_i com valor v_i, α_{i-1} e C_{i-1} calculadas no passo anterior são acrescentadas pela contribuição do voxel desta maneira:

$$\alpha_i = \alpha_{i-1} + (1 - \alpha_{i-1}) f_\alpha(v_i)$$

$$C_i = C_{i-1} + (1 - \alpha_{i-1}) f_\alpha(v_i) f_C(v_i)$$

Uma vez a cena atravessada (ou assim que a opacidade acumulada atingir seu valor máximo 1), multiplica-se a cor acumulada C_i pela opacidade acumulada α_i para achar a cor final atribuída ao pixel.

3.3.2.2 *Raycastings* otimizados

Uma otimização freqüente feita no *raycasting* é chamada de *early ray termination*. Quando a energia do raio luminoso foi totalmente absorvida pelos voxels atravessados, este não tem como continuar. Por isso, a propagação do raio é interrompida. No caso de volume binário, assim que o raio bater num voxel de objeto, ele é interrompido, pois o objeto é totalmente opaco (absorção máxima com $\alpha = 1$). Já, no caso de volume em níveis de cinza, as opacidades dos voxels atravessados são compostas e acumuladas. Quando a opacidade acumulada atinge o valor máximo de opacidade (1), o raio pode ser interrompido. Economiza-se o tempo de sua propagação no resto da cena.

Esse método com precisão de imagem tem a vantagem de não criar buracos na imagem. No entanto, sua complexidade é geralmente maior. Na prática, o tempo de execução depende dos casos. Cenas bem cheias são mais rapidamente renderizadas com o *raycasting* do que com projeção de voxels. Os raios não têm que percorrer a cena toda, eles são interrompidos quase logo por objetos. Já, em cenas com poucos voxels de objeto, poucas projeções são necessárias, enquanto muitos raios de um *raycasting* atravessam a cena toda sem ter encontrado nenhum objeto. O tempo de execução do *raycasting* depende muito do tempo durante o qual os raios caminham na cena sem encontrar objetos.

Variações do *raycasting* são possíveis para melhorar a qualidade da imagem ou otimizar o tempo de execução. O raio propaga-se passo a passo na cena. A cada passo, o algoritmo básico de *raycasting* analisa o voxel mais perto da posição do raio na cena. Certos autores examinam as características de vários voxels mais próximos para utilizar uma média ponderada destas. Isto permite a correção de artefatos que podem aparecer ao renderizar objetos muito finos.

Para ganhar em eficiência, pode-se aumentar o passo de andamento dos raios. Os raios podem então atravessar a cena mais rapidamente. Infelizmente, a qualidade da imagem piora. Danskin et al. (Danskin e Hanrahan, 1992) propõem um passo de propagação de raio adaptativo (*adaptive ray sampling*). O passo de propagação do raio diminui quando os dados são de maior interesse (critério relativo à magnitude do gradiente ou à opacidade do material encontrado). Essa mesma idéia também tinha sido aplicada na

imagem (*adaptive screen sampling*) por Levoy (Levoy, 1990). Enviam-se raios de alguns pixels da imagem, espaçados por um certo passo de amostragem. Obtém-se um tipo de pré-visualização (*preview*) de baixa resolução. Interpolam-se então os valores dos pixels intermediários ainda não definidos. Enfim, nos lugares onde o gradiente dos valores calculados é grande, jogam-se raios adicionais para melhorar a precisão da imagem.

Outra estratégia de se ganhar tempo na propagação dos raios é observar que raios paralelos atravessam a cena com a mesma seqüência de passos no volume digital. Essa seqüência pode ser armazenada como um padrão para um raio com uma certa incidência na cena (Yagel e Kaufman, 1992). Isso evita o cálculo das trajetórias e os arredondamentos efetuados a cada passo. Utilizamos uma otimização semelhante, baseada no algoritmo de Bresenham para traçar uma reta num espaço digital (3D no caso). Em cada passo, o erro entre a reta discretizada e a reta real é sempre compensado no próximo passo. Evita-se o cálculo das trajetórias com *floats* e arredondamentos. Um detalhamento do algoritmo de Bresenham em 3D encontra-se no apêndice B.

A introdução dos raios diretamente na cena pode evitar gastar tempo de propagação antes deles entrarem na cena. Generalizando essa idéia, é inútil perder tempo em propagar raios nos lugares vazios da cena. Várias técnicas, chamadas de *space-leaping* (pulo de espaço), ajudam os raios a pular ou atravessar rapidamente os espaços vazios. Avila et al. usam poliedros envolvendo os objetos, tipos de *bounding boxes* (Avila et al., 1992). De uma maneira análoga, pode-se economizar tempo em não jogar os raios que não vão cruzar a cena. Como a cena é convexa (paralelepípedo), pode-se utilizar a coerência da projeção da cena na imagem: Para cada linha da imagem, se pelo menos um raio já atingiu a cena e o raio atual está fora da cena, pode-se pular todos os pixels da linha, pois necessariamente passará fora da cena (Wolff, 2001).

A fatoração *shear-warp*, já apresentada na seção anterior, pode também ser aplicada ao algoritmo de *raycasting*.

Nesses últimos dez anos, o uso de textura 3D tem se desenvolvido bastante. As texturas 3D, também chamadas de texturas de volume, requerem o auxílio de hardware especializado (Tan, 2001). Geralmente, elas são usadas em programas que processam imagens 3D obtidas por fatias, como as IRM. Cabral et al. fizeram um *rendering* utilizando fatias de textura 3D (Cabral et al., 1994). Depois de ter armazenado a cena inteira na memória de textura 3D, corta-se a cena em fatias paralelas ao plano da imagem. As fatias de textura são compostas numa ordem BTF (de trás para frente) para obter a imagem final. A grande desvantagem das texturas 3D é delas usarem muita memória. Por exemplo, uma imagem de textura de $256 \times 256 \times 256$ texels renderizada com *double buffer* numa janela de 640×480 pixels com 32 bits de profundidade, mais um *z-buffer* gastam aproximadamente 20 Mbytes. Por isso, é imprescendível o uso de hardware especializado. Hoje em dia, várias placas de vídeo disponíveis no mercado possibilitam esse *rendering* via textura 3D. Alguns utilizadores falam desse método como muito eficiente no caso de renderização de volume de RM (Tan, 2001; Beets, 2000; Lima, 1999).

3.3.3 Métodos no domínio da freqüência

O *rendering* de freqüência é uma alternativa aos dois tipos de *renderings* apresentados. Ele é baseado numa mudança de espaço. Os dados do domínio espacial (3D) são transformados num domínio alternativo onde a projeção é diretamente gerada. Este domínio alternativo pode ser o das freqüências (transformada de Fourier), o da transformada em *wavelets* (Westenberg e Roerdink, 2000), o da DCT (transformada de cosseno discreta) entre outros.

O *rendering* de freqüência produz uma radiografia digital reconstruída (*Digital Reconstructed Radiography*, DRR), isto é, uma imagem obtida por projeção numa cena 3D adquirida por CT. Os métodos baseiam-se no modelo ótico de absorção e dão uma imagem de tipo radiográfico, não desejável pelos métodos apresentados nas seções anteriores que são aplicáveis a qualquer tipo de imagem.

Para obter a DRR, o método de *rendering* de Fourier (*Fourier Volume Rendering*, FVR) é menos complexo computacionalmente ($O(N^2 logN)$) que os métodos de *raycasting*, *splatting* ou *shear-warp* já apresentados ($O(N^3)$) (Ntasis et al., 2002). Ele é facilmente paralelizável e de boa qualidade de imagem. É baseado no teorema de plano-projeção de Fourier (Totsuka e Levoy, 1993). Segundo este, a imagem 2D obtida, pegando-se as integrais do volume ao longo dos raios perpendiculares à imagem, tem como transformada de Fourier, o espectro 2D obtido extraindo-se uma fatia da transformada de Fourier do volume, que passa pela origem e é paralela ao plano da imagem. Portanto, a estratégia do FVR é:

- transformar a cena (domínio espacial) em um espectro 3D (domínio das freqüências) via transformada 3D de Fourier rápida (*Fast Fourier Transform*, FFT);
- extrair a fatia paralela ao plano da imagem e passando pela origem do espectro 3D;
- aplicar a transformada inversa de Fourier (IFFT) 2D nessa fatia de espectro para obter a imagem da projeção esperada.

3.3.4 Observações

Depois dessa rápida introdução sobre o *rendering* freqüencial, apresentam-se algumas observações e comparações entre os métodos principais.

Primeiro, há de se notar a importância da segmentação ou de uma identificação qualquer (classificação por exemplo) dos objetos de interesse, antes de visualizar uma cena. Se não tiver nenhuma identificação no sentido largo (sgmentação ou classificação), a imagem obtida é do tipo radiográfico (como no FVR): Os objetos misturam-se, além de se adicionarem ruídos. A impressão de terceira dimensão é perdida.

Segundo, a segmentação difere bastante da classificação, pois particiona realmente cada voxel da cena num objeto ou no fundo. Neste sentido, ela é mais poderosa por permitir uma análise quantitativa dos objetos da cena. Após segmentação interativa, o usuário pode visualizar o resultado e quantificá-lo:

fazer diversas medidas relacionadas às estruturas segmentadas visíveis. Os métodos de *rendering* de volume em tons de cinza, que envolvem a classificação em materiais semitransparentes, não possibilitam a análise quantitativa dos objetos de interesse (medidas de volume, superfície, baricentro, etc.). Não tem, nesse caso, descrição volumétrica das estruturas de interesse. No entanto, esses métodos são adequados para a visualização de estruturas difusas ou nebulosas (nuvens, tumores difusas). No caso de estruturas discretas, prefere-se segmentá-las ou extraí-las previamente. Muitos autores têm tentado refinar as funções de transferência utilizadas na classificação, levando a funções multidimensionais bastante complexas ou ajustadas pelo usuário (Kniss et al., 2002; Kindlmann e Durkin, 1998). Lürig et al. utilizam ferramentas de morfologia matemática para analisar hierarquicamente o volume e, dessa maneira, estimar as propriedades óticas do volume a ser renderizado (Lürig e Ertl, 1998). É um eixo de pesquisa bastante ativo que é preocupado, antes da eficiência, pela qualidade do *rendering*, pela estética da imagem, sem interessar-se à quantificação das estruturas visualizadas.

Existe uma discussão sobre a escolha entre os *renderings* de volume ou de superfície. Essa escolha depende muito do tipo de aplicação desejada e do tipo de dados processados. O *rendering* de superfície tem, por exemplo, a vantagem de possibilitar manipulações interativas orientadas a estrutura (Udupa e Herman, 1991). A representação e visualização volumétricas (*volume rendering*) são mais apropriadas para dados amostrados ou computados, enquanto que, para os dados modelados que aparecem nas aplicações tradicionais de computação gráfica (modelagem e geração de cenas sintéticas), a visualização de superfície (*surface rendering*) parece mais adequada ou natural. No entanto, não se consegue representar com superfícies fenômenos amorfos tais como o fogo, as nuvens, a fumaça, etc. Segundo vários autores (Kaufman, 1996), a tendência é de se visualizar mais com *rendering* de volume, baseado nos voxels. A emergência da área chamada *Volume Graphics* (Manohar, 1999) apoia este ponto de vista.

Esta abordagem constitui uma alternativa à tradicional *Surface Graphics*. Efetua-se uma "voxelização" (ou *3D scan-conversion*) que melhor aproxima um modelo geométrico com um conjunto de voxels. Esses voxels são armazenados juntos com seus atributos independentes do ponto de vista. *Volume Graphics* apresenta várias vantagens sobre *Surface Graphics*: A visualização é independente da complexidade dos objetos e da cena; é possível efetuar operações por blocos e utilizar representações hierárquicas; pode-se facilmente misturar dados amostrados ou computados com objetos geométricos (útil nas imagens médicas onde aparecem órgãos e objetos sintéticos como próteses); estruturas internas são visualizadas; fenômenos amorfos também. Existem, porém, as seguintes desvantagens: um custo maior em memória, um tempo de processamento maior, o efeito de *aliasing* e a perda de informação geométrica. *Volume Graphics*, geralmente responsáveis de síntese, manipulação e *rendering* de objetos volumétricos modelados geometricamente, diferencia-se da área de *Volume Visualization* aplicada a dados amostrados ou computados. Mas as duas abordagens vão se aproximando e se complementando.

3.4 Tonalização

A tonalização é uma das principais técnicas utilizadas para dar de uma imagem 2D a ilusão de 3D. Esta seção relembra os três tipos de tonalização mais populares hoje em dia.

No caso de cenas previamente segmentadas, os métodos de visualização apresentados neste capítulo resultam na criação de um *z-buffer* (ou *d-buffer*), ou seja, uma imagem contendo, para cada pixel, a distância em que se encontra o objeto de interesse. A tonalização atribui então um valor de cinza (tipicamente entre 0 e 255 para display com 8 bits de profundidade) levando em conta tanto a distância dos objetos como as condições de luz presentes. Dependendo dos métodos de tonalização, a imagem resultante dá uma impressão mais ou menos realista da cena. A tonalização também é responsável das sombras e de certos efeitos de material (brilho metálico por exemplo). No caso de imagens médicas, não é desejável ter efeitos de sombras nas estruturas examinadas, por isso, a posição da fonte luminosa é confundida com a posição do observador.

3.4.1 *Depth-Shading*

Esta tonalização depende apenas dos valores do *d-buffer*, isto é, das distâncias do plano de visualização aos objetos. Considera-se que os objetos mais próximos são mais luminosos, enquanto os mais distantes aparecem mais escuros. A relação entre a distância e o nível de cinza associado é linear. O pixel de coordenadas (u, v) recebe o nível de cinza:

$$I_{dist}(u,v) = round\left(\frac{I_{max} - I_{min}}{d_{max} - d_{min}}[d_{max} - d(u,v)] + I_{min}\right)$$

onde

I_{max} é o nível de cinza máximo desejado na imagem (geralmente 255 para um display de 8 bits);
I_{min} é o nível de cinza mínimo desejado na imagem (geralmente 0, para ter o maior contraste possível);
d_{max} é a maior distância contida no *d-buffer*;
d_{min} é a menor distância contida no *d-buffer*;
$d(u, v)$ é a distância contida no *d-buffer* para o pixel (u, v).

Esse modelo de tonalização é muito simples e não dá imagens de muito boa qualidade, pois não muito realistas.

3.4.2 *Phong-Shading*

Na verdade, a tonalização proposta por Phong (*Phong shading*) aplica-se no caso de polígonos (*mesh*) e consiste em interpolar os vetores normais à superfície dos objetos. Estes vetores servem a calcular o valor de tonalização segundo a equação de iluminação dada pelo modelo de iluminação de Phong. Mas, muitas vezes, usa-se o termo tonalização de Phong (*Phong shading*) no lugar de modelo de iluminação de

Phong. Esse modelo (Phong, 1975) não se aplica apenas nos polígonos, mas é útil para tonalizar qualquer tipo de cena (em *rendering* direto ou não). Esse modelo, não baseado em modelo físico, mas sim, empírico, proporciona uma tonalização que é considerada, hoje em dia, de boa qualidade realista. O modelo leva em conta as influências da distância, das luzes ambiente, difusa e especular. A intensidade I de um pixel (u, v) é dada pela equação seguinte:

$$I(u, v) = K_a I_{max} + I_{dist}(u, v)[K_d(\vec{N} \cdot \vec{L}) + K_s(\vec{R} \cdot \vec{V})^n]$$

ou seja,

$$I(u, v) = K_a I_{max} + I_{dist}(u, v)[K_d \cos\theta + K_s \cos^n \alpha]$$

onde

K_a é o coeficiente de reflexão para a luz ambiente,
K_d é o coeficiente de reflexão difusa,
K_s é o coeficiente de reflexão especular,
n é o expoente de reflexão especular do material (responsável do aspecto metálico),
$I_{dist}(u, v)$ é a intensidade dada pelo *depth-shading*,
$\vec{N}$ é a normal à superfície do objeto no ponto que foi projetado no pixel (u, v),
$\vec{L}$ é o vetor unitário que aponta a direção da luz, do ponto incidente na superfície do objeto,
$\vec{R}$ é o vetor unitário de reflexão do raio luminoso, é simétrico de $\vec{L}$ em relação ao vetor $\vec{N}$,
$\vec{V}$ é o vetor unitário que sai do ponto incidente na superfície do objeto e aponta na direção do observador,
θ é o ângulo entre o vetor $\vec{N}$ normal à superfície e o raio de incidência $\vec{L}$,
α é o ângulo entre o vetor de reflexão $\vec{R}$ e o vetor do observador $\vec{V}$.

Obs.: Devem ser considerados apenas os valores de θ entre 0 e 90 graus para o componente difuso e valores de α entre 0 e 90 graus para o componente especular.

Para facilitar a implementação, $\vec{R}$ pode ser determinado a partir de $\vec{N}$ e $\vec{L}$:

$$\vec{R} = 2\vec{N}(\vec{N} \cdot \vec{L}) - \vec{L}$$

Logo, a equação de iluminação torna-se:

$$I(u, v) = K_a I_{max} + I_{dist}(u, v)\{K_d(\vec{N} \cdot \vec{L}) + K_s[(2\vec{N}(\vec{N} \cdot \vec{L}) - \vec{L}) \cdot \vec{V}]^n\}$$

Na visualização de imagens médicas, usa-se geralmente uma iluminação na mesma direção que o observador, isto é: $\vec{V} = \vec{L} \Leftrightarrow \alpha = 2\theta$. Essas condições de iluminação impedem a formação de sombras indesejáveis em visualização para diagnósticos. A equação do modelo torna-se:

$$I(u, v) = I_{max} K_a + I_{dist}(u, v)[K_d \cos\theta + K_s \cos^n 2\theta]$$

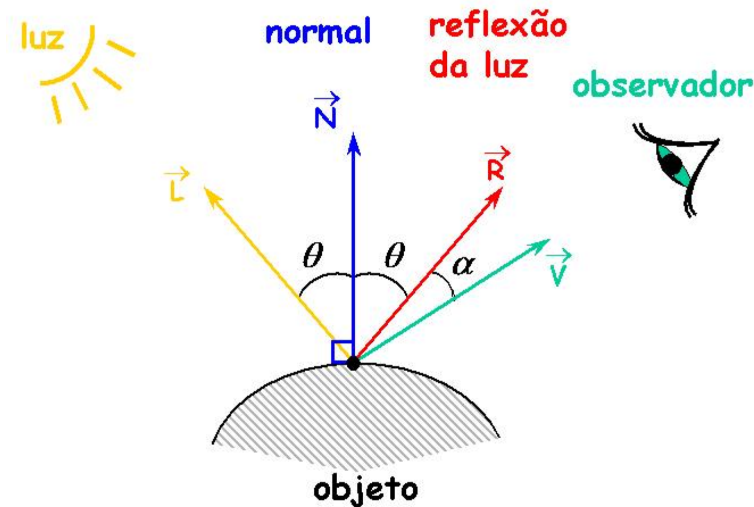


Figura 3.3: Modelo de iluminação de Phong

Nesse modelo de iluminação, a avaliação da normal ao objeto é essencial. Dela depende o aspecto final da tonalização. Várias estimativas desta normal foram implementadas neste trabalho: a partir do gradiente dos níveis de cinza dos voxels da cena; a partir do gradiente nos pixels da imagem; a partir da geometria dos objetos. Vide apêndice C para o desenvolvimento matemático dessas estimativas. Cada estimativa é mais apropriada para uma certa modalidade de imagem (CT, IRM, etc.).

3.4.3 Gouraud-*Shading* ou *Smooth Shading*

Essa tonalização aplica-se apenas em polígonos (Gouraud, 1971). Ela é baseada na interpolação linear dos valores de tonalização calculados apenas para os vértices dos polígonos. Considera-se que o polígono é plano (sempre válido no caso de triângulos). Isto evita calcular o valor de tonalização de cada pixel usando a equação de iluminação. A interpolação linear fica menos custosa.

Essa tonalização é usada implicitamente por OpenGL para o *rendering* de primitivas geométricas tais os triângulos de um *mesh*. O cálculo da normal a um triângulo é trivial

Capítulo 4

Segmentação e visualização interativas

Agora que foram introduzidos alguns conceitos em visualização tridimensional, assim como o método de segmentação iterativa escolhido, serão apresentadas as estratégias propostas de segmentação com visualização.

4.1 Estratégias de segmentação e visualização interativas

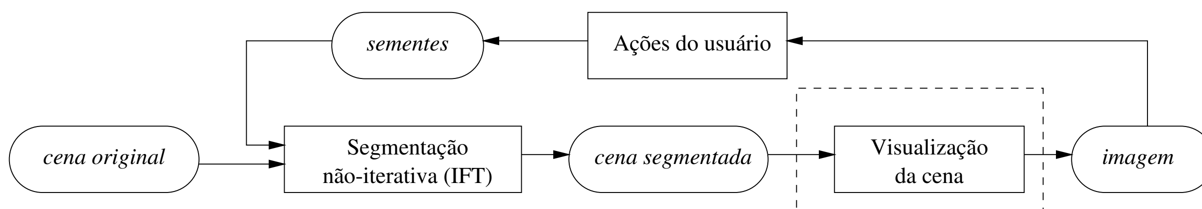


Figura 4.1: Segmentação não incremental com visualização de cena

O método mais simples e mais clássico é usar o algoritmo de segmentação (*watershed* via IFT) seguido de um algoritmo de visualização da cena (*splatting* da cena inteira por exemplo). Os processos de segmentação e visualização formam dois blocos totalmente independentes (vide a figura 4.1). Na prática, o usuário escolhe sementes a partir das vistas 2D (fatias) da cena, nos lugares que lhe parecem representar objetos de interesse diferentes ou fundo. Ele não precisa delinear os objetos, mas escolher pelo menos uma semente por objeto e uma para o fundo. A partir dessas sementes, a IFT segmenta a cena. Um algoritmo de *rendering* traduz a cena segmentada numa imagem, visualizada na tela e dando um efeito de 3D. Se o usuário julgar que o resultado não é o desejado, ele reitera esse processo: Ele acrescenta um conjunto de sementes (que pode aproveitar-se do conjunto anterior ou não); a IFT processa de novo a cena inteira com esse novo conjunto de sementes, independentemente do resultado da segmentação anterior; e uma nova imagem é criada a partir do novo resultado da segmentação. O processo é reiterado até o usuário ficar satisfeito com o resultado da segmentação. Esse método trivial tem a desvantagem de ser demorado por reprocessar em cada passo da segmentação a cena inteira.

A segunda estratégia, usada por Bergo e Falcão (Bergo e Falcão, 2003), consiste em usar o algoritmo de segmentação iterativa (*watershed* iterativo) e, no final de cada passo, aplicar também um método de visualização direta sobre os voxels da cena segmentada. Como no caso anterior, podem ser projetados todos os voxels (*voxel splatting*) da cena para obter a imagem, ou ainda, podem ser jogados raios (*raycasting*) a partir da imagem para encontrar os voxels da cena segmentada, cada voxel tendo um rótulo que identifica o objeto no qual ele foi particionado. Nesse caso, segmentação e visualização formam dois blocos independentes também, mas a segmentação é incremental, ela aproveita o resultado anterior (ver o diagrama da figura 4.2). O usuário escolhe sementes no primeiro passo para designar objetos e observa a imagem do resultado da segmentação. Caso não fique satisfeito, ele acrescenta novas sementes e também eliminadores para remover regiões mal segmentadas. A DIFT processa apenas parte da cena: a parte que vai ser modificada (reconquistada). Por isso, os passos posteriores ao primeiro são bem mais rápidos. Essa estratégia tem um ganho no processo de segmentação, mas não na visualização. Seguem duas propostas novas para tentar aproveitar os passos anteriores no processo de visualização: o algoritmo de "borda incremental" e o de "imagem incremental".

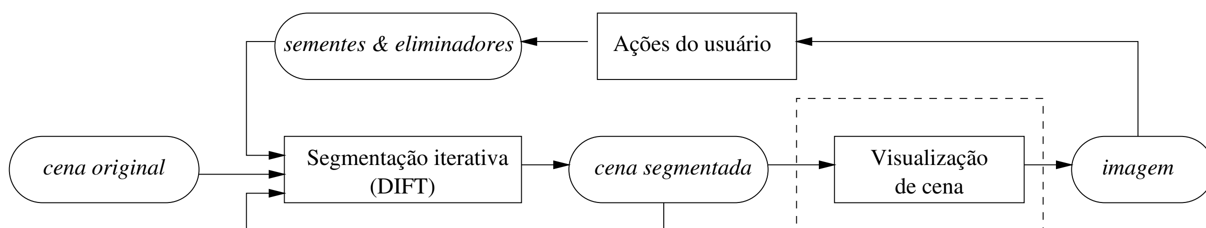


Figura 4.2: Segmentação iterativa com visualização de cena

A primeira proposta posiciona-se do ponto de vista da cena. Ela tenta otimizar o método de projeção de voxels. Com efeito, ela consiste em diminuir o número de voxels a serem projetados, esperando, desse jeito, acelerar a visualização. No fundo, parte-se da questão seguinte: Quais são os voxels realmente visíveis na imagem? Podemos constatar que, no caso de uma visualização de tipo opaca, apenas os voxels de objeto que são vizinhos de voxels de fundo são visíveis. Chamamos esses voxels de *voxels de borda*. Portanto, qualquer que seja o ângulo de visualização do observador, é preciso projetar apenas a borda dos objetos (ver o diagrama da figura 4.3). Economiza-se, desta maneira, a projeção dos voxels internos aos objetos (sempre invisíveis, pois escondidos pelos de borda), e, obviamente, não se projetam os voxels de fundo. Tipicamente, menos de 3% dos voxels das cenas tratadas são de borda. Por outro lado, para não perder tempo em detectar a borda dos objetos, ela é construída de maneira incremental. Com efeito, a detecção e atualização da borda é feita durante o processo de segmentação para ganhar tempo. Os processos de segmentação e visualização já não são mais totalmente independentes, pois a atualização da borda é embutida no algoritmo da "BDIFT" (DIFT com detecção de Borda). A dificuldade desse método é manter atualizada, em cada passo da segmentação, a borda dos objetos: uma lista de voxels de borda a projetar, como explicamos mais detalhadamente na seção 4.2. Quanto ao usuário, ele age da mesma maneira que na estratégia precedente: Escolhe sementes e eliminadores para a segmentação e reitera o processo até ficar satisfeito.

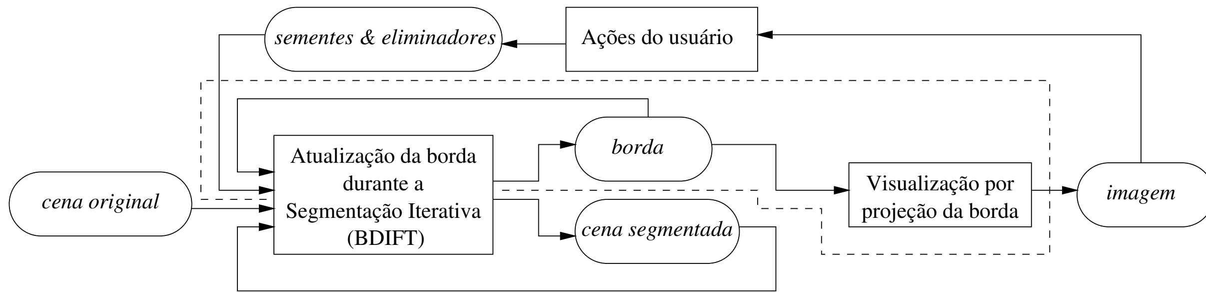


Figura 4.3: Segmentação iterativa com visualização de borda

A segunda proposta, apresentada com detalhes na seção 4.3, é baseada no ponto de vista da imagem e não da cena. Interessa-se nas mudanças que podem ocorrer na imagem, devido à atualização da segmentação. Supõe-se que o ângulo de visualização permanece o mesmo. A imagem pode variar de um passo da segmentação para outro, por causa da mudança de rótulos atribuídos a certos voxels. Alguns voxels, que pertenciam ao fundo no passo anterior de segmentação, podem ter se tornado objeto e, portanto, aparecer na imagem. Outros, pelo contrário, podem desaparecer por serem escondidos neste passo ou por terem se tornado fundo. Ainda outros podem simplesmente ter trocado de rótulo, isto é, de objeto, e permanecer visíveis. Como geralmente se indicam objetos diferentes com (falsas) cores diferentes, a imagem também deve levar em conta a alteração da cor. Com essas considerações, tentou-se desenvolver um método baseado na atualização da imagem que utiliza os dois tipos de técnica de visualização direta: a projeção dos voxels com rótulo alterado seguida de *raycastings* para atualizar o conteúdo dos pixels atingidos. Esse método poderia ser chamado de "*display* incremental" ou "imagem incremental", do mesmo modo que a DIFT é uma IFT incremental, pois cada passo é baseado e aproveita-se do anterior (ver o diagrama da figura 4.4). É um método construtivo.

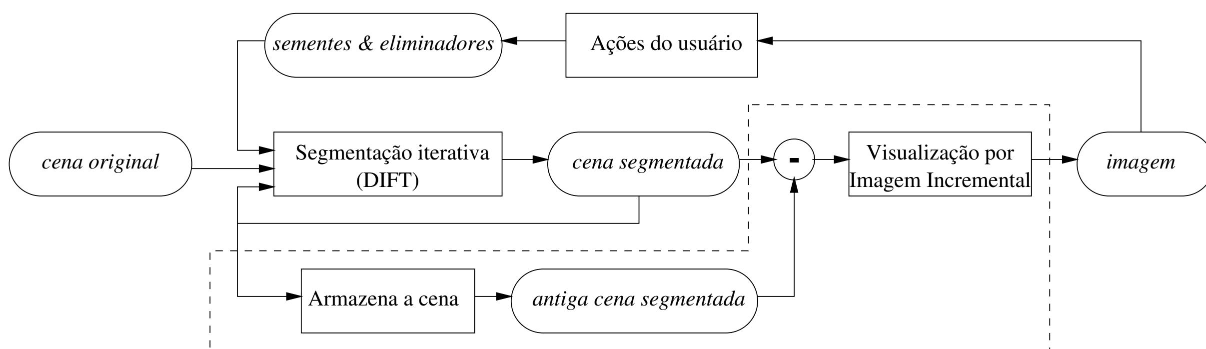


Figura 4.4: Segmentação iterativa com visualização incremental

4.2 Borda incremental

Como foi dito na introdução, o método de borda incremental consiste em armazenar os voxels que pertencem à borda dos objetos e atualizar essa lista de voxels em cada passo da segmentação. Essa lista de voxels é projetada por *splatting* para se obter a imagem desejada. O algoritmo responsável da projeção é chamado "*SplatBorder*" e funciona exatamente como o algoritmo clássico de *splatting* descrito no capítulo anterior (ver o pseudocódigo da seção 3.3.1), exceto que se passa, como entrada, uma lista de voxels rotulados em vez da cena inteira. É um tipo de algoritmo de visualização por *shell* (Udupa e Odhner, 1993). Ele oferece as seguintes vantagens: Usa-se apenas a lista de voxels de borda para criar a imagem e essa lista, geralmente reduzida, é independente do ponto de observação.

Para atualizar a borda dos objetos de um passo para outro, existem várias estratégias. A mais trivial, de força bruta, seria varrer toda a cena segmentada no final de cada passo e verificar para todos os voxels rotulados como objetos (rótulo não nulo convencionalmente) os rótulos dos voxels vizinhos. Se pelo menos um vizinho tiver um rótulo de fundo (rótulo nulo em geral), o voxel é inserido na lista dos voxels de borda. Esse método tem a vantagem de ser independente do algoritmo de segmentação usado. Porém, ele tem uma complexidade de $O(n \cdot v)$, onde n é o número de voxels da cena e v o número de vizinhos que pode ter um voxel. Tem que adicionar o tempo deste processamento ao tempo de projeção das bordas. Dependendo da cena, o tempo ganho em projetar apenas os voxels de borda, se for comparado ao tempo de projetar todos os voxels, não compensa o tempo adicional gasto em encontrar os voxels de borda.

O algoritmo proposto, responsável pela re-segmentação, batizado "*BorderDIFT*", incorpora a atualização da lista de voxels de borda ao processo de segmentação via DIFT. Com efeito, este método obriga a embutir a extração ou atualização da borda dos objetos no método de segmentação. Por isso, o algoritmo de segmentação, direcionado para a visualização, torna-se mais complexo.

4.2.1 Extração e atualização da borda

Encontrar os voxels de borda equivale a encontrar os voxels de objeto que possuem pelo menos um vizinho de fundo. O objetivo deste algoritmo é encontrar os voxels de borda, à medida que eles são processados pela DIFT. No algoritmo da DIFT, o laço principal trata o esvaziamento da fila de prioridades. Saem da fila, por ordem de prioridade, todos os voxels que entraram na fila, isto é:

- os voxels sementes,
- os voxels fronteiras (pertencentes a árvores não marcadas para remoção, mas vizinhos das árvores a serem removidas),
- os voxels dominados por um novo rótulo que se propaga, conseqüência do baixo custo proposto pelos voxels vizinhos que saem da fila,

- os voxels filhos dos voxels saindo da fila (condição para a validade do algoritmo incremental da DIFT).

Resumindo, passam pela fila os voxels das frentes de propagação (especificados pelo usuário) e suas descendências (a do passo anterior e a novamente conquistada).

Há várias situações em que a lista de voxels de borda precisa ser atualizada, seja acrescentando elementos, seja removendo.

Situação 1: Adição de elemento de borda. Quando um voxel sai da fila, ele adquiriu um novo rótulo, foi ***reconquistado***, portanto, tem-se de classificá-lo como voxel de borda ou não. Neste momento, é importante examinar os rótulos dos voxels vizinhos quando estes são permanentes. Se os rótulos dos vizinhos não são permanentes, não se pode concluir sobre a natureza do voxel. No entanto, eles podem adquirir um novo rótulo mais tarde. Quando um desses vizinhos sair da fila com um rótulo permanente, ele examinará aquele voxel que deixou de ser classificado. Em função dos rótulos dos dois, concluir-se-á sobre a natureza dos dois voxels (borda ou não).

Situação 2: Remoção de elemento de borda. Analisando passos posteriores, pode acontecer que o usuário marque um voxel para remover a árvore de voxels à qual ele pertence. Os atributos desses voxels (custo, rótulo/raiz, predecessor) são inicializados, pois os voxels vão ser conquistados por outras árvores que se propagam. Se por ventura, um dos voxels ***reinicializados*** pertencia à borda, ele tem que ser removido da lista dos voxels de borda, pois seu rótulo não é mais determinado. Ele será determinado novamente pelas propagações futuras das sementes e voxels fronteiras.

Situação 3: Remoção de elemento de borda. Por outro lado, os voxels de fronteira que pertenciam à borda não podem ser considerados ainda de borda, pois alguns de seus vizinhos foram removidos e não têm mais rótulo determinado. Então, é preciso tirar da borda os voxels de ***fronteira***.

Situação 4: Remoção de elemento de borda. Da mesma maneira, os voxels sementes acabaram de ser inseridos pelo usuário, com novos custo e rótulo. Portanto, em cada passo da DIFT, as ***sementes*** têm que ser removidas da borda.

Situação 5: Remoção de elemento de borda. Todos os voxels que entram na fila de prioridade são ***reconquistáveis*** por um vizinho oferecendo custo menor, e podem mudar de rótulo. Por isso, nada se garante sobre sua condição de voxel de borda: Eles têm que ser removidos da borda para ser avaliados de novo.

Nessas quatro últimas situações, os voxels deixam de pertencer à borda porque seu rótulo ou o rótulo de um vizinho é ou pode ser modificado. Pode acontecer, no entanto, que, por exemplo, esses voxels ou seus vizinhos assumam os mesmos rótulos que antes, após o processamento da DIFT. Portanto, esses voxels removidos da borda podem ser de novo voxels de borda. Nas quatro situações, existe a possibilidade desses voxels serem inseridos de novo na borda: Voxels fronteiras, sementes e reconquistados (situações 3, 4 e 5)

são inseridos na fila e, portanto, serão testados ao saírem dela para saber se pertencem à borda (situação 1). No caso dos voxels das árvores removidas (situação 2), o custo deles foi reinicializado com um custo infinito. Como o custo proposto pelos seus vizinhos será finito (custo menor que infinito), eles serão necessariamente reconquistados e entrarão na fila de prioridades. Ao entrarem, eles serão sistematicamente removidos da borda (situação 5). Finalmente, verificar-se-á se eles são de borda ao saírem da fila (situação 1). A situação 2 é um caso particular da situação 5, portanto, não necessita um tratamento particular.

4.2.2 Algoritmo

Como no algoritmo da DIFT, após inicialização da tabela de estado dos voxels, começa-se a tratar as árvores a serem removidas, selecionadas pelos eliminadores (ver o pseudocódigo E). Essa tarefa é efetuada pela função "*DIFT_TreeRemoval*", apresentada anteriormente (ver o pseudocódigo C).

A partir dos conjuntos de eliminadores E e de sementes S, forma-se o conjunto F dos voxels sementes e voxels fronteiras. Todos os voxels pertencentes às árvores designadas por um eliminador são reinicializados com custo infinito e sem nenhum predecessor, de maneira a poder ser reconquistados por outra árvore que se propaga. Encontram-se os voxels de fronteira que são vizinhos dos voxels reinicializados, porém, não pertencentes a árvores marcadas. Eles são, como as sementes, pontos de partida para uma nova propagação de rótulos e, portanto, são inseridos no conjunto F, juntos a elas.

Continuando na função principal "*BorderDIFT*" (vide o pseudocódigo E), inserem-se os voxels sementes e os voxels fronteiras na fila de prioridade (l. 4). Mas para não deixar erradamente um voxel semente ou fronteira na borda, remove-se sistematicamente da borda B o elemento que entra na fila de prioridade (l. 5). Corresponde à remoção dos voxels das situações 3 e 4. Em seguida, entra-se no laço central da DIFT. Enquanto a fila de prioridade não estiver vazia, remove-se o elemento de mais alta prioridade, isto é, de custo mínimo (l. 6 e 7).

O voxel saindo da fila de prioridade tenta, primeiro, propagar seu rótulo nos seus vizinhos (l. 9–16). Essa propagação é possível, somente se o vizinho ainda não recebeu rótulo permanente, ou seja, seu estado não é definitivo ($e(t) \neq 2$) e seu custo corrente é maior do que o proposto. Com estas condições os voxels vizinhos são reconquistados (l. 12–15). E conforme à situação 5, removem-se estes voxels da borda (l. 16).

No caso do vizinho já ter rótulo permanente, determina-se se é possível inserir o voxel ou seu vizinho na borda (l. 18–20). Essa parte é específica do algoritmo de borda incremental. Examinam-se os rótulos do voxel saindo da fila e do voxel vizinho definitivamente rotulado. Se um é de objeto (não nulo), enquanto o outro é de fundo (nulo), o voxel de objeto é inserido na borda. Assim é tratada a adição dos voxels da situação 1.

Existe finalmente o tratamento de casos pendentes nas linhas 21 a 25. Estes casos foram sinalizados ou atualizados durante o processo de propagação (nas linhas 8 e 17 do algoritmo) e só podem ser tratados no final da propagação. Esse apêndice é necessário para a corretude do algoritmo. Com efeito, imaginemos

a situação em que um voxel s não pode ser classificado como borda ou não-borda porque seus vizinhos t não têm um rótulo definitivo ($e(t) \neq 2$), nem no momento em que ele sai da fila de prioridade, nem mais tarde, e isso até o final da propagação, pois eles nunca entrarão na fila de prioridade durante este passo de *BorderDIFT*: Eles são voxels "*resistentes*" a qualquer reconquista porque o custo atribuído em passos anteriores da *BorderBDIFT* é menor ou igual aos custos propostos no passo atual. É preciso notificar este caso pendente e resolvê-lo depois da propagação.

Resumindo: Os casos pendentes ocorrem então no limite das frentes de reconquista, quando um voxel, saindo da fila, não consegue se propagar e reconquistar seu vizinho, um resistente. Os voxels resistentes permanecem no estado 'nunca esteve na fila' ($e(t) = 0$) até o final do passo da *BorderBDIFT*, porém, são vizinhos de voxels reconquistados que não puderam ser classificados em relação à borda.

Logo, quando um voxel não se propaga para o vizinho e que este vizinho nunca esteve na fila, ele é incluído no conjunto dos resistentes (l. 17), porque é vizinho de um caso pendente. Por outro lado, se, ao decorrer da propagação, um voxel resistente acaba sendo conquistado, ele entra na fila e ao sair dela, ele é sistematicamente removido do conjunto N dos resistentes (l. 8). Desta maneira, N contém apenas os voxels que realmente resistiram à reconquista durante todo o período da propagação. Após a propagação, testam-se, um por um (l. 21-25), os voxels resistentes, que são potencialmente de borda (l. 24) ou podem ter vizinhos de borda (l. 25). Atualiza-se então a borda.

Pseudocódigo E ***BorderDIFT***

Variáveis de entrada:

* *a cena I (conjunto de voxels),*
* *o mapa de custos C (custos acumulados dos caminhos até cada voxel, em um dado instante),*
* *o mapa de raízes R,*
* *o mapa de predecessores P.*
* *a vizinhança simétrica V: $V(p)$ designa o conjunto dos vizinhos do elemento p,*
* *o conjunto S de voxels sementes,*
* *o conjunto E de voxels eliminadores (para indicar as partes a serem removidas),*
* *o conjunto B de voxels de borda,*
* *a função de rotulação λ que associa um rótulo $\lambda(s)$ a cada semente s de S,*
* *a função suave de custo de caminho f que associa ao caminho π_p o custo de percurso $f(\pi_p)$.*

Variáveis de saída:

* *o mapa de custos C,*
* *o mapa de raízes R,*
* *o mapa de predecessores P e*
* *o conjunto de voxels de borda B.*

Variáveis auxiliares:

* *a fila de prioridade Q vazia, gerenciada com as seguintes funções:*
    * *$RemoveMin(Q)$ remove da fila Q, o elemento de custo mínimo.*
    * *$Remove(p, Q)$ remove o elemento p da fila Q.*

* $Insere(p, Q, c)$ *insere o elemento* p *na fila* Q *com o custo* c.
* $EstáVazia(Q)$ *retorna um booleano indicando se a fila* Q *está vazia;*

* *a tabela* e *de estado dos voxels associando a cada voxel* p *da imagem* I *o valor* $e(p)$ *igual a:*
  * *0 se o voxel* p *não foi tratado e não está na fila* Q,
  * *1 se* p *está em* Q *ou*
  * *2 se* p *já foi tratado definitivamente (portanto, está fora da fila);*
* *a variável* c *para armazenar um custo (geralmente inteiro);*
* *o conjunto* F *vazio (para conter voxels sementes ou de fronteira) e*
* *o conjunto* N *vazio (para conter voxels Não reconquistados mas vizinhos de voxels reconquistados: "os resistentes").*

1. $\forall t \in I, \quad e(t) = 0$;
2. $(C, P, F) \leftarrow DIFT_TreeRemoval(C, R, P, V, S, E)$;
3. $\forall t \in F$,
4. $\quad Insere(t, Q, C[t]); \quad e(t) \leftarrow 1$;
5. $\quad$ **Se** $t \in B, \quad B \leftarrow B/\{t\}$;

6. **Enquanto** $EstáVazia(Q) = FALSO$,
7. $\quad s \leftarrow RemoveMin(Q); \qquad e(s) \leftarrow 2$;
8. $\quad$ ***Se*** $s \in N, \quad N \leftarrow N/\{s\}$;
9. $\quad \forall t \in V(s)$,
10. $\qquad$ ***Se*** $e(t) \neq 2$,
11. $\qquad\quad c \leftarrow f(\pi_s \cdot \langle s, t \rangle)$;
12. $\qquad\quad$ **Se** $c < C[t]$ **ou** $P[t] = s$,
13. $\qquad\qquad$ **Se** $e(t) = 1, \quad Remove(t, Q)$;
14. $\qquad\qquad Insere(t, Q, c); \qquad e(t) \leftarrow 1$;
15. $\qquad\qquad C[t] \leftarrow c; \qquad R[t] \leftarrow R[s]; \qquad P[t] \leftarrow s$;
16. $\qquad\qquad$ **Se** $t \in B, \qquad B \leftarrow B/\{t\}$;
17. $\qquad\quad$ **Senão**, **Se** $e(t) = 0, \quad N \leftarrow N \cup \{t\}$;
18. $\qquad$ **Senão**,
19. $\qquad\quad$ **Se** $\lambda(R[s]) \neq 0$ **e** $\lambda(R[t]) = 0, \quad B \leftarrow B \cup \{s\}$;
20. $\qquad\quad$ **Senão**, **Se** $\lambda(R[s]) = 0$ **e** $\lambda(R[t]) \neq 0, \quad B \leftarrow B \cup \{t\}$;

21. $\forall s \in N$,
22. $\quad$ **Se** $s \in B, \quad B \leftarrow B/\{s\}$;
23. $\quad \forall t \in V(s)$,
24. $\qquad$ **Se** $\lambda(R[s]) \neq 0$ **e** $\lambda(R[t]) = 0, \quad B \leftarrow B \cup \{s\}$;
25. $\qquad$ **Senão**, **Se** $\lambda(R[s]) = 0$ **e** $\lambda(R[t]) \neq 0, \quad B \leftarrow B \cup \{t\}$;

Obs.: A função λ associa a cada voxel semente um rótulo. Esse rótulo é um inteiro associado a um objeto particular. O rótulo nulo é reservado ao fundo: $\lambda(R[s]) \neq 0 \Leftrightarrow s \in Objeto$ e $\lambda(R[s]) = 0 \Leftrightarrow s \in Fundo$. A função é inicializada (antes da primeira chamada do algoritmo) com o valor do rótulo de semente para cada voxel semente.

4.3 Imagem incremental

Como apresentado na introdução deste capítulo, o método de "imagem incremental" é baseado na atualização da imagem. Esta atualização é guiada pela observação dos voxels que mudaram de rótulo de um passo da segmentação para outro. Com efeito, fazem-se, na imagem, apenas as modificações correspondentes a esses voxels. Analisemos, primeiro, quais são os casos possíveis de modificação da imagem. Em seguida, detalharemos o algoritmo proposto.

4.3.1 Análise das mudanças

Parte-se da hipótese que o ângulo de visualização é fixo. Logo, qualquer mudança nos pixels da imagem só pode ser causada por uma mudança na segmentação dos voxels da cena que esta representa. Por isso, interessamo-nos apenas nos voxels que mudaram de rótulo de um passo de segmentação para outro. Analisando os possíveis rótulos atribuídos a um voxel após segmentação, existem as três situações seguintes:

- O voxel "aparece": Ele era fundo (rótulo nulo em geral) e passa a ser objeto (rótulo não nulo), ou seja, potencialmente visível.
- O voxel "desaparece": Ele era objeto (rótulo não nulo) e passa a ser fundo (rótulo nulo), ou seja, invisível.
- O voxel muda de objeto: Ele troca um rótulo de objeto (não nulo) para outro de objeto diferente (não nulo). Pertence ainda a um objeto e, portanto, permanece potencialmente visível. Eventualmente, se os objetos são associados a cores diferentes, este voxel poderá causar uma mudança de cor na imagem, nos pixels correspondentes.

No capítulo anterior, vimos que, no processo de visualização direta, calculam-se e armazenam-se no *z-buffer* as distâncias do plano de visualização (imagem) aos voxels visíveis. Além disso, anotam-se os índices dos voxels visíveis num mapa de índices de tamanho do *z-buffer*. Esses dois mapas são responsáveis da imagem tonalizada observada. Após um novo passo da segmentação, é necessário atualizar, portanto, o *z-buffer* e o mapa de índice de voxel. Há duas tarefas principais:

1. Processar todos os voxels "aparecidos" para verificar se eles estão, de fato, visíveis ou não (isso vai depender do ponto de vista do observador que foi fixado). Se eles estiverem, deve-se atualizar os dois mapas.
2. Processar todos os voxels "desaparecidos" para verificar quais estavam visíveis e que deverão desaparecer dos mapas para serem substituídos por outros voxels situados atrás deles.

No primeiro caso, podemos decompor a tarefa dessa maneira (vide a figura 4.5):

- Efetua-se a projeção do voxel "aparecido" (*splatting*) no plano da imagem. A distância D do voxel ao plano é automaticamente calculada nesta projeção.

- Para os pixels atingidos (o número de pixels depende do tamanho n da máscara de $splatting$), compara-se a distância D calculada com o valor do *z-buffer* desses pixels. Quando D for menor ou igual, atualiza-se o valor do *z-buffer* assim como o índice do voxel projetado (no mapa de índices). Fazem-se, então, por voxel "aparecido": uma projeção e n comparações de distâncias (com enventuais atualizações nos mapas).

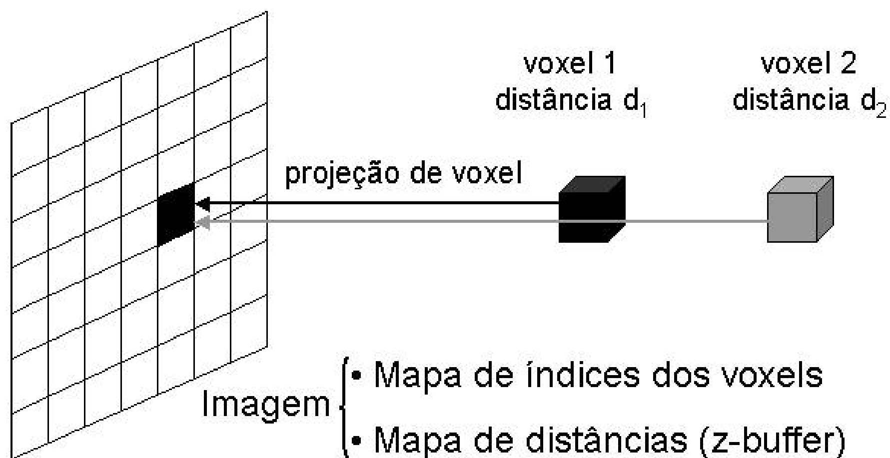


Figura 4.5: Imagem Incremental. Os voxels 1 e 2 *apareceram* depois de um passo de segmentação. Eles são projetados e sua distância ao plano de projeção é comparada à distância armazenada no *z-buffer*. A distância d_1 é a menor de todas, portanto, o voxel 1 é visível no pixel considerado.

O caso do voxel que muda de objeto e permanece potencialmente visível é tratado como o primeiro caso (vide a figura 4.6). É como se aparecesse um novo voxel de objeto, de mesmo índice, que fosse substituir o antigo voxel projetado e estivesse na mesma distância D. Atualiza-se apenas o mapa que indica o objeto (rótulo) associado aos pixels atingidos na imagem. Para poder fazer essa atualização, é imprescindível permitir a um voxel aparecer, mesmo que a sua distância D até a imagem esteja igual a esta já armazenada no *z-buffer*.

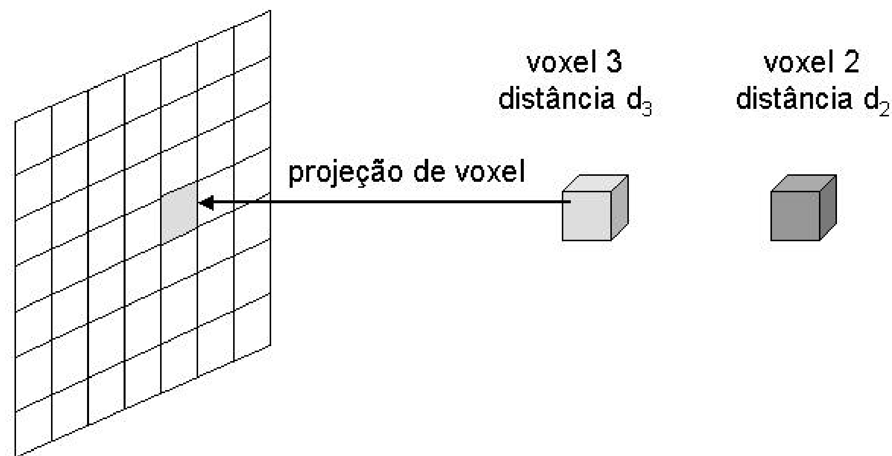


Figura 4.6: Imagem Incremental. O antigo voxel 1 foi reconquistado por outra região e *mudou* de rótulo: É agora chamado de voxel 3. Após projeção deste voxel, vê-se que a sua distância d_3 ao plano de projeção é igual à distância armazenada no *z-buffer* (d_1). Portanto, o conteúdo do pixel é atualizado com a cor correspondente ao rótulo do voxel 3.

No segundo caso, a decomposição é a seguinte (vide a figura 4.7):

- Efetua-se a projeção do voxel "desaparecido" (*splatting*).
- Para os n pixels atingidos, olham-se os índices dos voxels projetados previamente. Se o pixel contém o índice do voxel "desaparecido", tem que apagar esse índice e procurar qual novo voxel de trás vai substituir o voxel "desaparecido". Para isso, basta jogar-se um raio do pixel para a cena. Atualizam-se então os valores do *z-buffer* e do mapa de índices de voxel.

Fazem-se, portanto, por voxel "desaparecido": uma projeção e n comparações de índices com, no máximo, n *raycastings*.

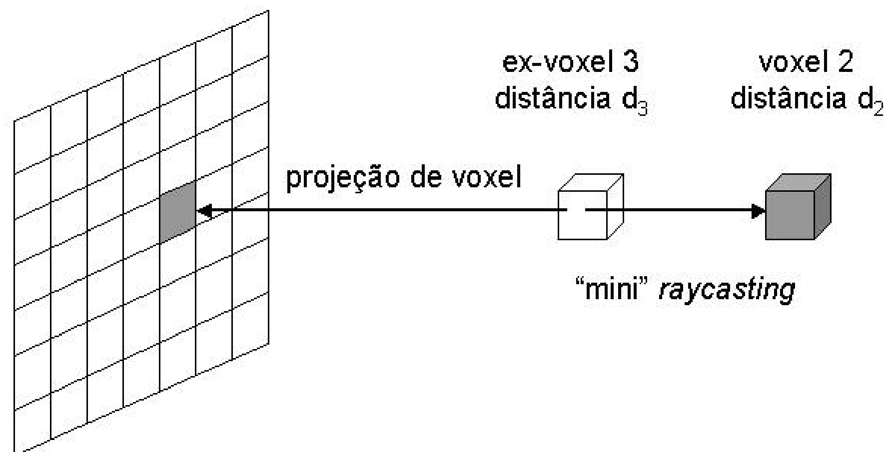


Figura 4.7: Imagem Incremental. O antigo voxel 3 *desapareceu*: foi conquistado pela região de "fundo". Após projeção do voxel 3 no plano da imagem, verifica-se que este estava bem visível (mapa dos índices de voxel) e joga-se um raio a partir dele para encontrar o voxel de trás atualmente visível: o voxel 2.

Pode-se observar que, na primeira situação, o algoritmo de "imagem incremental" tem o mesmo custo (uma projeção) que o algoritmo clássico de projeção de voxels, só que, provavelmente, não vai ser aplicado a todos os voxels da cena, e sim, apenas aos "aparecidos". Por outro lado, o custo do algoritmo na segunda situação é mais caro, pois, além da projeção de voxel, efetuam-se vários *raycastings*. Por isso, pode-se pensar que o algoritmo fica mais rápido quando se processam mais voxels "aparecidos" do que voxels "desaparecidos", isto é, quando as regiões de objeto vão crescendo a cada iteração da segmentação.

Para otimizar esse método, podemos reduzir o custo dos n *raycastings*. Se um voxel visível desapareceu, significa que ele era o primeiro a interceptar o raio. Portanto, o voxel que vai substituí-lo é necessariamente situado atrás dele, na direção de percurso do raio. A idéia é trazer diretamente o raio até o voxel "desaparecido" para ele continuar seu caminho a partir daí, sem ter que atravessar necessariamente toda a cena. Desta maneira, minimiza-se o caminho do raio dentro da cena e a complexidade do algoritmo.

Pode-se perguntar se a ordem de processamento dos voxels "aparecidos" e "desparecidos" importa. De fato, essa ordem não influi sobre o resultado final. À primeira vista, a maneira mais prudente de processar os voxels mudados seria começar por apagar todos os voxels "desaparecidos" e, depois, projetar todos os

voxels “aparecidos”. Pode-se pensar que tratar os voxels na ordem contrária impediria voxels “aparecidos” de aparecer na imagem, pois sua distância ao plano de projeção seria maior que a armazenada no *z-buffer*. Esta, porém, poderia corresponder a um voxel “desaparecido” e não ser mais válida. No entanto, quando este voxel “desaparecido” for tratado, graças à retroação dos *raycastings*, encontrar-se-á o voxel “aparecido” que fora impedido de aparecer. O método não pede nenhuma ordem de processamento particular. Portanto, cada voxel pode ser processado separadamente, assim que ele sair da fila de prioridade da IFT: O método de visualização fica embutido no processo de segmentação. Também se pode processar todos os voxels alterados depois de terminar o passo de segmentação, sem ordená-los nas categorias “aparecidos” ou “desaparecidos”. Neste caso, pouco importa o método de segmentação utilizado, pois basta comparar os mapas de rótulos anterior e posterior ao passo de segmentação. O método de atualização da imagem é independente do algoritmo de segmentação.

Se quisesse otimizar este método, deveria minimizar o número de raycastings efetuados, pois estes é que são caros. Sejam n_{ap} o número de voxels “aparecidos”, n_{des} o de voxels “desaparecidos” e n_{splat} o tamanho da máscara ou zona de *splatting*. Se processarmos primeiro todos os voxels “desaparecidos”, efetuaremos necessariamente e exatamente $n_{des} \times n_{splat}$ *raycastings*. Pelo contrário, se processarmos primeiro os voxels “aparecidos”, tem mais chances deles já substituirem voxels “desaparecidos” e, desta maneira, diminuirem o número de *raycastings* a serem feitos na hora de processar os voxels “desaparecidos”. O fato de processar primeiro os voxels “aparecidos” só pode diminuir a chance do voxel desaparecendo constar ainda como visível. Esta última otimização necessita armazenar separadamente os voxels “aparecidos” e “desaparecidos”, para tratá-los em dois passos separados.

4.3.2 Algoritmo

Apresentamos nesta seção uma versão genérica do algoritmo de imagem incremental, pois aplicável com qualquer método de segmentação, desde que se armazenam as cenas segmentadas corrente e anterior. Segmentação e visualização permanecem dois blocos distintos.

O algoritmo de imagem incremental (ver pseudocódigo F) recebe, como entradas, a imagem do passo anterior (mais exatamente, o *z-buffer* e o mapa dos índices dos voxels projetados nos pixels) e as cenas rotuladas (segmentadas) do passo corrente e do anterior. Percorre-se a cena inteiramente (l. 1). Para ser eficiente, a varredura linear *raster* é a mais indicada. Para cada voxel v da cena segmentada, comparam-se o rótulo corrente e o anteriormente atribuído. Se estes forem diferentes, projeta-se o voxel na imagem para ver em qual distância d dela ele se situa e em qual pixel p dela ele se projeta (l. 3 e 4). Se o novo rótulo atribuído for nulo, o voxel acabou de desaparecer (l. 5). Caso contrário, ele acabou de aparecer ou de mudar de objeto (l. 8).

No primeiro caso, para todos os pixels na zona de *splatting* do pixel atingido p (l. 6), verifica-se se os voxels projetados são justamente este voxel “desaparecido” (l. 7). No caso positivo, joga-se um raio a partir do pixel para encontrar o voxel mais próximo e atrás do “desaparecido”. Os mapas correspondentes

assumem a distância e o índice do novo voxel encontrado.

No segundo caso, para todos os pixels na zona de *splatting* do pixel atingido p (l. 9), verifica-se se a distância d do voxel "aparecido" é menor ou igual à armazenada no *z-buffer* (l. 10). No caso positivo, significa que este voxel é mais perto do plano de visualização que o voxel visível até agora, portanto, atualizam-se os mapas correspondentes com a distância e o índice deste voxel "aparecido".

Pseudocódigo F ***ImageUpdate***

Variáveis de entrada:
  * o z-buffer D (mapa das distâncias),*
  * o mapa V dos índices (raster) dos voxels visíveis na imagem (projetados/atingidos),*
  * o mapa dos novos rótulos R,*
  * o mapa A dos rótulos obtidos no passo anterior.*

Constantes de entrada:
  * os ângulos de visualização fixados $(\theta_x, \theta_y, \theta_z)$ contidos na variável θ,*
  * a zona ou máscara de splatting Z.*

Variáveis de saída:
  * o z-buffer D (mapa das distâncias),*
  * o mapa V dos índices (raster) dos voxels visíveis na imagem (projetados/atingidos).*

Variáveis auxiliares:
  * o rótulo novo r, o rótulo antigo a,*
  * o voxel v,*
  * o pixel p e seus vizinhos q,*
  * a distância d.*

1. $\forall v \in R$,
2. $r \leftarrow R[v]$; $a \leftarrow A[v]$;
3. **Se** $r \neq a$,
4. $(d, p) \leftarrow VoxelProject(v, \theta)$;
5. **Se** $r = 0$,
6. $\forall q \in Z(p)$,
7. **Se** $V[q] = v$, $(D[q], V[q]) \leftarrow DRaycast(q, R, \theta, d)$;
8. **Senão**,
9. $\forall q \in Z(p)$,
10. **Se** $d \leq D[q]$, $D[q] \leftarrow d$; $V[q] \leftarrow v$;

* $VoxelProject(v, \theta)$: *função que projeta na direção θ um único voxel v no plano da imagem e retorna a distância voxel-imagem e o índice do pixel atingido na imagem.*

* $DRaycast(q, R, \theta, d)$: *função que joga um único raio na direção θ a partir do pixel q de uma imagem, numa cena rotulada R até encontrar um voxel de objeto. Retorna a distância voxel-imagem e o índice do voxel atingido pelo raio. O raio é diretamente jogado na distância d da imagem e começa a percorrer a cena a partir desta distância.*

Capítulo 5

Resultados experimentais

5.1 Experimentos

Para avaliar e comparar os métodos propostos e os existentes, foram implementados os algoritmos citados e feitos vários experimentos sobre diferentes imagens. Esta seção descreve os experimentos e as cenas utilizadas nos testes.

5.1.1 Implementação

Os algoritmos de *watershed* via IFT (vide o pseudocódigo B), *watershed* iterativo via DIFT (vide os pseudocódigos C e D), o algoritmo de borda incremental via BDIFT (vide o pseudocódigo E) e o algoritmo de imagem incremental (vide o pseudocódigo F) foram implementados em linguagem C e seguem os pseudocódigos de perto.

Não houve preocupação especial em economia de memória. Algumas estruturas de dados específicas foram utilizadas. A fila hieráriquica, núcleo da IFT, utiliza uma fila circular com *buckets* e, internamente a cada *bucket*, listas duplamente ligadas para representar as filas FIFO, como explicado na seção 2.3.3. As inserções na fila ou as atualizações são executadas em $O(1)$, enquanto a remoção da fila pode levar $O(C_{max} + 1)$, onde $C_{max} + 1$ é o número de *buckets*. Os diversos mapas utilizados (rótulos, custos, predecessores) são cenas do tamanho da cena original. Juntos, eles constituem a estrutura "anotação" da cena. Os testes de pertinência a um conjunto são efetuados com auxílio de mapas de bits do tamanho da cena e, portanto, levam $O(1)$. Enfim, a borda dos objetos (no algoritmo BDIFT) é armazenada numa lista duplamente ligada. Tanto a inserção quanto a remoção de um elemento da borda é executado em $O(1)$, pois o índice do voxel corresponde ao endereço na memória. A grande desvantagem dessa estrutura de dados e das anteriores é o custo em memória: A borda ocupa o tamanho de duas cenas originais, apesar de utilizar efetivamente apenas 3% do tamanho da cena original.

5.1.2 Testes

Efetuaram-se 5 experimentos, para se comparar o desempenho dos algoritmos. Eles correspondem às estratégias apresentadas na seção 4.1:

1. **IFT + *rendering* da cena**: Esta estratégia clássica (vide a figura 4.1) é composta de uma segmentação por *watershed* não iterativa via IFT seguida da visualização da cena por cinco técnicas diferentes de *rendering*: *splatting* (**S**), *splatting* com fatoração *shear-warp* (**S-SW**), *raycasting* (**R**), *raycasting* com fatoração *shear-warp* (**R-SW**) e *raycasting* com o algoritmo de Bresenham tridimensional para a construção dos raios (**R-B**).
2. **DIFT + *splatting* da cena** (**S**): Essa segunda abordagem combina segmentação iterativa com projeção dos voxels da cena inteira (vide a figura 4.2).
3. **BDIFT + *splatting* de borda** (**S-Borda**): Esse é o primeiro algoritmo proposto (vide a figura 4.3). Ele projeta apenas as bordas dos objetos, atualizadas *durante* a segmentação iterativa via BDIFT.
4. **DIFT + detecção (D-Borda) e *splatting* de borda** (**S-Borda**): Esse experimento é similar ao anterior, exceto que a extração da borda é feita de força bruta *depois* de cada segmentação via DIFT.
5. **DIFT + atualização da imagem (A-Img)**: Segmentação e visualização são incrementais. A imagem é atualizada em função das mudanças observadas na segmentação (vide o algoritmo da imagem incremental e a figura 4.4).

A tonalização das imagens de todos os experimentos foi feita segundo o modelo de Phong (ver seção 3.4.2), utilizando uma estimativa de normal baseada em objetos (ver Apêndice C para mais detalhes).

5.1.3 Material

Os 5 programas de testes foram rodados em dois tipos de máquinas. A primeira máquina (**M1**) é um PC com processador Pentium 4 de 2,4 GHz e 1 GByte de RAM rodando Windows 2000. A segunda (**M2**) é uma Sun Enterprise 450, de arquitetura Sparc Ultra (sun4u) com 2 processadores Ultra-Sparc II de 450MHz, 512MB de RAM e rodando Solaris 7.

Os programas foram compilados com Microsoft Visual C++ 6.0 para o ambiente Windows, enquanto foi usado o compilador `gcc` com a opção de otimização `-O3` no ambiente Unix.

Os tempos de execução foram medidos com as funções `QueryPerformanceFrequency` e `QueryPerformanceCounter` no Windows e com a função `gettimeofday` no Unix, na precisão do microssegundo. Os tempos medidos são, portanto, os tempos em condições reais. No caso da máquina M2 (SUN-Solaris) que é servidora do laboratório, isso é para ser levado em conta, pois ela pode estar atendendo vários processos de outros usuários no mesmo tempo.

5.1.4 Imagens de teste

Nome	Modalidade	Dimensões	No. de voxels	Profundidade de voxel
cena1	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
cena2	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
cena3	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
cena4	IRM	$225 \times 175 \times 173$	6 811 875	16 bits
cena5	IRM	$234 \times 183 \times 195$	8 350 290	16 bits
brain	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
brain2	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
brain3	IRM	$181 \times 217 \times 181$	7 109 137	16 bits
brain_interp211	IRM	$361 \times 217 \times 181$	14 178 997	16 bits
brain_interp121	IRM	$181 \times 433 \times 181$	14 185 513	16 bits
brain_interp112	IRM	$181 \times 217 \times 361$	14 178 997	16 bits
brain_interp122	IRM	$181 \times 433 \times 361$	28 292 653	16 bits
brain_interp212	IRM	$361 \times 217 \times 361$	28 279 657	16 bits
brain_interp221	IRM	$361 \times 433 \times 181$	28 292 653	16 bits
knee	RX	$171 \times 193 \times 100$	3 300 300	8 bits
geometric	sintética	$200 \times 200 \times 100$	4 000 000	8 bits
geometric_interp211	sintética	$399 \times 200 \times 100$	7 980 000	8 bits
geometric_interp121	sintética	$200 \times 399 \times 100$	7 980 000	8 bits
geometric_interp112	sintética	$200 \times 200 \times 199$	7 960 000	8 bits
geometric_interp122	sintética	$200 \times 399 \times 199$	15 880 200	8 bits
geometric_interp212	sintética	$399 \times 200 \times 199$	15 880 200	8 bits
geometric_interp221	sintética	$399 \times 399 \times 100$	15 920 100	8 bits
geometric_interp222	sintética	$399 \times 399 \times 199$	31 680 999	8 bits

Tabela 5.1: Cenas de teste. O nome das cenas interpoladas comporta o sufixo $_interpXYZ$ onde X, Y e Z são os coeficientes de interpolação segundo os eixos x, y e z respectivamente.

Testaram-se os algoritmos implementados sobre três tipos de imagens (apresentadas no apêndice E, figuras E.1 e E.2): imagens de *ressonância magnética* (“*brain*”, “*cena1*”, “*cena2*”, “*cena3*”, “*cena4*”, “*cena5*”), imagem da modalidade *CT por raio-X* (“*knee*”) e cena *sintética* (“*geometric*”). As IRM representam um cérebro humano, a cena “*knee*” contém uma secção do osso do joelho e a cena “*geometric*” descreve duas esferas grandes coladas e outra pequena, embutida num cubo. Apesar de estarmos mais interessados nas IRM, por elas oferecerem desafio maior em segmentação, os outros tipos de imagem serviram a validar os métodos de segmentação e visualização, pois tinham formas diferentes (superfícies lisas no caso da cena de síntese). Além disso, a cena sintética, composta de padrões conhecidos, permitiu verificar a exatidão dos resultados de segmentação e visualização.

As cenas têm voxels de profundidade variada. As IRM contêm voxels de profundidade 16 bits (2 bytes) que, na verdade, codificam 4096 níveis de cinza (apenas 12 bits úteis). As outras têm voxels com intensidade variando entre 0 e 255 (8 bits).

Quanto ao tamanho das imagens testadas, ele varia de 3 a 7 milhões de voxels. Para poder ver a influência do tamanho da cena sobre a eficiência dos algoritmos, geraram-se cenas interpoladas a partir das cenas "*brain*" e "*geometric*". A interpolação trilinear permite ter cenas maiores com a mesma complexidade morfológica que as originais. Contando as diferentes interpolações criadas, foram 23 imagens de teste, com tamanho variando de 3 a 32 milhões de voxels, nas quais foram aplicados os 5 experimentos citados na seção 5.1.2. A tabela 5.1 resume as características principais destas. Cenas com mais de 32 milhões de voxels foram testadas, mas o computador não conseguiu executar por causa da limitação de memória RAM.

Observa-se que a maioria das IRM tem como dimensões $181 \times 217 \times 181$. Este é o tamanho do espaço estereotáxico de Talairach (Talairach e Tournoux, 1988). É neste espaço bem comum em neurologia que se registra uma imagen, isto é, mapeia-se uma imagem, aplicando transformações lineares, de forma que várias imagens de diferentes pacientes fiquem com a mesma orientação e mesmo tamanho. A grande maioria das imagens em neurologia é analisada após o registro, mas o dado original, às vezes indo até o pescoço, pode conter cerca de 11 milhões de voxels.

Antes de se testarem as cenas com os diferentes métodos baseados em *watershed*, calculou-se o gradiente morfológico destas. É no gradiente das cenas que se aplicam os diferentes algoritmos.

5.1.5 Roteiros de interatividade

Como quatro dos experimentos são segmentações iterativas (os experimentos 2 a 5 utilizam a DIFT ou a BDIFT), foi definido um protocolo por tipo de imagem, isto é, os conjuntos de sementes e eliminadores acrescentados em cada passo de uma segmentação ficticiamente interativa. O roteiro genérico das interações do usuário durante uma segmentação semi-automática por DIFT é apresentado na figura 5.1 a seguir. Para começar uma segmentação, adicionam-se pelo menos duas sementes: uma para designar o fundo e outra para destacar o objeto de interesse. Nenhum eliminador é inserido. Após cada segmentação via DIFT, o usuário julga a qualidade da segmentação obtida. Se as mudanças visualizadas são as esperadas, o processo interativo cessa: A cena foi segmentada corretamente. Caso o usuário não esteja plenamente satisfeito, ele adiciona eliminadores nas zonas que lhe parecem erradas e/ou adiciona novas sementes para refinar a segmentação. A DIFT é novamente processada até conseguir um resultado satisfatório. Um roteiro análogo aplica-se no caso da segmentação por BDIFT.

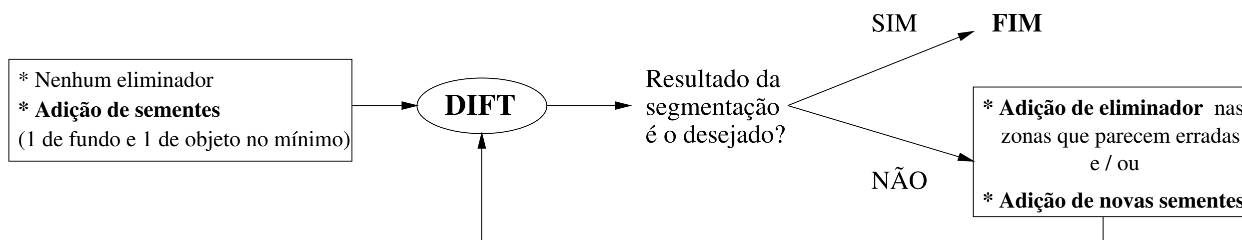


Figura 5.1: Roteiro genérico das interações do usuário durante uma segmentação semi-automática por DIFT.

Observa-se que a adição de sementes só (sem eliminador) nem sempre permite observar uma mudança na segmentação, pois é possível que os vizinhos das sementes já tenham custo menor que o custo oferecido pelas novas sementes. É o caso das sementes inférteis já discutido na seção 2.4.3. Para evitar tais incômodos, adicionam-se eliminadores no mesmo lugar das sementes adicionadas. Desta maneira, os eliminadores "liberam" espaço (remoção de árvore) e garante-se que as sementes possam propagar-se. Por isso, eliminadores foram utilizados nos roteiros aplicados às cenas "*geometric*" e "*knee*".

Também se podem esperar mudanças na segmentação ao adicionar apenas eliminadores sem novas sementes. Neste caso, os voxels ocupados pela árvore removida vão ser reconquistados pelos voxels fronteiras da árvore, pertencentes a árvores antigas não removidas (vide a figura 2.3).

Enfim, vale a pena ressaltar que não só os conjuntos de sementes e eliminadores inseridos importam, mas também sua ordem de inserção. Como foi explicado na seção 2.4.3, o resultado final da segmentação feita a partir de um conjunto de sementes não é único. Depende da ordem na qual as sementes foram inseridas e se elas concorreram simultaneamente num mesmo passo ou seqüencialmente em vários passos. Por isso, a adoção de um roteiro único por imagem permite comparar os resultados intermediários (equivalentes) dados pelos diversos métodos. As tabelas (5.2 e 5.3) a seguir descrevem os diferentes roteiros utilizados. No caso da cena de RM "*brain*" (cujas segmentações intermediárias são apresentadas nas figuras E.4, E.5 e E.6 do apêndice E), foram aplicados dois outros roteiros para multiplicar os tipos de situação que possam ocorrer:

- O roteiro "*brain2*" aplica o mesmo conjunto de sementes que no roteiro para "*brain*", só que a ordem de inserção das sementes é diferente.
- O roteiro "*brain3*" aplica as mesmas sementes, mas com rótulos diferentes, de forma que apenas os 3 objetos de interesse menores apareçam (cerebelo, medula e ventrículos laterais), ao invés dos 4 habituais (cérebro além dos três citados). Este roteiro testa, portanto, a segmentação e a visualização de objetos pequenos em cena grande (vide as figuras E.7 e E.8 do apêndice E).

cenas	*geometric*		*knee*	
iteração	N_S	N_E	N_S	N_E
1	3	0	3	0
2	4	1	5	1
3	5	1	6	0

Tabela 5.2: Roteiro para as cenas "*geometric*" e suas 7 interpolações e a cena "*knee*". Para cada iteração da segmentação, N_S e N_E designam respectivamente o número cumulado de sementes e o número de eliminadores inseridos pelo usuário.

iteração	brain	brain2	brain3	cena1	cena3	cena4	cena5
1	660	1150	750	600	950	650	700
2	730	1350	950	1000	1125	850	1000
3	1030	1550	1150	1400	1300	1050	1300
4	1230	1750	1350	1800	1475	1200	1600
5	1400	1950	1550	2200	1650	1300	1900
6	1550	2105	1750	2600	1825	1400	2200
7	1955	–	1950	3000	2000	1500	2500
8	2105	–	2105	3279	2250	1563	2801

Tabela 5.3: Roteiro para as cenas de RM. A tabela indica o número cumulado de sementes inseridas pelo usuário em cada iteração da segmentação. As 6 interpolações de "*brain*" seguem o mesmo roteiro que "*brain*". A "*cena2*" segue o mesmo roteiro que "*cena1*". Na prática, o usuário seleciona as sementes com o *mouse* em cortes 2D da cena. Ele sinaliza uma região de um traço de *mouse*, inserindo desta maneira centenas de voxels-sementes. Isso explica o número elevado de voxels-sementes nos roteiros acima.

5.2 Discussão dos resultados

Os resultados obtidos pelos 5 métodos podem ser compararados segundo os critérios seguintes:

1. **Eficiência**: Nenhuma interface foi implementada, por isso, só são comparados os tempos reais dedicados exclusivamente para as tarefas de segmentação e visualização, ignorando partes comuns como leitura de cena do arquivo.
2. **Qualidade**: Foram analisadas as qualidades dos *renderings* das imagens obtidas.
3. **Rotação**: Foi discutido o custo adicional para rotacionar os objetos segmentados, isto é, para mudar de ponto de vista na cena. Esse recurso é particularmente útil para a manipulação dos objetos, a fim de examinar e melhorar a segmentação obtida.

Antes da comparação dos métodos de segmentação e visualização, serão comparadas 5 técnicas de *rendering* sem preocupação do método de segmentação (seção 5.2.1). Em seguida, serão analisados os 5 experimentos apresentados na seção 5.1.2: primeiro, apenas do ponto de vista da fase de segmentação (seção 5.2.2); segundo, levando em conta apenas a etapa de visualização (seção 5.2.3); e enfim, comparando as estratégias de segmentação e visualização na sua totalidade (seção 5.2.4).

5.2.1 Comparação das técnicas de *rendering*

O experimento 1 permitiu comparar 5 técnicas de *rendering* diferentes: o *splatting* (**S**), o *splatting* com fatoração *shear-warp* (**S-SW**), o *raycasting* (**R**), o *raycasting* com fatoração *shear-warp* (**R-SW**) e o *raycasting* com o algoritmo de Bresenham tridimensional para a construção dos raios (**R-B**).

Para nosso conjunto de cenas (de 3 a 32 milhões de voxels), o *splatting* **S** revelou-se mais rápido que o *raycasting* e utilizou em média 55% do tempo necessário a um *raycasting* **R**, quando executados na máquina **M1** (Windows), como mostra o diagrama 5.2 (a). Tipicamente, para a cena "*brain*" de 7 milhões de voxels, (**S**) leva 235 ms enquanto (**R**) leva 449 ms (ver tabela 5.4). As mesmas experiências realizadas na máquina **M2** (Sun-Solaris) já não mostram mais tanta diferença: Em média, o *splatting* **S** demora 98% do tempo de um *raycasting* **R** (ver diagrama 5.2 (b)). Essa diferença deve-se à diferença de arquitetura e de compilador.

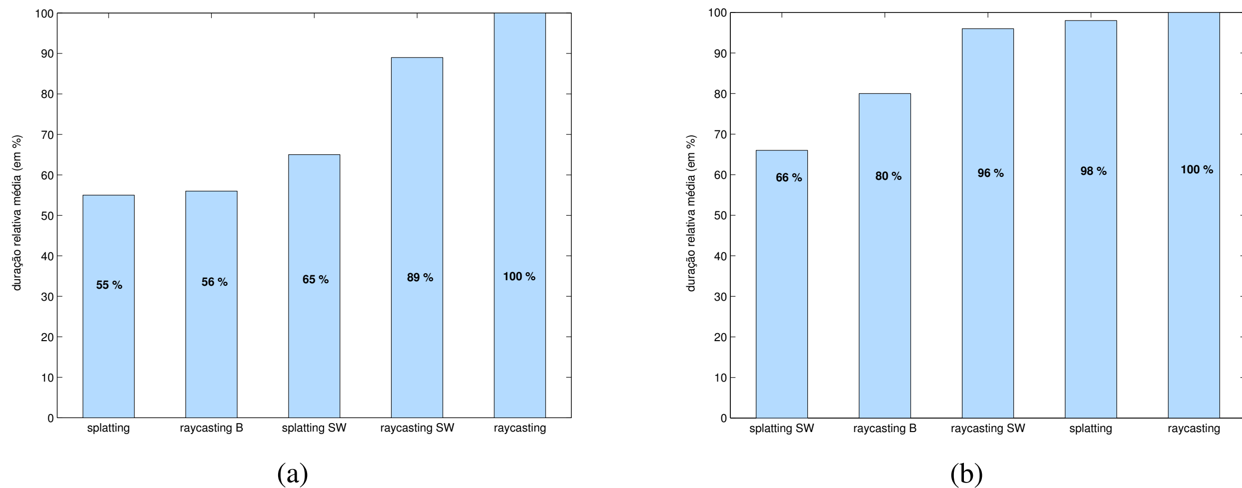


Figura 5.2: Tempos relativos médios de *rendering* nas máquinas M1 (**a**) e M2 (**b**) para as técnicas de: *splatting* (**S**), *raycasting* com Bresenham 3D (**R-B**), *splatting* com fatoração *shear-warp* (**S-SW**), *raycasting* com fatoração *shear-warp* (**R-SW**) e *raycasting* (**R**).

Máquina	S	R	S-SW	R-SW	R-B
M1	235	449	254	409	251
M2	1 561	2 527	1 599	2 443	1 968

Tabela 5.4: Tempos de *rendering* (em ms) da cena "*brain*" nas máquinas M1 e M2, usando as técnicas de *splatting* (**S**), *raycasting* (**R**), *splatting* com *shear-warp* (**S-SW**), *raycasting* com *shear-warp* (**R-SW**) e *raycasting* com Bresenham 3D (**R-B**).

A nossa proposta de *raycasting*, utilizando internamente o algoritmo de Bresenham tridimensional para calcular a trajetória dos raios (**R-B**), é da mesma ordem de velocidade que o *splatting* (**S**) (ver diagrama 5.2 (a)). Tipicamente, para a cena "*brain*", (**R-B**) leva 251 ms. Além disso, a tendência é que o método de *raycasting* com Bresenham (**R-B**) seja mais rápido quando aumenta o número de voxels de objeto. Com efeito, aumentando-se o número de voxels de objeto, aumenta-se o número de projeções de voxel (multiplicação matricial e comparação no *z-buffer*) no caso de qualquer tipo de *splatting*, enquanto que, no caso de qualquer *raycasting*, não se aumenta o número de raios a serem jogados e, além disso, aumenta-se a chance de um raio encontrar um voxel mais rapidamente. Como foi explicado na seção 3.3.2, a principal

desvantagem do método *raycasting* é a perda de tempo ao percorrer a cena inteira até encontrar um objeto. Quando a cena está mais "cheia" (mais voxels de objeto), esse tempo de percurso é amortecido. Portanto, os métodos de *raycasting* tendem a ter um tempo de execução amortecido enquanto os *splattings* precisam de um tempo mais ou menos linear em relação ao número de voxels de objeto.

Quanto à qualidade das imagens, ela é geralmente parecida. No entanto, pode-se notar maior diferença para ângulos oblíquos, ou seja, não múltiplos de 90 graus (vide a figura E.3 do apêndice E). Nesse caso, a qualidade do *raycasting* é superior à do *splatting*. A projeção de voxel tem o problema inerente de gerar buracos na imagem, problema resolvido em parte pelo *splatting* que espalha os voxels projetados em pixels nos pixels da vizinhança. A escolha dessa vizinhança, ou tamanho do *splatting*, é decisiva na qualidade da visualização. Tipicamente, usou-se a vizinhança 3×3 para o *splatting*. Uma vizinhança 2×2 também é possível mas não é simétrica. Vizinhanças simétricas maiores (5×5 e 7×7) também poderiam ser usadas. Tem um compromisso entre evitar buracos (pontinhos pretos) na imagem e obter uma imagem que parece borrada no caso de vizinhança grande demais. A escolha do tamanho do *splatting* depende dos tamanhos relativos voxel/pixel e, por conseqüência, da distância e do tamanho do plano da imagem e da cena observada.

A aparição de buracos não é visível nos nossos experimentos, pois não há grande diferença entre o tamanho do voxel e o do pixel. O observador está sempre a uma distância razoável da cena: ele não está situado dentro da cena. Por outro lado, a imagem é dimensionada em relação às maiores dimensões da cena, de forma que esta caiba sempre na imagem, qualquer que seja o ângulo de visualização. Se o observador, portanto o plano da imagem, estivesse mais perto dos objetos, entrando na cena, o tamanho do voxel seria bem maior em relação ao tamanho do pixel. Como antes, cada voxel projetar-se-ia num só pixel (aquele situado mais perto da projeção do centro do voxel). Os pixels vizinhos corresponderiam a nenhum centro de voxel e apareceriam preto apesar de ficar na zona de projeção do voxel (cubinho e não só o centro do cubo). Por isso, nesse caso, a operação de *splatting* é muito importante para preencher esses buracos. No entanto, o espalhamento provocado pelo *splatting* é aplicado para todos os pixels que tenham vizinhos sem voxel projetado. Isso resulta em espalhar os pixels que formam o contorno real do objeto em pixels que deveriam representar o fundo. As bordas vão aparecer borradas, ou como aumentadas em relação a uma imagem obtida por *raycasting*. O *raycasting* dá uma imagem que parece mais nítida nesse caso.

A fatoração *shear-warp* (**R-SW**) acelera um pouco o processo de *raycasting*, sem, todavia, apresentar o desempenho da aceleração por Bresenham 3D (**R-B**). A técnica **R-SW** leva 89% (respectivamente 96%) do tempo gasto por **R**, enquanto **R-B** só leva 56% (80 %) do tempo de **R** com a máquina M1 (M2): tipicamente, (**R-SW**) demora 409 ms, (**R**) 449 ms e (**R-B**) 251 ms (para a cena "*brain*").

Quanto ao método de *splatting* com *shear-warp* (**S-SW**), ele traz poucas vantagens em relação ao *splatting* clássico (**S**), na máquina M1: **S-SW** leva 65% do tempo de **R** enquanto **S** leva apenas 55% do tempo de **R**. Por exemplo, (**S-SW**) com 254 ms não melhora (**S**) que demora 235 ms (para a cena "*brain*"). Na máquina M2, porém, (**S-SW**) torna-se mais rápido que o *raycasting* com Bresenham (**R-B**) ou o próprio

splatting (**S**). Infelizmente, essa fatoração junta ao *splatting* introduz artefatos não desejáveis na imagem para vistas não ortogonais. No caso do *raycasting*, a qualidade do resultado não muda ao usar ou não a fatoração *shear-warp*.

A fatoração *shear-warp* tem outras vantagens, citadas em (Lacroute, 1995), que não foram utilizadas no nosso caso (vista perspectiva). Também não foi implementada a mesma estrutura de dados proposta pelos autores (com código de corrida), que melhoraria o tempo de processamento.

Por simplicidade, escolhemos a técnica de *splatting* para visualizar os resultados da segmentação no experimento 2. Além disso, ela revelou-se mais rápida na máquina M1, máquina na qual foram efetuados os testes que dão embasamento quantitativo às comparações a seguir.

5.2.2 Comparação das estratégias em relação à segmentação

Dos 5 experimentos efetuados, um utiliza a IFT (experimento 1), três a DIFT (experimentos 2, 4 e 5) e outro a BDIFT (experimento 3), para segmentar as cenas de teste.

A figura 5.3, que corresponde aos valores da tabela 5.5, mostra a duração de cada iteração para a segmentação interativa da cena "*brain*" com os três tipos de segmentação. Enquanto a IFT tem duração constante (cerca de 16 s) em cada passo da segmentação, a DIFT e a BDIFT variam. Depois de uma primeira iteração mais demorada (de 17 a 21 s), esses métodos são muito mais rápidos (menos de 4,5 s). Percebe-se claramente a grande vantagem dos algoritmos iterativos (DIFT e BDIFT) no diagrama da figura 5.4, que apresenta a duração relativa média de um passo de segmentação (posterior ao primeiro), medida no conjunto de cenas de teste. O fato de reaproveitar em cada passo o resultado da segmentação anterior permite economizar, em média, 76% (respectivamente 73%) do tempo necessário para segmentar a cena do zero, ou seja, a DIFT (BDIFT) demora em média apenas 24% (27%) da duração da IFT, nas iterações posteriores à primeira. A primeira iteração da DIFT (BDIFT) é mais demorada que uma IFT: 3% mais longo em média (46% respectivamente). Essa perda de tempo, no entanto, é amplamente compensada pelo ganho de tempo nos passos seguintes.

O custo superior da BDIFT em relação à DIFT (em média, 14% mais demorado que a DIFT, vide diagrama 5.4) é devido ao fato da BDIFT extrair a borda dos objetos ao segmentar. Esse tempo adicional gasto é, entretanto, minimizado. Se compararmos os experimentos 3 e 4, todos dois voltados à uma estratégia de visualização de borda, observa-se que o método de força bruta para a extração de borda é mais demorado. Com efeito, extrair a borda dos objetos após cada segmentação por DIFT (como é feito no experimento 4) leva uns 665 ms para uma cena de 4 milhões de voxels e aproximadamente 5,5 s para uma cena de 32 milhões de voxels. Deve-se adicionar esse tempo de detecção de borda ao tempo da DIFT para compará-lo à duração da BDIFT. No total, para segmentar a cena e detectar a borda dos objetos, a BDIFT revela-se mais rápida, pois utiliza, em média, apenas 79% do tempo levado pelo experimento 4 (detecção não embutida na segmentação). O resultado explica-se desta maneira: No experimento 3, a BDIFT apenas atualiza a borda

dos objetos, aproveitando-se do resultado anterior e da varredura dos voxels já necessária à segmentação. Já no experimento 4, a extração das bordas feita após cada segmentação não aproveita nenhum resultado anterior, e, portanto, efetua uma varredura explícita suplementar da cena inteira, para detectar os voxels com rótulo de objeto (não nulo) e com pelo menos um vizinho cujo rótulo é de fundo (nulo).

iteração	IFT	DIFT	BDIFT	DIFT/IFT	BDIFT/IFT	BDIFT/DIFT
1	15.835	16.800	20.926	106%	132%	125%
2	15.694	0.811	0.825	5%	5%	102%
3	15.694	3.631	4.388	23%	28%	121%
4	15.847	3.441	4.141	22%	26%	120%
5	15.821	1.169	1.281	7%	8%	110%
6	15.801	1.044	1.070	7%	7%	102%
7	15.895	1.471	1.569	9%	10%	107%
8	15.921	0.838	0.779	5%	5%	93%
média das iterações						
1 a 8	15.814	3.651	4.372	23%	28%	110%
2 a 8	15.811	1.772	2.007	11%	13%	108%

Tabela 5.5: Tempos de segmentação (em s) de *"brain"* para cada iteração nos experimentos 1, 2 e 3 (via IFT, DIFT e BDIFT respectivamente). À direita da tabela aparecem as razões desses tempos para cada iteração. A parte de baixo mostra a média dos tempos ou das razões para todos os passos e para os posteriores ao primeiro passo.

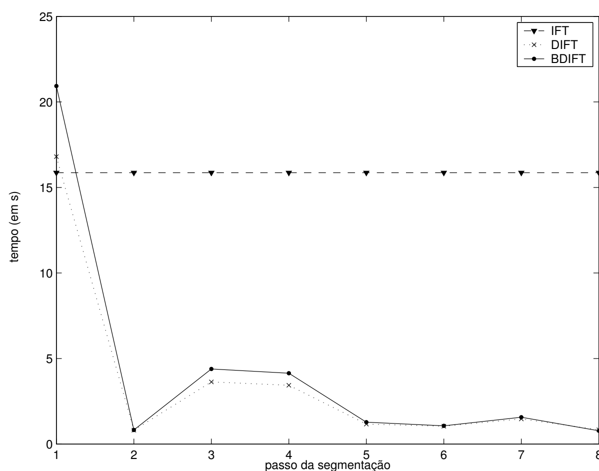


Figura 5.3: Tempos de segmentação de "*brain*" para cada iteração usando: a IFT, a DIFT ou a BDIFT (experimentos 1 a 3 respectivamente) .

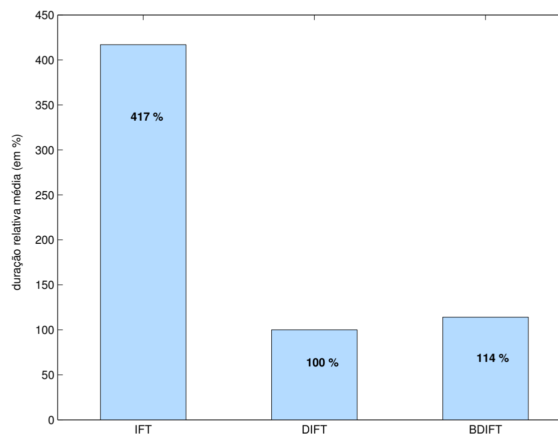


Figura 5.4: Duração relativa média de um passo de segmentação (passos posteriores ao primeiro) usando: a IFT, a DIFT ou a BDIFT (experimentos 1 a 3 respectivamente).

5.2.3 Comparação das estratégias em relação à visualização

Três estratégias de visualização diferentes são utilizadas nos experimentos: o *rendering* clássico de projeção da cena – *splatting* (**S**) – (experimentos 1 e 2); o *rendering* por projeção da borda – *splatting* de borda (**S-Borda**) – (experimentos 3 e 4); e um *rendering* mixto de *splatting* e *raycasting* – atualização de imagem (**A-Img**) – (experimento 5).

A vantagem dos métodos de projeção de voxels de borda (experimentos 3 e 4) em relação à projeção clássica da cena é que a borda contém bem menos voxels que a cena. Por exemplo, em todas as cenas de teste, a borda representa menos de 3,3% dos voxels da cena. Por outro lado, no espaço discreto das imagens digitais, o número de voxels descrevendo a borda do objeto é sempre inferior ou igual ao número de voxels contidos no objeto. Portanto, é preciso efetuar menos projeções do que para um *splatting* da cena inteira. Logo, o método de visualização por projeção de borda (**S-Borda**) é geralmente mais rápido que o de projeção de cena (**S**), como mostram o gráfico da figura 5.5 e a tabela 5.6 correspondente, no caso particular da visualização de "*brain3*": 45 ms em média para (**S-Borda**), contra 131 ms para (**S**). A diferença importante explica-se também pelo fato da borda representar apenas 0,1% dos voxels da cena. Para o conjunto das cenas de teste, mediu-se que, em média, um *splatting* de borda leva aproximadamente 58% do tempo gasto ao efetuar o *splatting* da cena inteira (vide o diagrama da figura 5.6).

No caso de "*geometric*", o *splatting* da borda (66 000 voxels de borda) demora 115 ms em média enquanto a cena correspondente (4 milhões de voxels) é projetada em 208 ms. Já, para "*geometric_interp2*", 460 ms são necessários para o *splatting* da borda de 265 milhares de voxels e 1,29 s para projetar a cena de 32 milhões de voxels. Enquanto que para "*brain_interp211*", os tempos tornam-se 280 ms e 465 ms para projetar respectivamente os 400 000 voxels de borda e os 14 milhões de voxels na cena. Com esses exemplos suplementares, observa-se que o tempo de *splatting* não é simplesmente proporcional ao número de voxels projetados, pois no caso de projeção de borda, o acesso aos voxels não é ordenado mas aleatório e, portanto, demora mais tempo que no caso de projeção dos voxels da cena, onde se processam todos os voxels consecutivos.

O método de visualização por atualização da imagem (**A-Img**) não mostra um desempenho tão bom quanto o *splatting* de borda, mas consegue, no entanto, economizar em média 24% do tempo necessário ao *splatting* clássico (ver o diagrama da figura 5.6). A figura 5.5 e a tabela 5.6 mostra os tempos correspondentes à atualização da imagem em cada passo da segmentação de "*brain3*". Deve-se notar que a atualização é mais rápida quando poucos voxels mudam de rótulo, isto é, no caso de correção de detalhes finos na cena segmentada.

Enfim, a qualidade das imagens obtidas pela atualização da imagem e pelos *splattings* de cena ou de borda é comparável.

iteração	**S**	**S-Borda**	**A-Img**	**S-Borda / S**	**A-Img / S**
1	109	41	78	37%	72%
2	131	40	113	31%	86%
3	129	41	81	32%	63%
4	129	48	109	37%	85%
5	138	47	116	34%	84%
6	138	47	88	34%	64%
7	138	47	88	34%	64%
8	138	47	88	34%	64%
média	131	45	95	34%	73%

Tabela 5.6: Tempos de visualização (em ms) para cada passo da segmentação de "*brain3*" usando: o *splatting* de borda (**S-Borda**), a atualização da imagem (**A-Img**) ou o *splatting* de cena (**S**) (experimentos 3, 5 e 2 respectivamente). À direita da tabela aparecem as razões desses tempos para cada iteração. A parte de baixo mostra a média dos tempos ou das razões.

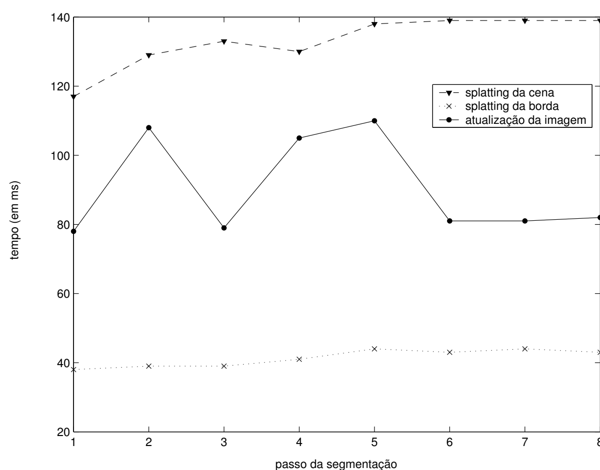


Figura 5.5: Tempos de visualização para cada passo da segmentação de "*brain3*" usando: o *splatting* de borda (**S-Borda**), a atualização da imagem (**A-Img**) ou o *splatting* de cena (**S**) (experimentos 3, 5 e 2 respectivamente) .

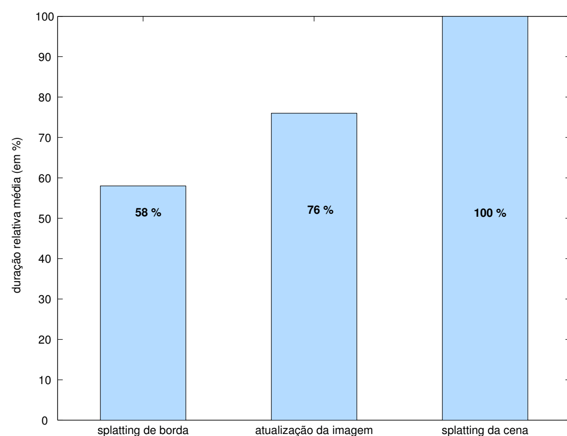


Figura 5.6: Duração relativa média da visualização (em cada passo posteriores ao primeiro) usando: a projeção de borda (**S-Borda**), a atualização da imagem (**A-Img**) ou a projeção de cena (**S**) (experimentos 3, 5 e 2 respectivamente).

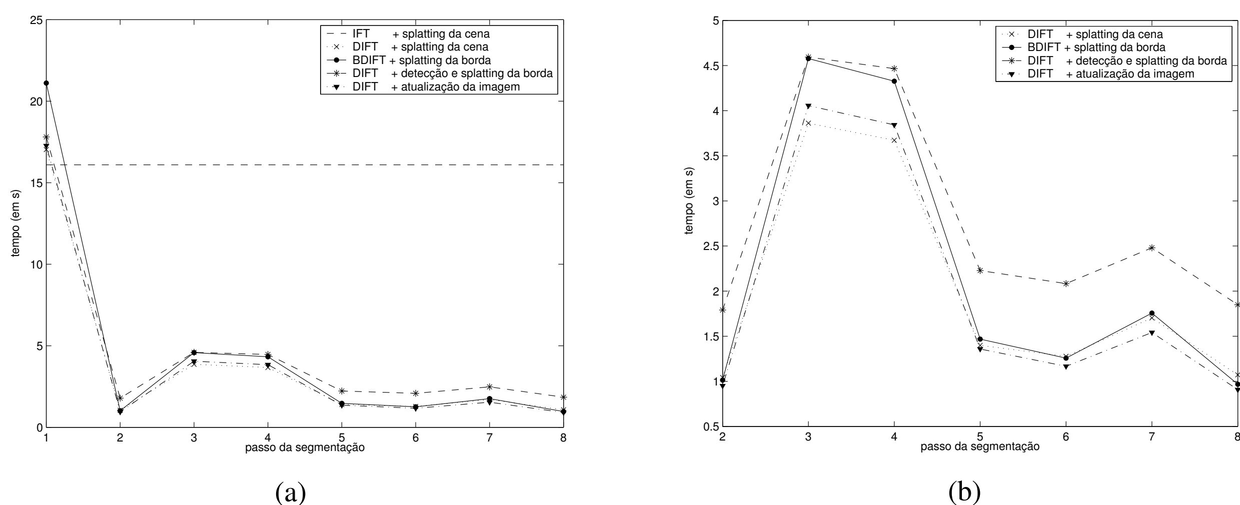
5.2.4 Comparação geral das estratégias de segmentação e visualização

Após as comparações parciais das estratégias dos 5 experimentos efetuados, analisamos de um ponto de visto global esses métodos de segmentação e visualização.

A estratégia do experimento 1 é obviamente a mais lenta de todas, pois o algoritmo de segmentação (IFT) não é iterativo (o processo de segmentação é preponderante na duração total de um método). Por

isso, sempre é preferido o método do experimento 2 (via DIFT). A comparação é imediata na figura 5.7 (a) que reúne os tempos totais de cada experimento para cada iteração da segmentação de “*brain*”. A tabela 5.7 apresenta os tempos correspondentes: uma média de 16 s por iteração contra apenas 2 s no caso do experimento 2.

Vimos também na seção 5.2.2 que o experimento 4 tinha sido efetuado para comprovar a velocidade do experimento 3 (via BDIFT), onde detecção de borda é embutida na segmentação. O exemplo particular da figura 5.7 (b) confirma que o experimento 4 sempre é mais lento que o 3: uma média de 2,8 s por iteração em vez de 2,2 s no caso da cena “*brain*” (ver tabela 5.7). Portanto, esses experimentos 1 e 4 não entrarão mais na nossa discussão.



(a) (b)

Figura 5.7: Tempo total de segmentação e visualização em cada passo (cena “*brain*”). (**a**) usando os experimentos 1 a 5: (**IFT + S**), (**DIFT + S**), (**BDIFT + S-Borda**), (**DIFT + D-Borda + S-Borda**) ou (**DIFT + A-Img**). (**b**) zoom nas iterações 2 a 8 para os experimentos 2 a 5.

iteração	exp1	exp2	exp3	exp4	exp5
1	16.069	17.031	21.113	17.806	17.269
2	15.928	1.042	1.014	1.790	0.950
3	15.929	3.862	4.578	4.594	4.056
4	16.081	3.672	4.327	4.467	3.843
5	16.056	1.399	1.468	2.228	1.358
6	16.035	1.278	1.258	2.084	1.166
7	16.130	1.704	1.757	2.480	1.542
8	16.156	1.072	0.967	1.849	0.909
média	16.045	2.004	2.196	2.785	1.975

Tabela 5.7: Tempos de {segmentação + visualização} (em s) para cada passo da segmentação de “*brain*” nos experimentos 1 a 5: respectivamente (**IFT + S**), (**DIFT + S**), (**BDIFT + S-Borda**), (**DIFT + D-Borda + S-Borda**) e (**DIFT + A-Img**). A parte de baixo mostra a média dos tempos das iterações 2 a 8.

Comparando-se os métodos dos experimentos 2 e 3 aplicados ao conjunto de cenas de teste, o tempo total para a segmentação iterativa e visualização da borda necessita 8% mais tempo que a DIFT seguida do *splatting* da cena (vide o diagrama da figura 5.8). Na prática, isso corresponde a uma iteração de {segmentação + visualização} com duração média de 2,2 s contra 2 s no caso da cena "*brain*" (vide a tabela 5.7). Citando outro exemplo: com "*brain2*" (composto de 7 milhões de voxels) e com adição de 200 sementes em cada passo, a duração média da iteração torna-se 2,47 s e 2,76 s para os experimentos 2 e 3.

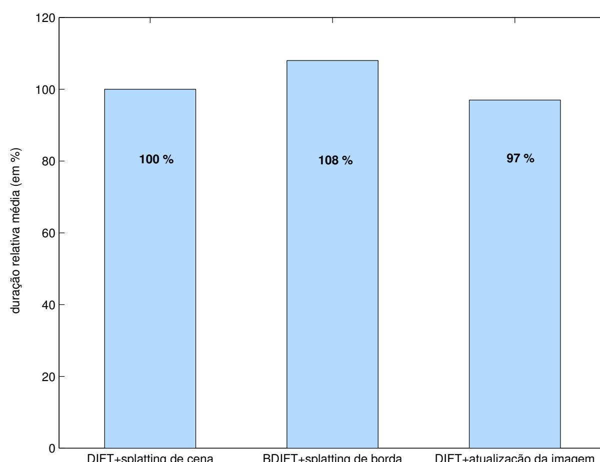


Figura 5.8: Duração relativa média de um passo de segmentação e visualização (passos posteriores ao primeiro) usando os experimentos 2, 3 e 5: (**DIFT + S**), (**BDIFT + S-Borda**) e (**DIFT + A-Img**).

Apesar do algoritmo *BorderDIFT* não ser muito eficiente à primeira vista, ele oferece a grande vantagem de ter a borda dos objetos armazenada em memória. Isto possibilita uma visualização rápida em outros ângulos de visualização, pois basta projetar os voxels da borda. O usuário, após uma segmentação, pode querer rotacionar o objeto para inspectar o resultado e, se for preciso, continuar a segmentação. Após cada rotação, a visualização terá o custo do *splatting* de borda, isto é, como já comentado, 58% do tempo do *splatting* da cena inteira. Em certos casos, este ganho pode ser significativo quanto à impressão de interatividade deixada ao usuário: No exemplo já citado da cena "*geometric_interp2*", o ganho permite reduzir o tempo de visualização de 1,29 s para 460 ms, possibilitando a passagem abaixo do limite de 1 s – fronteira entre "tempo real" e tempo dito "interativo" segundo Udupa e Herman (Udupa e Herman, 1991).

Comparemos agora as estratégias dos experimentos 2 e 5. As duas utilizam a DIFT para segmentar a cena iterativamente mas para visualizar cada resultado, a primeira efetua o clássico *splatting* da cena inteira e a segunda detecta apenas as mudanças entre cenas segmentadas e verifica se estas induzem mudanças na imagem. A economia de 24% feita por (**A-Img**) em relação ao *splatting* (**S**) reduz-se a 3% em média, quando comparadas as estratégias completas dos experimentos 2 e 5 (vide o diagrama da figura 5.8). Essa economia de tempo depende das mudanças ocorridas entre duas iterações. O método é adequado a refinamentos de segmentação, como é o caso quando se segmentam estruturas tais como os ventrículos laterais ou a medula numa IRM.

O método de atualização da imagem (experimento 5) pressupõe um ângulo de visualização fixo, o que pode ser limitativo para o usuário. Se o usuário quiser mudar de ponto de vista e rotacionar o objeto, ele terá que optar por projetar a cena inteira de novo, pois neste caso, não se pode construir a nova imagem a partir da atual. Para uma utilização prática do método da imagem incremental, é necessário mesclar este com o *splatting* de cena que possibilita então a mudança de ângulo de observação.

Enfim, comparando-se os métodos dos experimentos 3 e 5, conclui-se que as duas propostas dão imagens de qualidade equivalente ao método do experimento 2 que não aproveita resultados anteriores (*splatting* da cena total). Em relação ao tempo de processamento, a proposta da imagem incremental consegue ficar um pouco mais rápida que o método 3 (leva 90% do tempo do método 3). Porém, a proposta da borda incremental, apesar de ficar mais lenta, permite a visualização rápida dos objetos após rotações. Portanto, ela se revela uma boa estratégia para a segmentação interativa, possibilitando uma realimentação visual rápida assim como uma inspeção rápida dos objetos segmentados.

Capítulo 6

Conclusão

Para concluir essa dissertação, apresentamos neste capítulo uma síntese do trabalho efetuado em função dos objetivos lançados e mostrando os principais resultados. Acrescentamos também algumas sugestões para trabalhos futuros.

Os objetivos fixados foram alcançados, pois foram propostos dois novos métodos, onde segmentação e visualização tridimensional são integradas e possibilitam uma realimentação visual rápida adequada para o usuário durante a segmentação interativa de estruturas de interesse.

O algoritmo de atualização da imagem (**A-Img**) provê uma visualização incremental que aproveita o resultado da iteração anterior, uma vez o ângulo de visualização fixado. Ele atualiza apenas as zonas da imagem correspondentes às mudanças de segmentação ocorridas. Por sua vez, o segundo algoritmo proposto (**BDIFT**) armazena e atualiza a borda dos objetos de interesse a serem visualizados. Basta projetar a borda com um simples e rápido *splatting* para visualizar os objetos de interesse. No entanto, a detecção e atualização da borda são processos complexos, pois estão embutidos no algoritmo de segmentação.

Os dois métodos foram comparados a três outros e mostraram desempenhos aceitáveis. A visualização por (**A-Img**) leva 76% do tempo de um *splatting* de cena (**S**). Os métodos (**DIFT + A-Img**) e (**BDIFT + S-Borda**) são aproximadamente 4 vezes mais rápidos do que o método não iterativo via IFT e *splatting* da cena (**IFT + S**) e comparáveis à estratégia via DIFT e *splatting* (**DIFT + S**): Com efeito, eles demoram respectivamente 3% menos tempo e 8% mais tempo do que (**DIFT + S**). Entretanto, o método (**DIFT + A-Img**) consegue ser mais rápido que a estratégia (**DIFT + S**) quando não há muitas mudanças entre os resultados de segmentação de duas iterações consecutivas. Portanto, o método é adequado a refinamentos de segmentação, como é o caso quando os médicos estão interessados em segmentar pequenas estruturas tais como o cerebelo, os ventrículos laterais ou a medula numa IRM. Por outro lado, apesar de precisar de mais tempo que a (**DIFT + S**) para cada iteração (8% a mais), o algoritmo da (**BDIFT + S-Borda**) permite uma visualização rápida dos objetos após simples rotações, levando apenas 58% do tempo de (**S**). Portanto, é uma boa estratégia para a segmentação interativa, por possibilitar uma realimentação visual relativamente rápida assim como uma manipulação rápida dos objetos segmentados.

Um conjunto de 23 imagens variadas em tamanho, tipo e forma ajudou a testar os métodos comparados. Quatorze dessas imagens foram imagens médicas (IRM) provenientes do laboratório de NeuroImagem (FCM-Unicamp), de onde surgiu a motivação inicial de investigação de métodos interativos tridimensionais de segmentação e visualização.

Como trabalhos futuros, a otimização das estruturas de dados (a da borda por exemplo) poderá ser feita para agilizar o algoritmo de BDIFT. Uma melhoria do algoritmo de DIFT resolvendo o caso dos empates também é possível e deverá então ser avaliada a relação do aumento em complexidade e em tempo com a qualidade da segmentação efetuada. Enfim, uma interface gráfica poderá ser implementada para integrar os algoritmos de segmentação e visualização propostos.

Referências Bibliográficas

Artzy, E., Frieder, G. e Herman, G. T. (1981). The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm, *Computer Graphics and Image Processing* **15**: 1–24.

Avila, R., Sobierajski, L. e Kaufman, A. (1992). Towards a comprehensive volume visualization system, *Proceedings Visualization'92*, Boston (MA), EUA, pp. 13–20.

Beets, K. (2000). *3D Textures: Analyzed and Explained*. Tutorial da Beyond3D disponível em `http://www.beyond3d.com/articles/3dtextures/index1.php`.

Bergo, F. (2004). *Segmentação interativa de volumes baseada em regiões*, Tese de mestrado, Instituto de Computação - Universidade Estadual de Campinas, Campinas (SP), Brasil.

Bergo, F. e Falcão, A. (2003). Interactive 3D segmentation of brain MRI with differential watersheds, *Technical Report IC-03-16*, Instituto de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil. Disponível em `http://www.ic.unicamp.br/ic-tr`.

Beucher, S. e Meyer, F. (1993). The Morphological Approach to Segmentation: The Watershed Transform, *in* E. R. Dougherty (ed.), *Mathematical Morphology in Image Processing*, Marcel Dekker, New York (NY), EUA, chapter 12, pp. 433–481.

Bonilha, L., Rorden, C., Kobayashi, E., Montenegro, M. A., Guerreiro, M. M., Li, L. M. e Cendes, F. (2003). Voxel based morphometry study of partial epilepsies, *Arquivos de Neuro-Psiquiatria* **64**(Suppl.1): 93–97.

Cabral, B., Cam, N. e Foran, J. (1994). Accelerated volume rendering and tomographic reconstruction using texture mapping hardware, *1994 Symposium on Volume Visualization*, ACM, Washington (DC), EUA, pp. 91–97.

Carnevalle, A. D., Rondina, J. M., Kobayashi, E., Lotufo, R. A. e Cendes, F. (2003). Validation of a semi-automated system for MRI-based hippocampal volumetry in patients with temporal lobe epilepsy, *J. Epilepsy Clin Neurophysiol.* **9**(2): 97–104.

Cendes, F. (1997). *The use of proton magnetic resonance spectroscopic imaging and MRI volumetric measurements in the clinical investigation of partial epilepsies*, PhD dissertation, McGill University, Department Of Neurology And Neurosurgery, Montreal (Quebec), Canadá.

Cendes, F., Andermann, F., Gloor, P., Evans, A., Jones-Gotman, M., Watson, C., Melanson, D., Olivier, A., Peters, T., Lopes-Cendes, I. e Leroux, G. (1993). MRI volumetric measurements of amygdala and hippocampus in temporal lobe epilepsy, *Neurology* **43**(4): 719–725.

da Cunha, B. (2001). *Projeto de operadores de processamento de imagens usando a transformada imagem-floresta*, Tese de mestrado, Instituto de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil. Agência financiadora: FAPESP (Proc. No. 99/10100-3).

Danskin, J. e Hanrahan, P. (1992). Fast algorithms for volume ray tracing, *Proceedings of the 1992 Workshop on Volume Rendering*, Vol. 19, pp. 91–98.

Demarco, F. A., Ghizoni, E., Kobayashi, E. e Cendes, F. (2003). Cerebelar volume and long term use of phenytoin: MRI volumetric studies, *Seizure* **12**: 312–315.

Dougherty, E. (1992). *An Introduction to Morphological Image Processing*, (D.C. O'Shea Ed.), SPIE – The International Society for Optical Engineering, Bellingham (Washington), EUA.

Dougherty, E. e Lotufo, R. (2003). *Hands-on Morphological Image Processing*, SPIE – The International Society for Optical Engineering, Bellingham (Washington), EUA.

Ekoule, A., Peyrin, F. e Odet, C. (1991). A triangulation algorithm from arbitrary shaped multiple planar contours, *ACM Transactions on Graphics* **10**(2): 182–199.

Elvins, T. (1992). A survey of algorithms for volume visualization, *Computer Graphics* **26**(3): 194–201.

Falcão, A., Stolfi, J. e Lotufo, R. (2002). The image foresting transform: Theory, algorithms and applications, *Relatório Técnico 12*, Instituto de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil.

Falcão, A. e Udupa, J. (2000). A 3D generalization of user-steered live wire segmentation, *Medical Image Analysis* **4**(4): 389–402.

Falcão, A., Udupa, J. e Miyazawa, F. K. (2000). An ultra-fast user-steered image segmentation paradigm: Live-wire-on-the-fly, *IEEE Transactions on Medical Imaging* **19**(1): 55–62.

Falcão, A., Udupa, J., Samarasekera, S., Sharma, S., Hirsch, B. e Lotufo, R. (1998). User-steered image segmentation paradigms: Live wire and live lane, *Graphical Models and Image Processing* **60**(4): 233–260.

Foley, J., van Dam, A., Feiner, S. e Hughes, J. (1990). *Computer Graphics: Principles and Practice*, 2nd edn, Addison-Wesley, New York (NY), EUA.

Frieder, G., Gordon, D. e Reynolds, R. (1985). Back-to-front display of voxel-based objects, *IEEE Computer Graphics and Applications* **5**(1): 52–60.

Fuchs, H., Kedem, Z. e Uselton, S. (1977). Optimal surface reconstruction from planar contours, *Communications of the ACM* **20**(10): 693–702.

Gomes, J. e Velho, L. (1994). *Computação Gráfica: Imagem*, Série de Computação e Matemática, Instituto de Matemática Pura e Aplicada (IMPA) – Sociedade Brasileira de Matemática (SBM), Rio de Janeiro (RJ), Brasil. Capítulos 1 e 6.

Gonzalez, R. e Woods, R. (1993). *Digital Image Processing*, Addison-Wesley, New York (NY), EUA.

Gordon, D. e Udupa, J. K. (1989). Fast surface tracking in three-dimensional binary images, *Computer Vision, Graphics, and Image Processing* **45**: 196–214.

Gouraud, H. (1971). Continuous shading of curved surfaces, *IEEE Transaction on Computers* **20**(6): 623–629.

Goutsias, J. e Batman, S. (2000). Morphological methods for biomedical image analysis, *in* J. Fitzpatrick e M. Sonka (eds), *Handbook on Medical Imaging - Volume III: Progress in Medical Image Processing and Analysis*, SPIE Optical Engineering Press, Bellingham (Washington), EUA.

Haralick, R. e Shapiro, L. (1985). Survey: Image segmentation techniques, *Computer Vision, Graphics, and Image processing* **29**: 100–132.

Haralick, R., Sternberg, S. e Zhuang, X. (1987). Image analysis using mathematical morphology, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9**(4): 532–550.

Herman, G. T. e Webster, D. (1983). A topological proof of a surface tracking algorithm, *Computer Vision, Graphics, and Image Processing* **23**: 162–177.

Kass, M., Witkin, A. e Terzopoulos, D. (1988). Snakes: Active contour models, *International Journal of Computer Vision* **1**: 321–331.

Kaufman, A. (1996). Volume visualization, *ACM Computing Survey* **28**(1): 165–167.

Keppel, E. (1975). Approximating complex surfaces by triangulation of contour lines, *IBM Journal of Research and Development* **19**(1): 2–11.

Kindlmann, G. e Durkin, J. W. (1998). Semi-automatic generation of transfer functions for direct volume rendering, *IEEE Symposium on Volume Visualization*, Research Triangle Park (NC), EUA, pp. 79–86. Disponível em `citeseer.nj.nec.com/kindlmann98semiautomatic.html`.

Kniss, J., Kindlmann, G. e Hansen, C. (2002). Multidimensional transfer functions for interactive volume rendering, *IEEE Transactions on Visualization and Computer Graphics* **8**(3): 270–285.

Lacroute, P. (1995). *Fast volume rendering using a shear-warp factorization of the viewing transformation*, Tese de Doutorado, Dept. of Electrical Engineering and Computer Science – Stanford University, Stanford (CA), EUA.

Lacroute, P. e Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation, *Computer Graphics* **28**(Annual Conference Series): 451–458.

Levoy, M. (1990). Efficient ray tracing of volume data, *ACM Transactions on Graphics* **9**(3): 245–261.

Lima, C. (1999). Geração de imagens com texturas utilizando OpenGL, *Revista do CCEI* **3**(3): 29–37. Disponível em `http://www.urcamp.tche.br/site/ccei/revista/numero3.pdf`.

Lorensen, W. e Cline, H. (1987). Marching cubes: A high resolution 3D surface construction algorithm, *Computer Graphics* **21**(4): 163–169.

Lotufo, R. e Falcão, A. (2000). The Ordered Queue and the Optimality of the Watershed Approaches, *5th International Symposium on Mathematical Morphology*, Kluwer Academic, Palo Alto (CA), EUA, pp. 341–350.

Lürig, C. e Ertl, T. (1998). Hierarchical volume analysis and visualization based on morphological operators, *Proceedings of the conference on Visualization'98*, IEEE Computer Society Press, Research Triangle Park (NC), EUA, pp. 335–341.

Machado, R. (2002). *Adesso: Ambiente para desenvolvimento de software científico*, Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil.

Machiraju, R. e Yagel, R. (1993). Efficient feed-forward volume rendering techniques for vector and parallel processors, *Proceedings of Supercomputing'93*, Portland (Oregon), EUA, pp. 699–708.

Manohar, S. (1999). Advances in volume graphics, *Computers & Graphics* **23**: 73–84.

Maravilla, K. (1978). Computer reconstructed sagittal and coronal computed tomography head scans: clinical applications, *Journal Comput. Assist. Tomogr.* **2**(2): 120–123.

Meyer, F. e Beucher, S. (1990). Morphological segmentation, *Journal of Visual Communication and Image Processing* **1**(1): 21–46.

Morgenthaler, D. G. e Rosenfeld, A. (1981). Surfaces in three-dimensional digital images, *Information and Control* **51**: 227–247.

Mueller, K. e Crawfis, R. (1998). Eliminating popping artifacts in sheet buffer-based splatting, *Proceedings of the Visualization'98 conference*, IEEE Computer Society Press, Research Triangle Park (NC), EUA, pp. 239–245.

Mueller, K., Möller, T. e Crawfis, R. (1999). Splatting without the blur, *Proceedings of the Visualization'99 conference*, IEEE Computer Society Press, San Francisco (CA), EUA, pp. 363–370.

Nisbet, P., Gedroyc, W. e Rankin, S. (1987). *Case studies in diagnostic imaging: Film interpretation for postgraduate examinations*, Springer-Verlag, New York (NY), EUA.

Ntasis, E., Maniatis, T. e Nikita, K. (2002). Fourier volume rendering for real time preview of digital reconstructed radiographs: a web-based implementation, *Computerized Medical Imaging and Graphics* **26**(1): 1–8.

Owen, G. (1999). *Volume Visualization and Rendering*. The ACM SIGGRAPH Education Committee. Tutorial do projeto HyperVis – Teaching Scientific Visualization Using Hypermedia. Disponível em `http://www.siggraph.org/education/materials/HyperVis/vistech/volume/volume.htm`.

Phong, B. T. (1975). Illumination for computer generated pictures, *Communications of the ACM* **18**(6): 311–317.

Rhodes, M., Jr., W. G. e Azzawi, V. (1980). Extracting oblique planes from serial ct sections, *Journal Comput. Assist. Tomogr.* **4**: 649–657.

Rocha, L. (2002). *Shell rendering com fatoração shear-warp*, Tese de mestrado, Instituto de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil.

Rondina, J. (2001). *Segmentação interativa do ventrículo esquerdo em sequências de imagens de ressonância magnética (Cine MR)*, Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas, Campinas (SP), Brasil.

Rondina, J., Lotufo, R., Gutierrez, M. e Rebelo, M. (2000). An interactive segmentation system based on watershed approach applied to magnetic resonance images, *World Congress on Medical Physics and Biomedical Engineering*, Chicago (IL), EUA.

Rothman, S., Dobben, G., Rhodes, M., Glenn, W. e Azzawi, V. (1984). Computed tomography of the spine: Curved coronal reformation from serial images, *Radiology* **150**(1): 180–185.

Saha, P., Udupa, J. e Odhner, D. (2000). Scale-based fuzzy connected image segmentation: Theory, algorithms, and validation, *Computer Vision and Image Understanding* **77**(9): 145–174.

Schmidt, A., Gattass, M. e Carvalho, P. (2000). Combined 3D visualization of volume data and polygonal models using a shear-warp algorithm, *Computers & Graphics* **24**(4): 583–601.

Shung, K., Smith, M. e Tsui, B. (1992). *Principles of Medical Imaging*, Academic Press, San Diego (CA), EUA.

Soille, P. (1999). *Morphological Image Analysis: Principles and Applications*, Springer Verlag, New York (NY), EUA.

Stytz, M. R., Frieder, G. e Frieder, O. (1991). Three-dimensional medical imaging: algorithms and computer systems, *ACM Computing Surveys (CSUR)* **23**(4): 421–499.

Talairach, J. e Tournoux, P. (1988). *Co-Planar Stereotaxic Atlas of the Human Brain*, Thieme Medical Publishers, New York (NY), EUA.

Tan, A. (2001). *OpenGL - Texture Mapping*. Tutorial da Beyond3D disponível em `http://www.beyond3d.com/articles/opengltexmap/index1.php`.

Totsuka, T. e Levoy, M. (1993). Frequency domain volume rendering, *Proceedings of the 20th Annual Conference on Computer Graphics*, Vol. 20, ACM, pp. 271–278.

Udupa, J. e Herman, G. (1991). *3D Imaging in Medicine*, CRC Press, Boca Raton (Florida), EUA.

Udupa, J. K. (1987). A unified theory of objects and their boundaries in multi-dimensional digital images, *Proceedings of Computer Assisted Radiology CAR'87*, Springer-Verlag, Berlin, Alemanha, pp. 779–784.

Udupa, J. K., Srihari, S. N. e Herman, G. T. (1982). Boundary detection in multidimensions, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4**(1): 41–50.

Udupa, J. e Odhner, D. (1993). Shell rendering, *IEEE Computer Graphics and Applications* **13**(6): 58–67.

Westenberg, M. e Roerdink, J. (2000). X-ray volume rendering by hierarchical wavelet splatting, *Proceedings of the 15th International Conference on Pattern Recognition (ICPR'2000)*, Barcelona, Espanha, pp. 163–166.

Westover, L. (1990). Footprint evaluation for volume rendering, *Computer Graphics SIGGRAPH'90* **24**(4): 367–376.

Wolff, S. (2001). *Computer aided analysis of MR brain images*, MSc dissertation, Informatics and Mathematical Modelling – Technical University of Denmark, Kgs. Lyngby, Dinamarca.

Woo, M., Neider, J., Davis, T. e Shreiner, D. (1999). *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, 3rd edn, Addison-Wesley Publishing Company, Reading (MA), EUA. Version 1.2.

Yagel, R. (1997). Volume viewing and shading, mini-curso do *Siggraph'97*, Los Angeles (CA), EUA. Disponível em `http://www.merl.com/people/pfister/courses/Bonn2000/YagelViewingAndShading-Slides.pdf`.

Yagel, R., Ebert, D., Scott, J. e Kurzion, Y. (1995). Grouping volume renderers for enhanced visualization in computational fluid dynamics, *IEEE Transactions on Visualization and Computer Graphics* **1**(2): 117–132.

Yagel, R. e Kaufman, A. (1992). Template-based volume viewing, *Computer Graphics Forum* **11**(3): 153–167.

Apêndice A

Convenções adotadas no ambiente de visualização

A.1 Definições

O **voxel** (*volume element*) é um elemento de volume, geralmente um paralelepípedo, situado na cena, que recebe um nível de cinza ou tom de cinza representado por um número inteiro que corresponde à integração de uma propriedade física da cena no espaço por ele ocupado.

Cena, objeto: a cena é constituída do conjunto dos voxels armazenados em memória, independentemente do valor ou rótulo de cada um. Ela tem ($sizex \times sizey \times sizez$) voxels e constitui um paralelepípedo. Quando esta é segmentada, seus voxels são particionados em vários subconjuntos que constituem um ou vários objetos e o fundo.

Plano de projeção ou **plano de visualização** (*projection plane, view-plane*) são sinônimos do plano da imagem.

A **normal ao plano de projeção** (*view-plane normal VPN*) é o vetor unitário perpendicular ao plano de projeção e coincide geralmente com a direção de visualização. Ela sai da posição do observador e aponta para o objeto.

A **direção do observador** (*viewer direction*) está no sentido oposto à direção de visualização. Sai do objeto e aponta para o observador. Tem como sinônimo a direção de projeção. Ela é representada pelo vetor unitário $\vec{V}$. A **direção da fonte de luz** sai do objeto e aponta para a luz. Ela é representada pelo vetor unitário $\vec{L}$.

O **espaço voxel** ou **sistema de coordenadas voxel SCV** (*voxel coordinate system*) é montado a partir das dimensões da cena e definido sobre $\aleph^3$. Um canto da cena é a origem. Todos os voxels pertencendo à cena têm, como coordenadas, inteiros positivos (vide a figura A.1 no final do apêndice).

O **índice *raster*** de um voxel ou índice linear indica a ordem na qual os voxels são armazenados em

memória e é relacionado a estas coordenadas voxel pelas relações seguintes:

$$ind_{raster} = x + y \,.\, sizex + z \,.\, sizex \,.\, sizey$$

e

$$\begin{cases} x = r \div sizex \\ y = r \; / \; sizex \\ z = ind_{raster} \; / \; (sizex \,.\, sizey) \\ r = ind_{raster} \div (sizex \,.\, sizey) \end{cases}$$

Obs.: O símbolo ÷ dá o resto da divisão inteira (euclidiana), enquanto o símbolo / representa o quociente da divisão inteira (euclidiana).

Espaço objeto ou **sistema de coordenadas objeto SCO** (*object coordinate system*) ou sistema de coordenadas do mundo (*world coordinate system*): Este sistema admite tanto coordenadas não inteiras como negativas (definido sobre $\Re^3$). A origem deste sistema está no centro da cena (ponto do meio).

Espaço imagem ou **sistema de coordenadas imagem SCI** (*image coordinate system*) tem como eixos principais $\vec{u}$, $\vec{v}$ e $\vec{d}$. O eixo horizontal é $\vec{u}$, o vertical é $\vec{v}$ e o terceiro aponta na direção de visualização: $\vec{d} = \vec{u} \times \vec{v}$. A origem desse sistema fica no centro da imagem.

Sistema de coordenadas pixel (SCP): é, em relação ao espaço imagem, o que é o sistema de coordenadas voxel para o espaço objeto. É definido apenas sobre $\aleph^3$ restrito a $[0; u_{max}] \times [0; v_{max}]$. A origem desse sistema fica no canto esquerdo superior.

Matriz de projeção, **matriz de visualização**, matriz de **transformação direta** e **inversa**:

São combinações de rotações e translações. A matriz de projeção, chamada também de transformação direta (M_{proj} ou M_{direta}), transforma um ponto *(x,y,z)*^{T} da cena expresso no sistema de coordenadas objeto, num outro ponto *(u,v,d)*^{T} no espaço imagem. Desprezando a coordenada segundo o terceiro eixo d, obtêm-se as coordenadas dos pixels correspondentes à projeção ortogonal do ponto da cena sobre o plano da imagem.

$$\begin{pmatrix} u \\ v \\ d \\ 1 \end{pmatrix}_{sci} = M_{proj} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}_{sco} \quad \text{ou simplesmente:} \quad \begin{pmatrix} u \\ v \end{pmatrix}_{sci} = M_{proj(2\times4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}_{sco}$$

onde $M_{proj(2\times4)} = \begin{bmatrix} M_{proj}[0,0] & M_{proj}[0,1] & M_{proj}[0,2] & M_{proj}[0,3] \\ M_{proj}[1,0] & M_{proj}[1,1] & M_{proj}[1,2] & M_{proj}[1,3] \end{bmatrix}$

A matriz de visualização ou transformação inversa (M_{view} ou $M_{inversa}$) permite passar do espaço imagem para o espaço objeto. Ela é utilizada nos métodos com precisão de imagem (como o *raycasting*).

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}_{sco} = M_{view} \begin{pmatrix} u \\ v \\ d \\ 1 \end{pmatrix}_{sci} \quad \text{com } M_{view} = M_{proj}^{-1}.$$

A.2 Convenções adotadas nos programas

Considera-se um sistema de eixos ortonormais (x, y, z) que respeitam a regra da mão direita. Adota-se a convenção de notação em coluna para os vetores.

A.2.1 Coordenadas homogêneas

Um ponto P que tem as coordenadas $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ no espaço 3D tem como coordenadas homogêneas $\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$ onde: $\frac{X}{W} = x$; $\frac{Y}{W} = y$; $\frac{Z}{W} = z$; $W \in \Re$ e $W \neq 0$. Para simplificar usa-se $W = 1$: $P : \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$

A.2.2 Matrizes de transformação

Uma matriz de transformação M transforma um ponto P de coordenadas $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ num ponto P' de coordenadas $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ tais que: $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$, onde M é uma matriz 4×4: $\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$

1. Matrizes de rotação de ângulo:
θ em volta do eixo x, φ em volta do eixo y e ψ em volta do eixo z, respectivamente.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\varphi) = \begin{bmatrix} cos\varphi & 0 & sen\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -sen\varphi & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\psi) = \begin{bmatrix} cos\psi & -sen\psi & 0 & 0 \\ sen\psi & cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Matriz de translação de um vetor $\vec{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$: $T(\vec{t}) = T \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

A.2.3 Composição de transformações

Considerando-se as matrizes de transformação M_1 e M_2 das transformações T_1 e T_2 efetuadas nessa ordem, podemos simplificar as duas transformações numa única T, cuja matriz é: $M = M_2 \cdot M_1$. Cuidado: $R_x(\theta) \cdot R_y(\varphi) \neq R_y(\varphi) \cdot R_x(\theta)$. A ordem em qual as transformações elementares são aplicadas importa.

Nos algoritmos de visualização implementados, usa-se tanto a matriz de transformação direta (aplicada aos pontos do objeto nos métodos de projeção de voxels) quanto a matriz de transformação inversa (aplicada aos raios jogados do plano de visualização).

Vale a pena ressaltar como se obtém a matriz de transformação inversa a partir da matriz de transformação direta. Por exemplo, rotacionar a cena (objetos) em seguida, de um ângulo θ em volta do eixo x, de um ângulo φ em volta do eixo y e de um ângulo ψ em volta do eixo z é equivalente a rotacionar o plano de visualização (posição do observador) na ordem contrária com ângulos negativos: rotação de um ângulo $-\psi$ em volta do eixo z, rotação de um ângulo $-\varphi$ em volta do eixo y e rotação de um ângulo $-\theta$ em volta do eixo x.

Para ver o interior do volume, usa-se um plano de corte. Na prática, desloca-se o plano de visualização (projeção) de modo a ele cair dentro do objeto. Por isso a matriz de visualização comportará rotações em volta dos 3 eixos assim como uma translação para posicionar o plano de corte ou plano de visualização.

Para evitar que haja corte na visualização da cena, utiliza-se um plano de visualização quadrado com as seguintes dimensões: $altura = largura = \sqrt{xsize^2 + ysize^2 + zsize^2}$ ao visualizar uma cena de dimensões: $xzise \times ysize \times zsize$. Por isso também, geralmente, posiciona-se o plano de visualização numa distância d_z do centro da cena tal que: $d_z = \frac{1}{2}\sqrt{xsize^2 + ysize^2 + zsize^2}$. Com efeito, a diagonal principal da cena (que passa pelo centro da cena e dois vértices opostos) é a maior dimensão possível na cena.

Assim temos a matriz de transformação direta:

$$M_{proj} = M_{direta} = T(0, 0, -d_z) \cdot R_z(\psi) \cdot R_y(\varphi) \cdot R_x(\theta)$$

para rotacionar em x, y e z e transladar a cena em seguida de uma distância $-d_z$ no eixo z.

A matriz de transformação inversa correspondente é:

$$M_{view} = M_{inversa} = M_{direta}^{-1} = R_x(-\theta) \cdot R_y(-\varphi) \cdot R_z(-\psi) \cdot T(0, 0, +d_z)$$

para transladar o plano de visualização de uma distância $+d_z$ no eixo z e rotacionar em seguida em z, y e x com os ângulos respectivos negativos.

$$M_{direta} = \begin{bmatrix} cos\,\psi\; cos\,\varphi & -sen\,\psi\; cos\,\theta\; + cos\,\psi\; sen\,\varphi\; sen\,\theta & sen\,\psi\; sen\,\theta\; + cos\,\psi\; sen\,\varphi\; cos\,\theta & 0 \\ sen\,\psi\; cos\,\varphi & cos\,\psi\; cos\,\theta\; + sen\,\psi\; sen\,\varphi\; sen\,\theta & -cos\,\psi\; sen\,\theta\; + sen\,\psi\; sen\,\varphi\; cos\,\theta & 0 \\ -sen\,\varphi & cos\,\varphi\; sen\,\theta & cos\,\varphi\; cos\,\theta & -d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Na matriz de transformação direta M_{direta}, as três primeiras colunas são iguais às coordenadas dos três eixos principais do plano de visualização (u, v e d), no sistema de coordenada de objeto (mundo). A

quarta coluna contém, por sua vez, informações sobre as translações da origem do plano de visualização em relação à origem do sistema mundo.

$$M_{inversa} = \begin{bmatrix} cos\,\psi\; cos\,\varphi & sen\,\psi\; cos\,\varphi & -sen\,\varphi & -d_z sen\,\varphi \\ -sen\,\psi\; cos\,\theta\; + cos\,\psi\; sen\,\varphi\; sen\,\theta & cos\,\psi\; cos\,\theta\; + sen\,\psi\; sen\,\varphi\; sen\,\theta & cos\,\varphi\; sen\,\theta & d_z cos\,\varphi\; sen\,\theta \\ sen\,\psi\; sen\,\theta\; + cos\,\psi\; sen\,\varphi\; cos\,\theta & -cos\,\psi\; sen\,\theta\; + sen\,\psi\; sen\,\varphi\; cos\,\theta & cos\,\varphi\; cos\,\theta & d_z cos\,\varphi\; cos\,\theta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obs.: Nas convenções adotadas, é chamado o ângulo em volta do eixo x de *tilt* (inclinação) enquanto o ângulo em volta do eixo y é chamado *spin* (rotação).

A.3 Outras convenções possíveis

Considera-se um sistema de eixos ortonormais (x, y, z) que respeitam a regra da mão direita. Adotando-se a convenção de notação em linha para os vetores.

A.3.1 Coordenadas homogêneas

Um ponto P que tem as coordenadas (x, y, z) no espaço 3D tem como coordenadas homogêneas (X, Y, Z, W) onde: $\frac{X}{W} = x$; $\frac{Y}{W} = y$; $\frac{Z}{W} = z$; $W \in \Re$ e $W \neq 0$. Para simplificar usa-se $W = 1$: $P : \begin{pmatrix} x & y & z & 1 \end{pmatrix}$

A.3.2 Matrizes de transformação

Uma matriz de transformação M transforma um ponto P de coordenadas $\begin{pmatrix} x & y & z \end{pmatrix}$ num ponto P' de coordenadas $\begin{pmatrix} x' & y' & z' \end{pmatrix}$ tais que: $\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot M$, onde M é uma matriz 4×4:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

1. Matrizes de rotação de ângulo:
θ em volta do eixo x, φ em volta do eixo y e ψ em volta do eixo z, respectivamente.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sen\theta & 0 \\ 0 & -sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\varphi) = \begin{bmatrix} cos\varphi & 0 & -sen\varphi & 0 \\ 0 & 1 & 0 & 0 \\ sen\varphi & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\psi) = \begin{bmatrix} cos\psi & sen\psi & 0 & 0 \\ -sen\psi & cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Matriz de translação de um vetor $\vec{t} = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}$: $T(\vec{t}) = T((t_x, t_y, t_z)) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$

Essas matrizes correspondem às transpostas das matrizes utilizadas na convenção vetor-coluna.

A.3.3 Composição de transformações

Considerando-se as matrizes de transformação M_1 e M_2 das transformações T_1 e T_2 efetuadas nessa ordem, podemos simplificar as duas transformações numa única T, cuja matriz é: $M = M_1 \cdot M_2$.

A.4 Ambiente de visualização

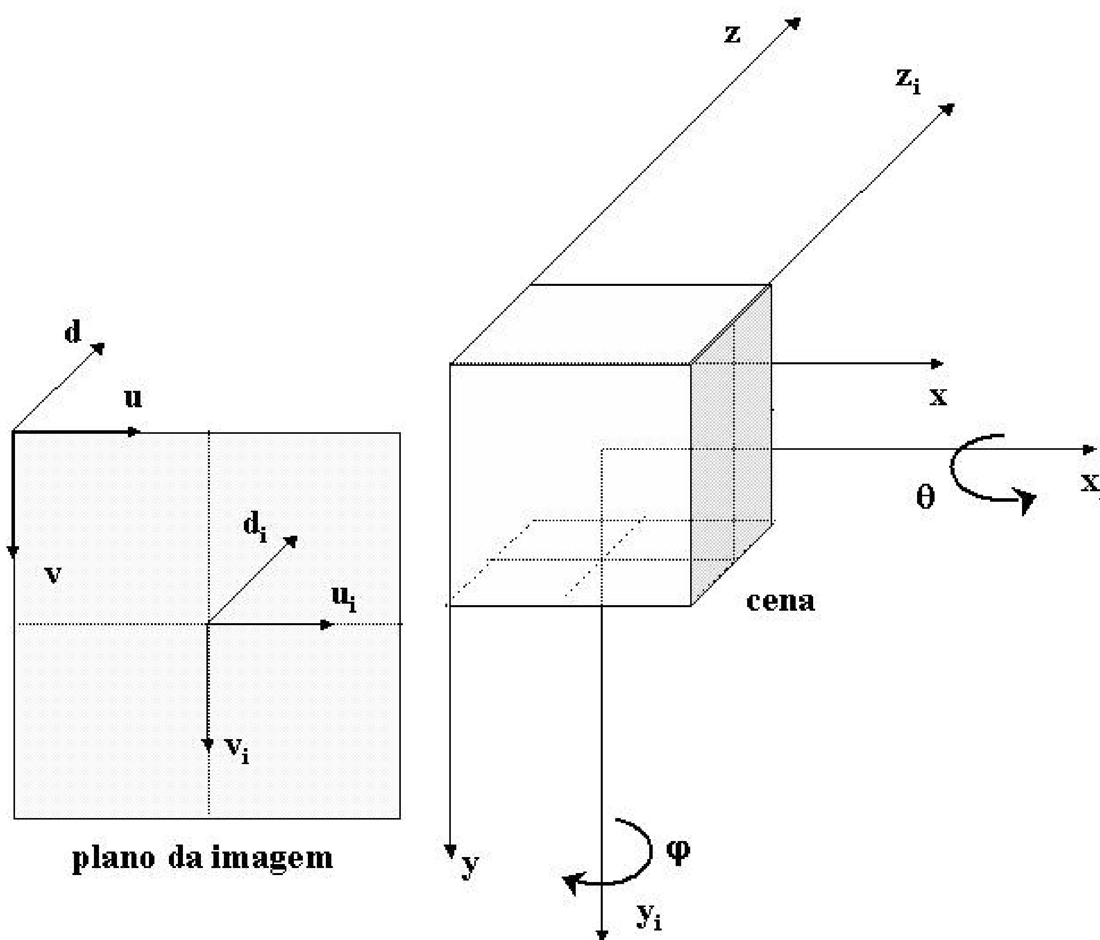


Figura A.1: Ambiente de visualização: convenções e sistemas adotados. Sistema de coordenadas de objeto ou do "mundo" (x_i, y_i, z_i). Sistema de coordenadas dos voxels (x, y, z). Sistema de coordenadas da imagem (u_i, v_i, d_i). Sistema de coordenadas dos pixels (u, v).

Apêndice B

Algoritmo de Bresenham

B.1 Introdução

O algoritmo de Bresenham 3D permite traçar uma reta no espaço 3D discreto. Isto é, ele designa quais os voxels do espaço 3D discreto correspondem melhor à reta contínua. No nosso caso, a reta corresponde à trajetória do raio. Ele inicia-se no ponto de partida do raio correspondente a um pixel particular do plano da imagem e atravessa a cena até cruzar um objeto de interesse. Ele tem como direção a direção de visualização dada por:

$$\vec{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} M_{inversa}[0,2] \\ M_{inversa}[1,2] \\ M_{inversa}[2,2] \end{pmatrix} = \begin{pmatrix} M_{view}[0,2] \\ M_{view}[1,2] \\ M_{view}[2,2] \end{pmatrix}$$

Como os raios variam em comprimento, dependendo do ponto de encontro com um objeto, escolhe-se um comprimento padrão c: inteiro e fixo. Por exemplo, 10 vezes o tamanho da grande diagonal da cena (esta diagonal é o comprimento máximo a ser percorrido através da cena):

$$c = 10\,diagonal = 10\,\sqrt{size_x^2 + size_y^2 + size_z^2}$$

Essa reta pode ser decomposta segundo os três eixos principais em: $\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} = round \begin{pmatrix} c.n_x \\ c.n_y \\ c.n_z \end{pmatrix}$. Chamemos esses três componentes de "componentes do raio".

O algoritmo de Bresenham é incremental, ou seja, ele constrói a reta voxel por voxel, passo a passo. Para saber qual é o voxel que vai seguir o voxel corrente, basta adicionar um incremento nas coordenadas do voxel corrente. Esse incremento é unitário, e positivo ou negativo segundo a direção de propagação do raio. Os incrementos são:

$$\Delta_x = \begin{cases} +1 & \text{se } d_x \geq 0 \\ -1 & \text{se } d_x < 0 \end{cases} \qquad \Delta_y = \begin{cases} +1 & \text{se } d_y \geq 0 \\ -1 & \text{se } d_y < 0 \end{cases} \qquad \Delta_z = \begin{cases} +1 & \text{se } d_z \geq 0 \\ -1 & \text{se } d_z < 0 \end{cases}$$

Daqui para frente, o algoritmo varia segundo o eixo principal. Lembramos que o eixo principal é definido como sendo o eixo segundo o qual o vetor de visualização (raio) tem o valor absoluto máximo:

$$e_{princ} = \{i \mid |n_i| = max\left(|n_x|, |n_y|, |n_z|\right)\} \Leftrightarrow e_{princ} = \{i \mid |d_i| = max\left(|d_x|, |d_y|, |d_z|\right)\}$$

Pegamos o caso de z ser o eixo principal (caso que corresponde a olhar a cena de frente) e dx, dy, dz positivos. Estamos interessados em traçar um raio que parte do pixel (u, v). A cada passo a coordenada do eixo principal (z) será incrementada de $\Delta_z = +1$. No entanto, as coordenadas segundo os dois eixos não principais (x e y) serão ou não incrementados dos passos respectivos: $\Delta_x = +1$ e $\Delta_y = +1$. A cada passo, temos portanto a escolha entre quatro possibilidades de voxel para representar a reta no passo seguinte:

Ponto corrente: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Pontos seguintes potenciais: $\begin{pmatrix} x \\ y \\ z+1 \end{pmatrix}$ ou $\begin{pmatrix} x+1 \\ y \\ z+1 \end{pmatrix}$ ou $\begin{pmatrix} x \\ y+1 \\ z+1 \end{pmatrix}$ ou $\begin{pmatrix} x+1 \\ y+1 \\ z+1 \end{pmatrix}$.

Para saber quais coordenadas escolher, separa-se o problema 3D em dois problemas 2D. Nos planos (x, z) e (y, z), observam-se as projeções da reta 3D. O algoritmo de Bresenham 3D é então equivalente ao algoritmo de Bresenham 2D aplicado às duas projeções da reta 3D nos planos contendo o eixo principal e outro secundário. Assim, no algoritmo 2D, as escolhas serão:

No plano (x, z). Ponto corrente: $\begin{pmatrix} x \\ z \end{pmatrix}$. Pontos seguintes potenciais: $\begin{pmatrix} x \\ z+1 \end{pmatrix}$ ou $\begin{pmatrix} x+1 \\ z+1 \end{pmatrix}$.

No plano (y, z). Ponto corrente: $\begin{pmatrix} y \\ z \end{pmatrix}$. Pontos seguintes potenciais: $\begin{pmatrix} y \\ z+1 \end{pmatrix}$ ou $\begin{pmatrix} y+1 \\ z+1 \end{pmatrix}$.

B.2 Algoritmo de Bresenham em 2D

O algoritmo do "ponto meio" (*midpoint*), formulação particular do algoritmo de Bresenham, utiliza uma variável de decisão "$erro$" relacionada à diferença entre o ponto seguinte teórico (ponto no espaço contínuo) e o "ponto meio" M. Este é situado no meio dos dois pontos seguintes potenciais: $\begin{pmatrix} x+\frac{1}{2} \\ z+1 \end{pmatrix}$ no caso do plano (x, z). Se o erro for positivo, significa que a reta passa acima do ponto meio M. Portanto, o pixel que melhor representa a reta, é o pixel de cima: $\begin{pmatrix} x+1 \\ z+1 \end{pmatrix}$. Se, pelo contrário, esse erro for negativo, a reta passa abaixo do ponto meio M. Portanto, o pixel de baixo $\begin{pmatrix} x \\ z+1 \end{pmatrix}$ melhor representa a reta.

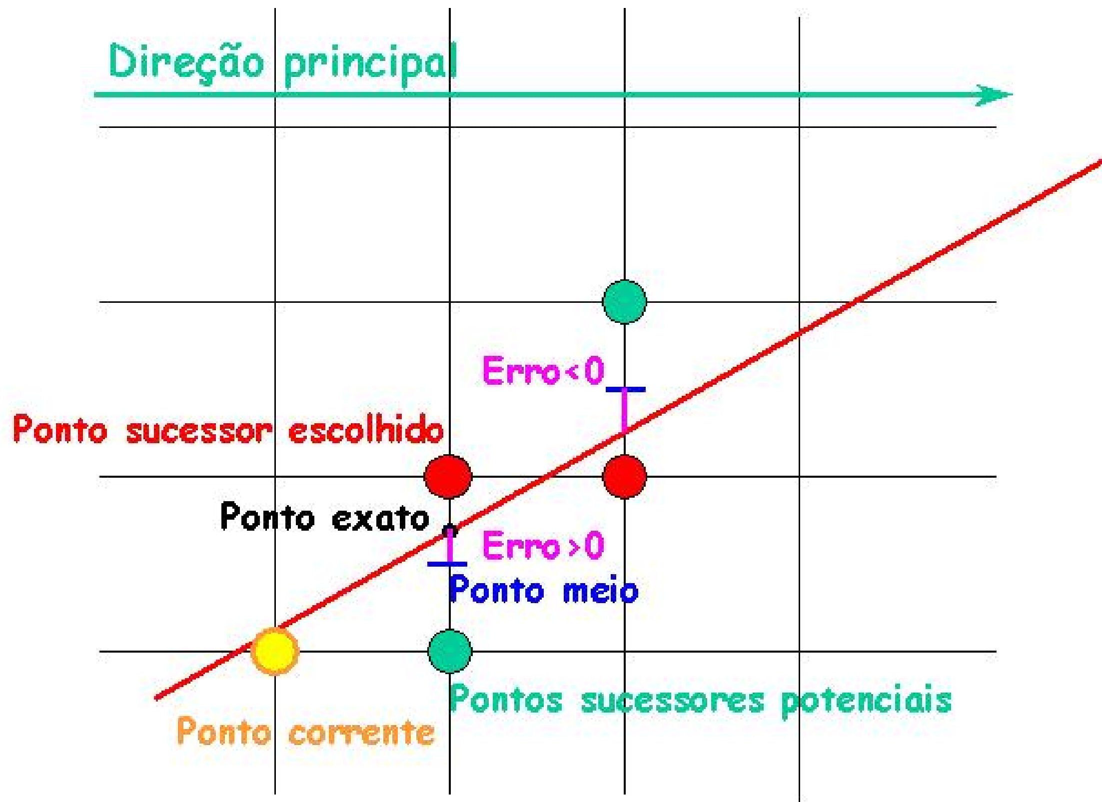


Figura B.1: Algoritmo de Bresenham 2D.

O algoritmo do ponto meio (2D) para traçar uma reta entre os pontos $\begin{pmatrix} x_0 \\ z_0 \end{pmatrix}$ e $\begin{pmatrix} x_1 \\ z_1 \end{pmatrix}$ é o seguinte (z é eixo principal):

1. Inicialização

$d_x \leftarrow x_1 - x_0;$

$d_z \leftarrow z_1 - z_0;$

$erro \leftarrow 2\,d_x - d_z;$

$x \leftarrow x_0;$

$z \leftarrow z_0;$

$Escreva \begin{pmatrix} x \\ z \end{pmatrix};$

2. Traçado incremental

$Enquanto\;\; z < z_1$

$Se\;\; erro > 0$

$erro \leftarrow erro - 2\,d_z;$

$x \leftarrow x + 1;$

$erro \leftarrow erro + 2\,d_x;$

$z \leftarrow z + 1;$

$Escreva \begin{pmatrix} x \\ z \end{pmatrix};$

B.3 Algoritmo de Bresenham em 3D

Passando ao caso 3D, o algoritmo do "ponto meio" torna-se:

1. Inicialização

$d_x \leftarrow x_1 - x_0;$

$d_y \leftarrow y_1 - y_0;$

$d_z \leftarrow z_1 - z_0;$

$erro_1 \leftarrow 2\,d_x - d_z;$

$erro_2 \leftarrow 2\,d_y - d_z;$

$x \leftarrow x_0;$

$y \leftarrow y_0;$

$z \leftarrow z_0;$

$Escreva \begin{pmatrix} x \\ y \\ z \end{pmatrix};$

2. Traçado incremental

$Enquanto\;\; z < z_1$

$\quad Se\;\; erro_1 > 0$

$\quad\quad erro_1 \leftarrow erro_1 - 2\,d_z;$

$\quad\quad x \leftarrow x + 1;$

$\quad Se\;\; erro_2 > 0$

$\quad\quad erro_2 \leftarrow erro_2 - 2\,d_z;$

$\quad\quad y \leftarrow y + 1;$

$\quad erro_1 \leftarrow erro_1 + 2\,d_x;$

$\quad erro_2 \leftarrow erro_2 + 2\,d_y;$

$\quad z \leftarrow z + 1;$

$\quad Escreva \begin{pmatrix} x \\ y \\ z \end{pmatrix};$

Para poder tratar qualquer tipo de reta (não somente com z^+ como direção principal), tem que levar em conta os valores absolutos dos componentes da reta. O algoritmo genérico é obtido fazendo as seguintes modificações:

1. Inicialização: O cálculo dos erros iniciais nas duas direções não principais torna-se o seguinte.

Se o eixo principal for z:

$$erro_1 \leftarrow 2\,|d_y| - |d_z|$$

$$erro_2 \leftarrow 2\,|d_x| - |d_z|$$

Se o eixo principal for x:

$$erro_1 \leftarrow 2\,|d_y| - |d_x|$$

$$erro_2 \leftarrow 2\,|d_z| - |d_x|$$

Se o eixo principal for y:

$$erro_1 \leftarrow 2\,|d_x| - |d_y|$$

$$erro_2 \leftarrow 2\,|d_z| - |d_y|$$

2. Traçado incremental: A parte da atualização das coordenadas do voxel corrente torna-se:

Se o eixo principal for z:

$Se\ \ erro_1 > 0,$
    $y \leftarrow y + \Delta_y;$
    $erro_1 \leftarrow erro_1 - 2\ |d_z|;$
$Se\ \ erro_2 > 0,$
    $x \leftarrow x + \Delta_x;$
    $erro_2 \leftarrow erro_2 - 2\ |d_z|;$
$erro_1 \leftarrow erro_1 + 2\ |d_y|;$
$erro_2 \leftarrow erro_2 + 2\ |d_x|;$
$z \leftarrow z + \Delta_z;$

Se o eixo principal for x:

$Se\ \ erro_1 > 0,$
    $y \leftarrow y + \Delta_y;$
    $erro_1 \leftarrow erro_1 - 2\ |d_x|;$
$Se\ \ erro_2 > 0,$
    $z \leftarrow z + \Delta_z;$
    $erro_2 \leftarrow erro_2 - 2\ |d_x|;$
$erro_1 \leftarrow erro_1 + 2\ |d_y|;$
$erro_2 \leftarrow erro_2 + 2\ |d_z|;$
$x \leftarrow x + \Delta_x;$

Se o eixo principal for y:

$Se\ \ erro_1 > 0,$
    $x \leftarrow x + \Delta_x;$
    $erro_1 \leftarrow erro_1 - 2\ |d_y|;$
$Se\ \ erro_2 > 0,$
    $z \leftarrow z + \Delta_z;$
    $erro_2 \leftarrow erro_2 - 2\ |d_y|;$
$erro_1 \leftarrow erro_1 + 2\ |d_x|;$
$erro_2 \leftarrow erro_2 + 2\ |d_z|;$
$y \leftarrow y + \Delta_y;$

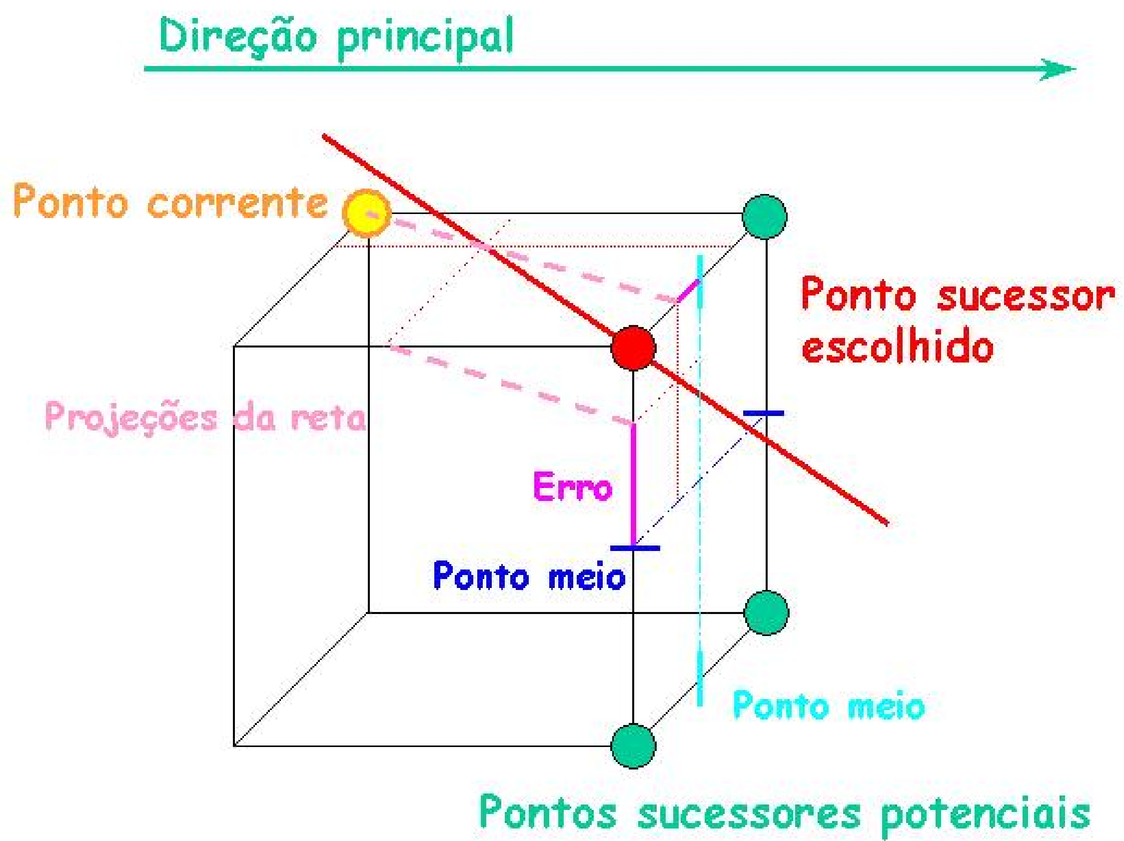


Figura B.2: Algoritmo de Bresenham 3D.

Apêndice C

Avaliação da normal

C.1 Avaliação de normal no espaço imagem

Esse método avalia as normais em cada pixel a partir do gradiente das distâncias contidas no *d-buffer* só. Com efeito, a estimação da normal ao objeto a partir do *d-buffer* utiliza o método da diferença central para pixels que não estão nas bordas da imagem e o método da diferença progressiva para pixels de borda.

Para um pixel (u, v) que não está numa borda da imagem, um vetor $\vec{N}$ perpendicular ao objeto é dado por:

$$\vec{N} = \begin{pmatrix} N_u \\ N_v \\ N_d \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[d(u+1,v) - d(u-1,v)] \\ \frac{1}{2}[d(u,v+1) - d(u,v-1)] \\ -1 \end{pmatrix}$$

onde $d(u, v)$ corresponde à distância contida no *d-buffer* para o pixel (u, v).

Normalizando esse vetor perpendicular, obtém-se a normal ao objeto:

$$\vec{n} = \begin{pmatrix} n_u \\ n_v \\ n_d \end{pmatrix} = \begin{pmatrix} \frac{N_u}{\|\vec{N}\|} \\ \frac{N_v}{\|\vec{N}\|} \\ \frac{N_d}{\|\vec{N}\|} \end{pmatrix}, \quad \text{com } \left\|\vec{N}\right\| = \sqrt{N_u^2 + N_v^2 + N_d^2} \quad \text{se } \left\|\vec{N}\right\| \neq 0 \text{ e}$$

$$\vec{n} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{caso contrário.}$$

Para um pixel (u, v) de borda ou de canto, isto é, se pelo menos uma dessas condições estiver satisfeita: $u = u_{min} = 0$ ou $u = u_{max} = size_u - 1$ ou $v = v_{min} = 0$ ou $v = v_{max} = size_v - 1$; um dos dois (no caso de pixel de borda) ou os dois primeiros componentes de $\vec{N}$ (no caso de pixel no canto da imagem) mudam:

$$N_u = d(u+1,v) - d(u,v) \quad \text{se } u = u_{min}, \qquad N_u = d(u,v) - d(u-1,v) \quad \text{se } u = u_{max};$$

$$N_v = d(u,v+1) - d(u,v) \quad \text{se } v = v_{min}, \qquad N_v = d(u,v) - d(u,v-1) \quad \text{se } v = v_{max}.$$

Observação: Nota-se que essa estimativa de normal é expressa no sistema de coordenadas imagem (SCI).

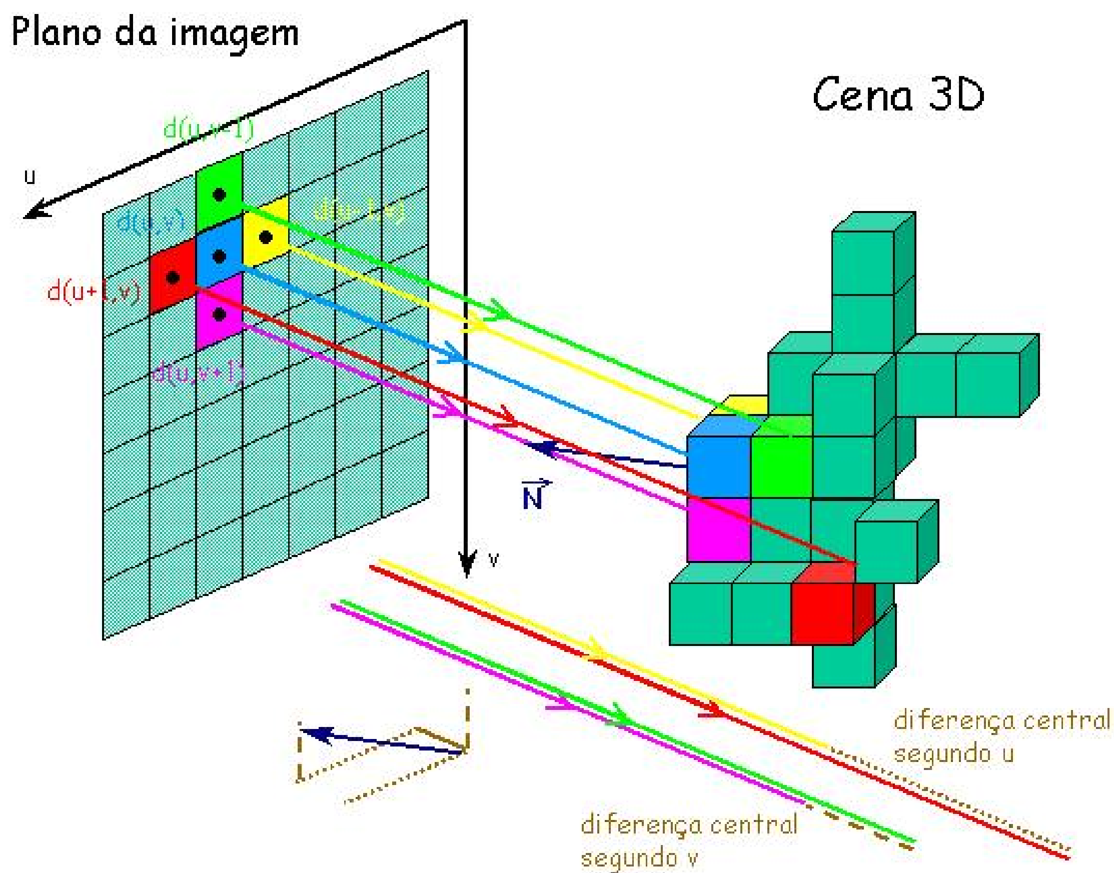


Figura C.1: Estimação da normal a partir do *d-buffer* (espaço imagem).

C.2 Avaliação de normal no espaço voxel

A estimativa da normal é avaliada em cada voxel da cena e é baseada no gradiente dos níveis de cinza no espaço dos voxels. A normal num voxel depende só dos valores de cinza dos voxels vizinhos e independe desses fazerem parte ou não do objeto de interesse. Como para a estimativa a partir do *d-buffer*, utilizam-se dois métodos para avaliar esse gradiente, segundo a posição do voxel na cena: voxel de borda ou não.

Para um voxel (x, y, z) que não está numa borda da cena, utiliza-se o método da diferença central:

$$\vec{N} = \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[v(x+1,y,z) - v(x-1,y,z)] \\ \frac{1}{2}[v(x,y+1,z) - v(x,y-1,z)] \\ \frac{1}{2}[v(x,y,z+1) - v(x,y,z-1)] \end{pmatrix}$$

onde $v(x, y, z)$ corresponde ao nível de cinza do voxel (x, y, z).

Normalizando esse vetor perpendicular, obtém-se a normal ao objeto:

$$\vec{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} \frac{N_x}{\|\vec{N}\|} \\ \frac{N_y}{\|\vec{N}\|} \\ \frac{N_z}{\|\vec{N}\|} \end{pmatrix}, \quad \text{com } \left\|\vec{N}\right\| = \sqrt{N_x^2 + N_y^2 + N_z^2} \quad \text{se } \left\|\vec{N}\right\| \neq 0 \text{ e}$$

$$\vec{n} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{caso contrário.}$$

Para um voxel (x, y, z) de borda (face, aresta ou canto), isto é, se pelo menos uma dessas condições estiver satisfeita: $x = x_{min} = 0$ ou $x = x_{max} = size_x - 1$ ou $y = y_{min} = 0$ ou $y = y_{max} = size_y - 1$ ou $z = z_{min} = 0$ ou $z = z_{max} = size_z - 1$; um dos três (no caso de voxel de face), dois dos três (no caso de voxel de aresta) ou os três componentes de $\vec{N}$ (no caso de voxel de canto da cena) mudam:

$$N_x = v(x+1,y,z) - v(x,y,z) \text{ se } x = x_{min}, \quad N_x = v(x,y,z) - v(x-1,y,z) \text{ se } x = x_{max};$$

$$N_y = v(x,y+1,z) - v(x,y,z) \text{ se } y = y_{min}, \quad N_y = v(x,y,z) - v(x,y-1,z) \text{ se } y = y_{max};$$

$$N_z = v(x,y,z+1) - v(x,y,z) \text{ se } z = z_{min}, \quad N_z = v(x,y,z) - v(x,y,z-1) \text{ se } z = z_{max}.$$

É importante ressaltar que essa estimativa é expressa no sistema de coordenadas objeto SCO. Além disso, observa-se que as normais podem ser calculadas com a cena em níveis de cinza, não segmentada. Desse jeito, essa etapa pode ser vista como um pré-processamento, durante o qual todas as normais aos voxels são calculadas uma vez por todas, independentemente da(s) futura(s) segmentação(ões). Uma vez as normais calculadas na cena toda, extraem-se as normais dos voxels visíveis, ou seja, dos que são visíveis no *d-buffer*, após *raycasting* ou projeção de voxel.

Nota-se que segundo as equações acima, a direção da normal é do escuro para o claro, devido ao fato de representar cinzas claros com um valor maior do que para cinzas escuros. Geralmente considera-se que as superficies de interesse são claras e o fundo é mais escuro (cf. Raios-X). Como nas nossas convenções de tonalização, a normal aponta para o exterior do objeto, as normais têm que apontar do claro (objeto) para o escuro (fundo externo). Por isso, as normais serão negadas para poder ser utilizadas adequadamente no processo de tonalização. Supõe-se também que voxels pertencentes a um mesmo objeto têm níveis de cinza semelhantes. Nessas condições, o gradiente dos níveis de cinza representa uma boa estimativa da normal. Ele será nulo entre voxels de um mesmo objeto e será de valor absoluto elevado entre um voxel pertencente ao objeto e outro ao fundo.

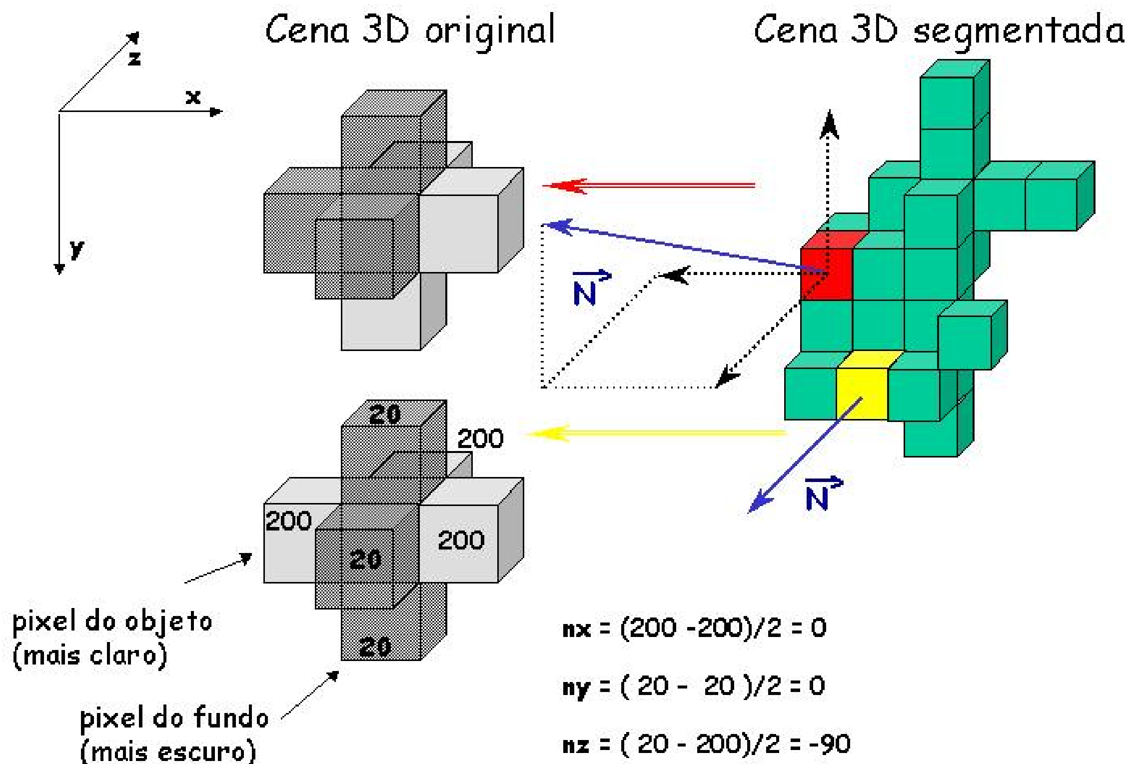


Figura C.2: Estimação da normal a partir dos níveis de cinza (espaço voxel).

C.3 Avaliação da normal no espaço objeto a partir do *d-buffer*

Esse método avalia as normais em cada pixel do *d-buffer* obtido após *raycasting* ou projeção de voxel. A avaliação é feita a partir das coordenadas dos voxels correspondentes aos vizinhos-8 (xadrez) no *d-buffer* de cada pixel e a ele mesmo. Oito normais são calculadas a partir de vetores entre voxel de origem e voxel vizinho. A média das oito estimativas constitui a normal final. O fato de pegar a média de 8 normais suaviza a interpolação da normal e garante uma tonalização mais suave.

Algoritmo:

Seja V o mapa dos voxels atingidos ou projetados. Considera-se um pixel (u, v) que não está numa borda ou num canto da imagem. Seja $V(u, v)$ o índice raster do voxel correspondente a esse pixel (por projeção ou *raycasting*). Chamamos esse voxel de "voxel origem" V_0: $V_0 = V(u, v)$. Ele tem como coordenadas no sistema das coordenadas voxel: (x_0, y_0, z_0). Tendo as dimensões da cena, $size_x$, $size_y$ e $size_z$, podem-se calcular as coordenadas do voxel $V(u, v)$:

$$x_0 = r_0 \div size_x, \quad y_0 = r_0/size_x, \quad z_0 = V_0/(size_x\, size_y), \quad \text{com } r_0 = V_0 \div (size_x\, size_y).$$

Sejam V_1, V_2, ..., V_8 os voxels correspondentes aos oito pixels vizinhos-8 de (u, v): $(u, v - 1)$, $(u + 1, v - 1)$, $(u + 1, v)$, $(u + 1, v + 1)$, $(u, v + 1)$, $(u - 1, v + 1)$, $(u - 1, v)$ e $(u - 1, v - 1)$. Esses vizinhos podem também ser chamados respectivamente de vizinhos N (norte), NE (nordeste), E (leste),

SE (sudeste), S (sul), SW (sudoeste), W (oeste) e NW (noroeste) e têm como coordenadas respectivas: (x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_8, y_8, z_8). Para $i = 1, 2, ..., 8$, formamos os vetores $\vec{W_i}$ indo do voxel origem V_0 para o voxel V_i:

$$\begin{bmatrix} x_i - x_0 \\ y_i - y_0 \\ z_i - z_0 \end{bmatrix}$$

Calculam-se os produtos vetoriais $\vec{P_i}$ entre os vetores $\vec{W_{i \oplus 1}}$ e $\vec{W_i}$:

$$\vec{P_i} = \vec{W_{i \oplus 1}} \times \vec{W_i} = \begin{pmatrix} P_{ix} \\ P_{iy} \\ P_{iz} \end{pmatrix}$$

Obs.: O símbolo $\oplus$ representa uma adição periódica ou cíclica. $i \oplus 1 = \begin{cases} i+1 & \text{se } i \in [1, 2, ..., 7] \\ 1 & \text{se } i = 8 \end{cases}$

Normalizam-se os vetores $\vec{P_i}$:

$$\vec{n_i} = \begin{pmatrix} \frac{P_{ix}}{\|\vec{P_i}\|} \\ \frac{P_{iy}}{\|\vec{P_i}\|} \\ \frac{P_{iz}}{\|\vec{P_i}\|} \end{pmatrix}, \quad \text{com } \left\|\vec{P_i}\right\| = \sqrt{P_{ix}^2 + P_{iy}^2 + P_{iz}^2} \quad \text{se } \left\|\vec{P_i}\right\| \neq 0 \quad \text{e } \vec{n_i} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{ caso contrário.}$$

Adicionam-se as oito normais $\vec{n_i}$: $\vec{N} = \sum_{i=1}^{8} \vec{n_i}$.

Normaliza-se o resultado para obter a estimativa da normal para o pixel (u, v): $\vec{n} = \frac{\vec{N}}{\|\vec{N}\|}$.

Observa-se que existem casos nos quais o pixel (u, v) não tem oito vizinhos: Para os pixels na borda da imagem, há cinco vizinhos, enquanto que para os dos cantos da imagem, há apenas três vizinhos. Nesses casos, o algoritmo aplica-se só com um conjunto reduzido de vizinhos. Quando um pixel de origem não tem nenhum voxel correspondente (o raio atingiu o fundo da cena sem cruzar o objeto ou nenhum voxel do objeto projetou-se neste pixel), não se calcula a normal, pois o pixel não é de objeto.

Lembramos, finalmente, que a estimativa da normal é expressa no sistema de coordenadas objeto SCO. Apesar dessa estimação ser de ordem de objeto, ela utiliza uma vizinhança de ordem de imagem.

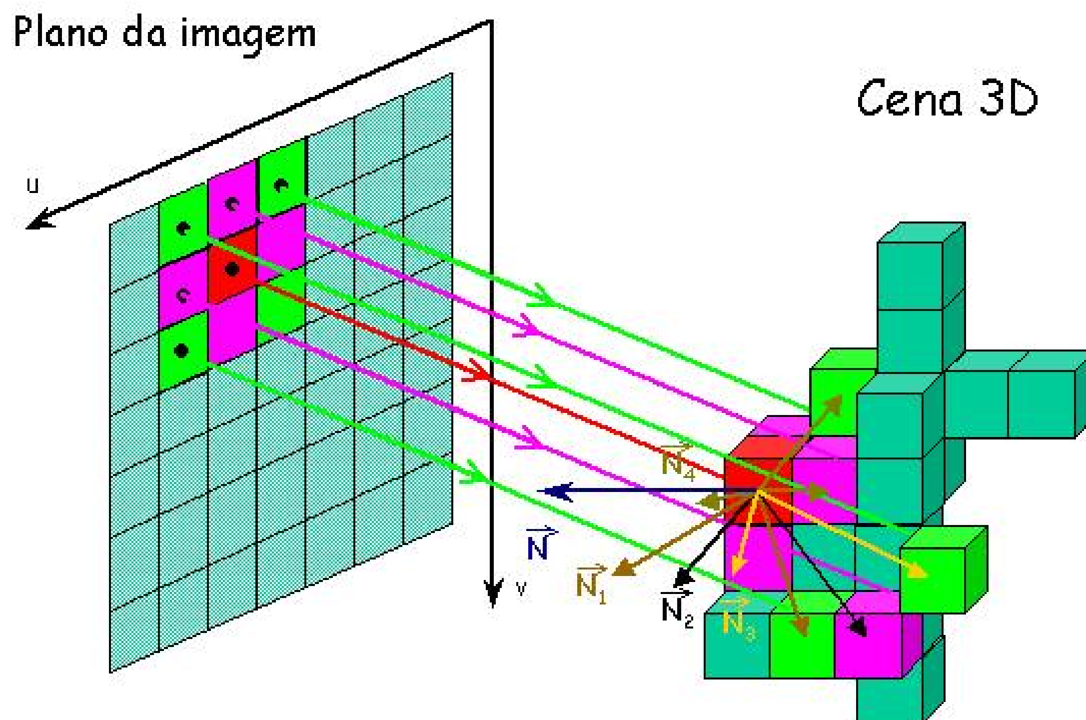


Figura C.3: Estimação da normal a partir das coordenadas de voxels (espaço objeto).

Apêndice D

Fatoração *shear-warp*

D.1 Generalidades

O *shear-warp rendering* é um método de *rendering* de volume que consiste na fatoração da matriz de projeção (ou de visualização, no caso do *raycasting*):

$$M_{proj} = M_{warp} \cdot M_{shear}$$

O *rendering* clássico efetua uma multiplicação matricial para calcular a projeção dos voxels na imagem (*voxel projection*) ou a localização dos raios na cena (*raycasting*). A idéia da fatoração *shear-warp* é processar os dados da cena em duas etapas computacionalmente baratas, ao invés de transformar os dados de uma vez só, usando a matriz clássica de projeção (ou de visualização). Essas etapas são as seguintes (vide figura D.1):

1.
   - Aplica-se uma transformação ***shear*** (cisalhamento em duas direções) nas fatias da cena. Essa transformação desloca as fatias, conservando-as paralelas entre si e de forma a torná-las perpendiculares aos raios de visualização, estes permanecendo paralelos entre si também.
   - Projetam-se as fatias (cisalhadas) para obter uma imagem intermediária distorcida.
2. Aplica-se uma transformação ***warp*** (distorcer) 2D para tirar as distorções e obter a imagem esperada.

A vantagem dessa fatoração é que as linhas de voxels no volume são alinhadas às linhas de pixels na imagem intermediária. Isso permite varrer a imagem intermediária e a cena em sincronismo. Esse acesso sincronizado reduz bastante o tempo do *rendering*. Além disso, a projeção efetuada para obter a imagem intermediária é ortogonal, portanto, não necessita nenhum cálculo adicional. Enfim, a transformação *warp* é pouco custosa, pois apenas bidimensional.

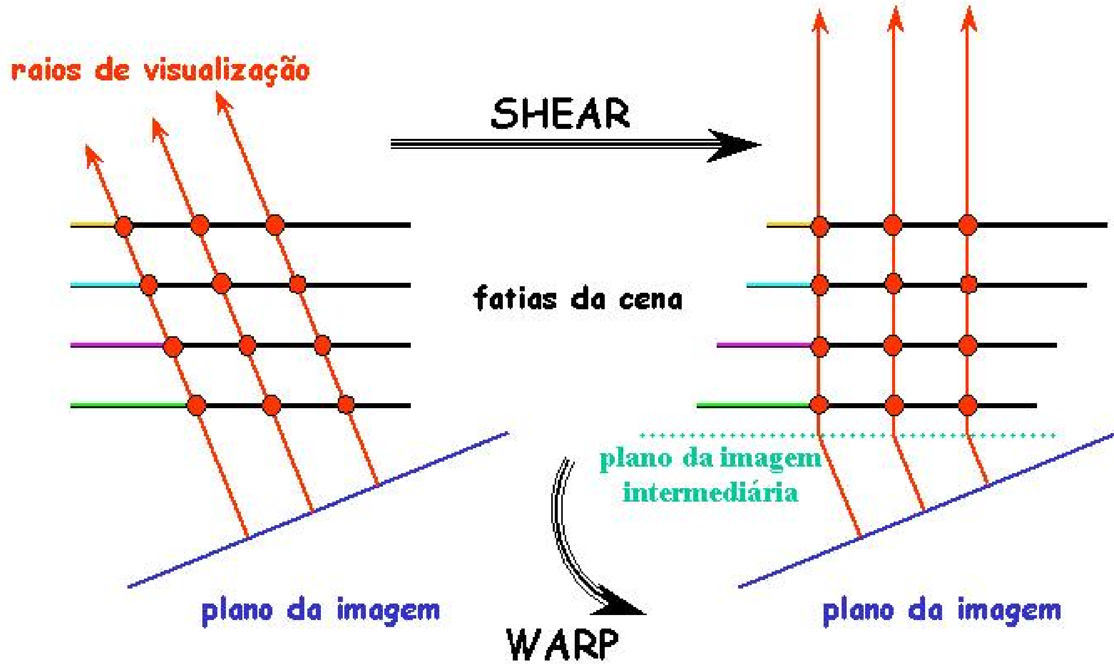


Figura D.1: Fatoração *shear-warp*.

D.2 Matrizes *shear* e *warp*

D.2.1 No caso de um eixo principal em z

A matriz de cisalhamento e a matriz correspondente à transformação inversa são da forma:

$$M_{shear} = \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_{shear}^{-1} = \begin{bmatrix} 1 & 0 & -s_i & 0 \\ 0 & 1 & -s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A matriz *warp* é deduzida da relação de fatoração:

$$M_{proj} = M_{warp} \cdot M_{shear} \Longleftrightarrow M_{warp} = M_{proj} \cdot M_{shear}^{-1}$$

com $M_{proj} = M_{direta}$ e $M_{proj}^{-1} = M_{inversa} = M_{view}$ (essas matrizes são dadas no Apêndice A).

D.2.2 Generalização para qualquer eixo principal

Para saber qual é o eixo principal, basta olhar para o vetor $\vec{n}$ normal ao plano da imagem (vetor de visualização) e calcular os valores absolutos dos 3 componentes. O eixo correspondente ao componente de valor absoluto máximo é chamado de eixo principal: $e_{princ} = \{i \mid |n_i| = max\left(|n_x|, |n_y|, |n_z|\right)\}$

Segundo o eixo principal, os fatores de *shear* variam:

Se for x: $s_i = n_y/n_x$ e $s_j = n_z/n_x$

Se for y: $s_i = -n_z/n_y$ e $s_j = n_x/n_y$

Se for z: $s_i = n_x/n_z$ e $s_j = n_y/n_z$

$$\text{com } \vec{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} M_{inversa}[0,2] \\ M_{inversa}[1,2] \\ M_{inversa}[2,2] \end{pmatrix}$$

Geralmente se usa uma matriz de permutação para levar em conta todos os casos de eixo principal. Com o eixo principal em x:

$$M_{perm} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_{perm}^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Com o eixo principal em y:

$$M_{perm} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_{perm}^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Com o eixo principal em z:

$$M_{perm} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_{perm}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Também é usada uma matriz de translação para transladar todas as fatias depois do *shear*, a fim de centralizá-las (passagem do sistema de coordenadas voxel ao sistema de coordenadas objeto).

$$M_{transl} = \begin{bmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_{transl}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_i \\ 0 & 1 & 0 & -t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

com $t_i = -\frac{1}{2}\, s_i\, size_{princ}$ e $t_j = -\frac{1}{2}\, s_j\, size_{princ}$, onde $size_{princ}$ representa o tamanho da cena na direção principal de visualização.

Resumindo, para levar em conta todas as orientações possíveis, a fatoração da matriz de projeção (transformação direta) é a seguinte: $M_{proj} = M_{warp} \cdot M_{transl} \cdot M_{shear} \cdot M_{perm}$

Todas as matrizes são conhecidas exceto a matriz de transformação *warp* facilmente computada:

$$M_{warp} = M_{proj} \cdot M_{perm}^{-1} \cdot M_{shear}^{-1} \cdot M_{transl}^{-1}$$

Logo, a matriz *warp* é da forma:

$$M_{warp} = \begin{bmatrix} A & B & ... & -t_iA - t_jB \\ C & D & ... & -t_iC - t_jD \\ ... & ... & ... & ... \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Antes de aplicar a transformação *warp*, temos uma imagem intermediária constituída de pontos de coordenadas:

$$\begin{pmatrix} x_{interm} \\ y_{interm} \\ z_{interm} = 0 \\ 1 \end{pmatrix}$$

Com efeito, a coordenada em z (eixo principal na verdade) pode ser zerada, pois se efetuou uma projeção ortogonal logo após da transformação *shear*. Desse jeito, é possível fazer uma economia de cálculo para a transformação *warp*, tirando a terceira coluna da matriz M_{warp}. Também se sabe que das 3 coordenadas dos pontos obtidos após transformação *warp*, apenas as duas primeiras (u e v) são relevantes, pois o plano da imagem é 2D. Por isso, eliminamos também a terceira linha da matriz M_{warp}. Resta então apenas a parte útil da matriz *warp*: uma matriz 3×3 (transformação 2D).

$$M_{warp2D} = \begin{bmatrix} A & B & -t_iA - t_jB \\ C & D & -t_iC - t_jD \\ 0 & 0 & 1 \end{bmatrix}$$

As coordenadas dos pixels atingidos são dadas pela equação seguinte:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = M_{warp2D} \begin{pmatrix} x_{interm} \\ y_{interm} \\ 1 \end{pmatrix} \Longleftrightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A(x_{interm} - t_i) + B(y_{interm} - t_j) \\ C(x_{interm} - t_i) + D(y_{interm} - t_j) \end{pmatrix}$$

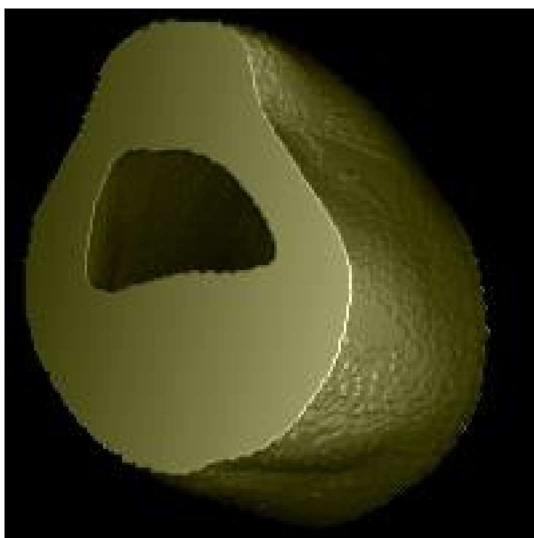
Os valores de A, B, C e D dependem do eixo principal.
Caso o eixo principal for x: $A = M_{proj}[0,1]$, $B = M_{proj}[0,2]$, $C = M_{proj}[1,1]$, $D = M_{proj}[1,2]$.
Caso o eixo principal for y: $A = M_{proj}[0,2]$, $B = M_{proj}[0,0]$, $C = M_{proj}[1,2]$, $D = M_{proj}[1,0]$.
Caso o eixo principal for z: $A = M_{proj}[0,0]$, $B = M_{proj}[0,1]$, $C = M_{proj}[1,0]$, $D = M_{proj}[1,1]$.

Apêndice E

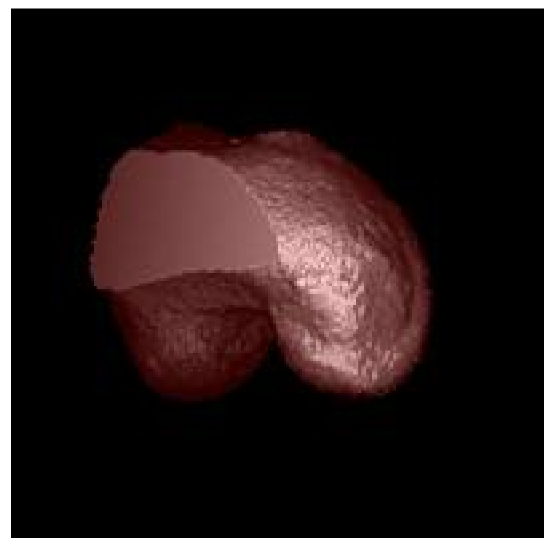
Imagens obtidas

Apresentamos nesta seção algumas imagens obtidas durante os testes.

- As figuras E.1 e E.2 apresentam algumas cenas de teste segmentadas e visualizadas por *splatting*.
- A figura E.3 mostra as imagens obtidas, aplicando-se várias técnicas de *rendering* na cena sintética "*geometric*" segmentada, para uma direção de visualização não ortogonal às direções principais do espaço da cena.
- Enfim, para entender os conceitos de roteiro e segmentação interativa, seguem as segmentações passo a passo das cenas "*brain*" e "*brain3*", com imagens vistas segundo as direções principais do espaço da cena (vide respectivamente as figuras E.4, E.5, E.6 e E.7, E.8).

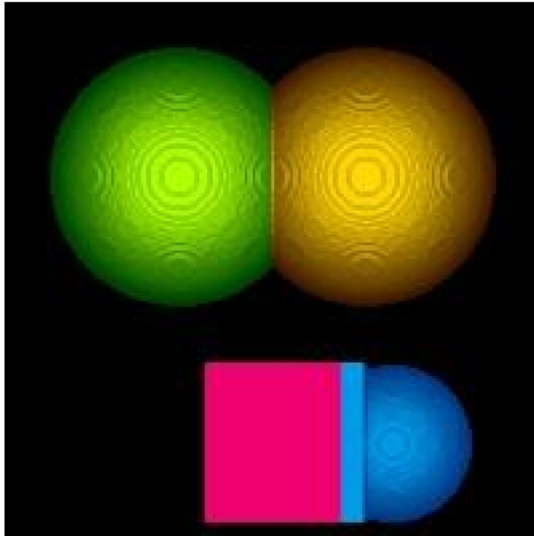


(a)

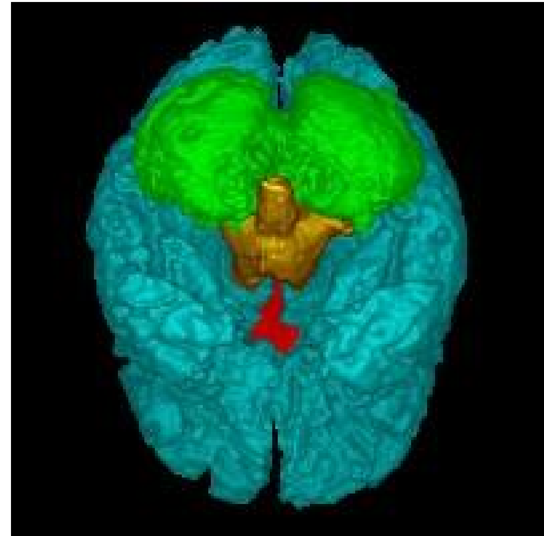


(b)

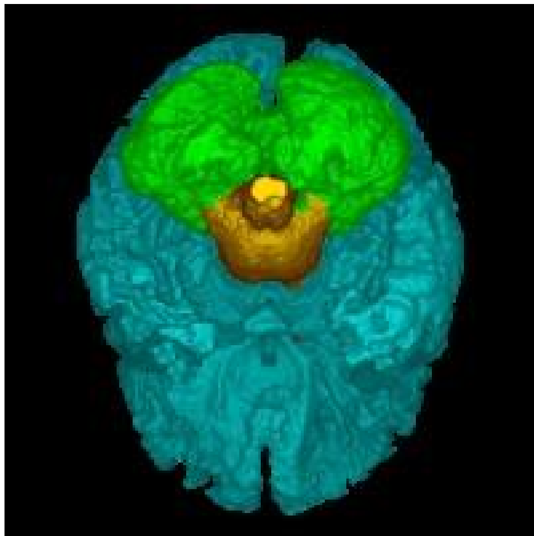
Figura E.1: Cena de teste "*knee*" segmentada: parte externa (pele) (**a**) e parte interna (osso) (**b**).



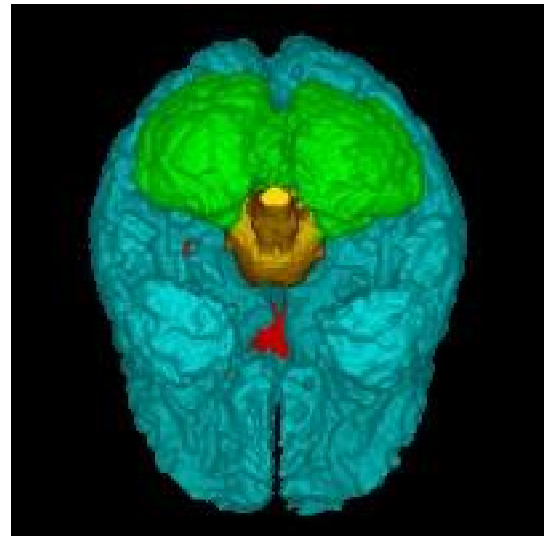
(a)



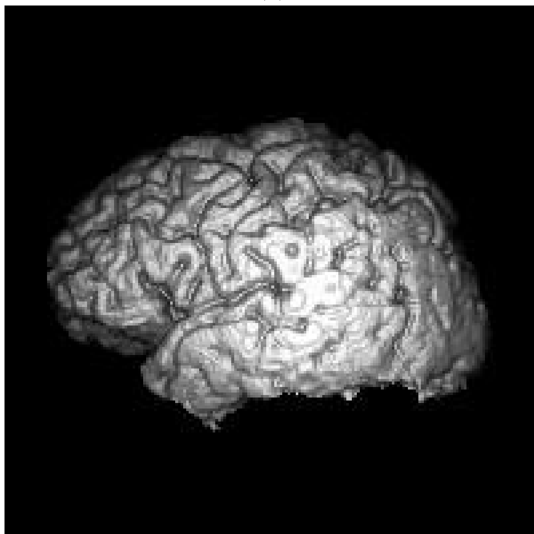
(b)



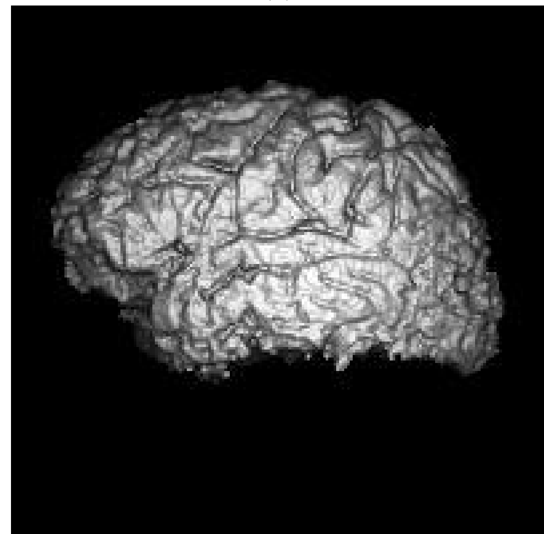
(c)



(d)

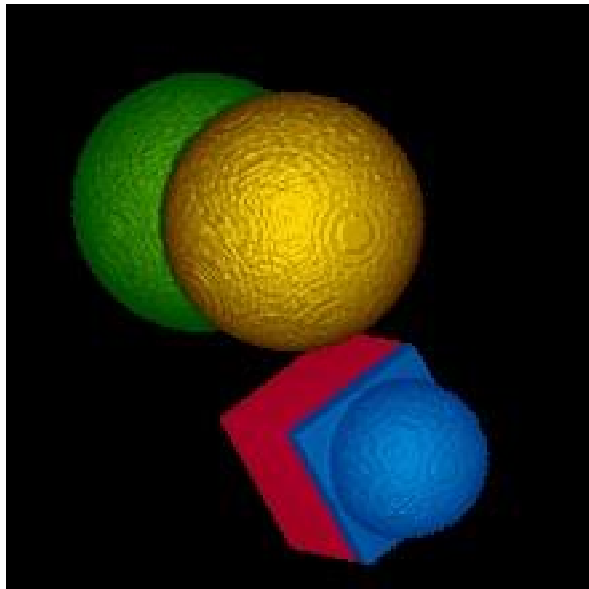


(e)

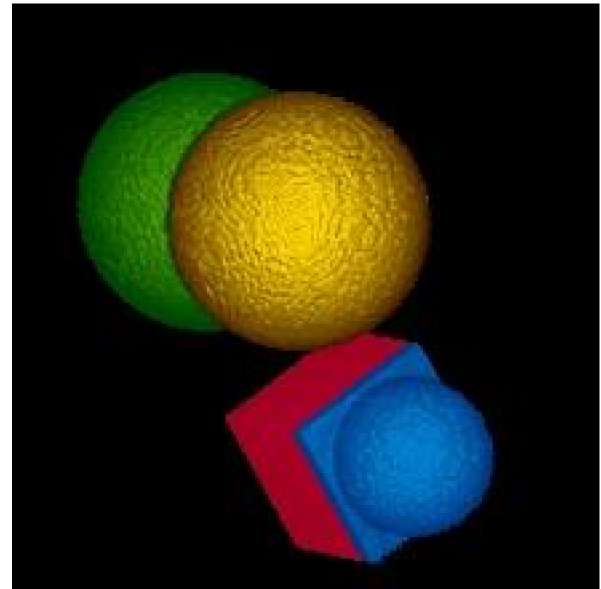


(f)

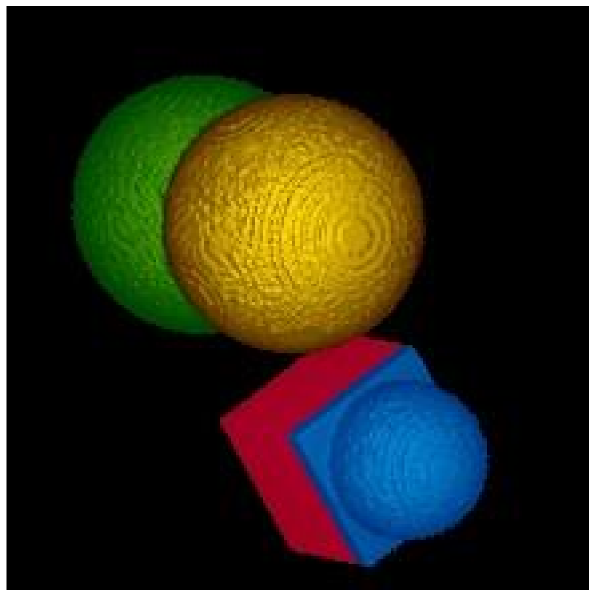
Figura E.2: Cenas de teste segmentadas: "*geometric*" (**a**), "*cena1*" (**b**), "*cena2*" (**c**), "*cena3*" (**d**), "*cena4*" (**e**) e "*cena5*" (**f**).



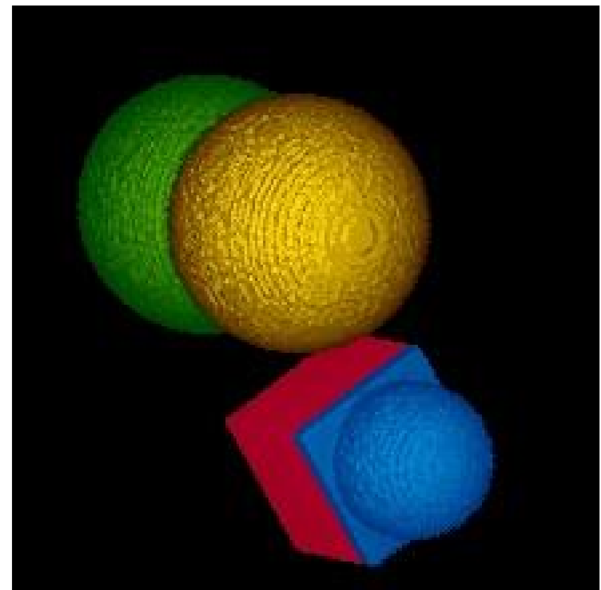
(a)



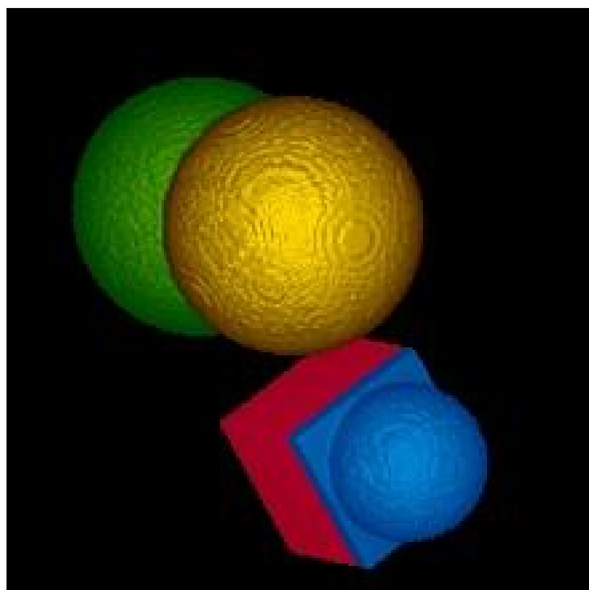
(d)



(b)



(e)



(c)

Figura E.3: Diferentes técnicas de rendering aplicadas à cena "*geometric*" segmentada com uma direção de visualização oblíqua: *raycasting* (**a**), *raycasting* com Bresenham (**b**), *raycasting* com *shear-warp* (**c**), *splatting* (**d**) e *splatting* com *shear-warp* (**e**).

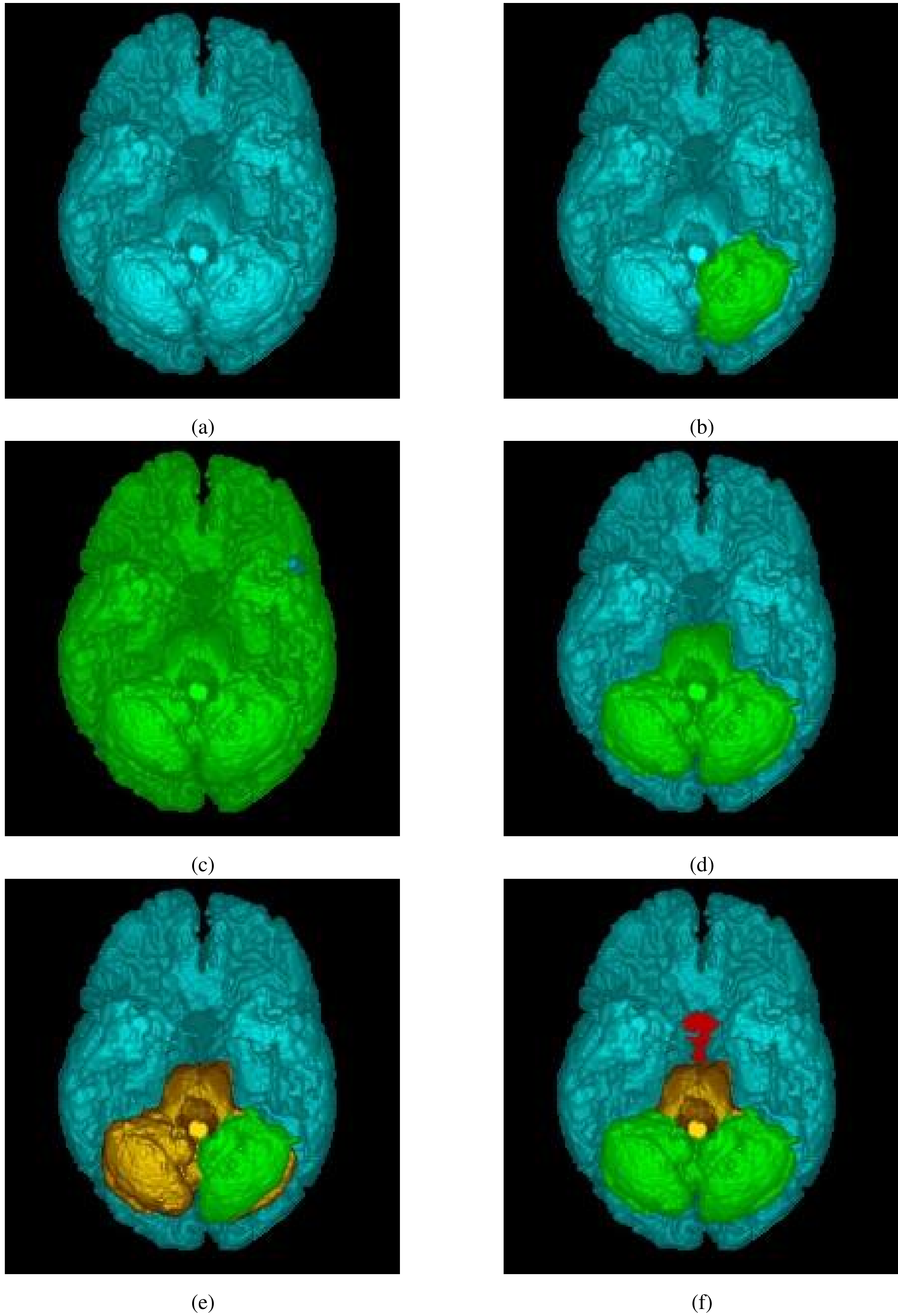
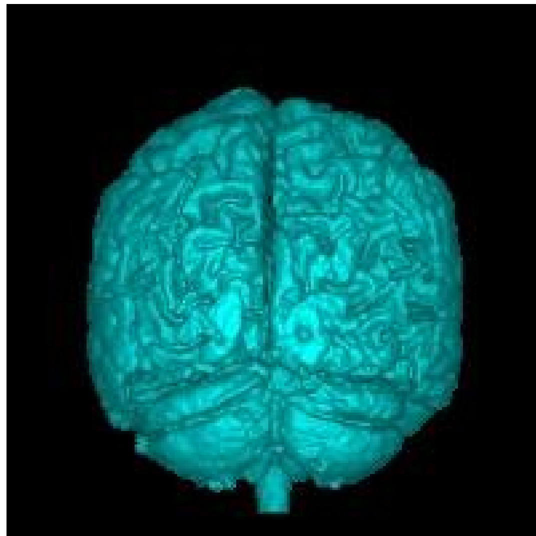
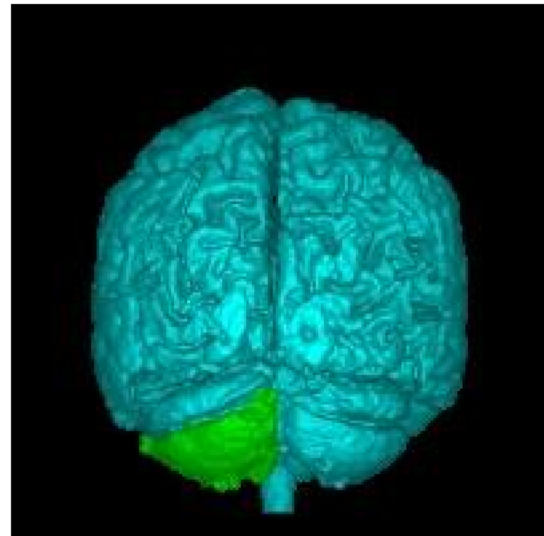


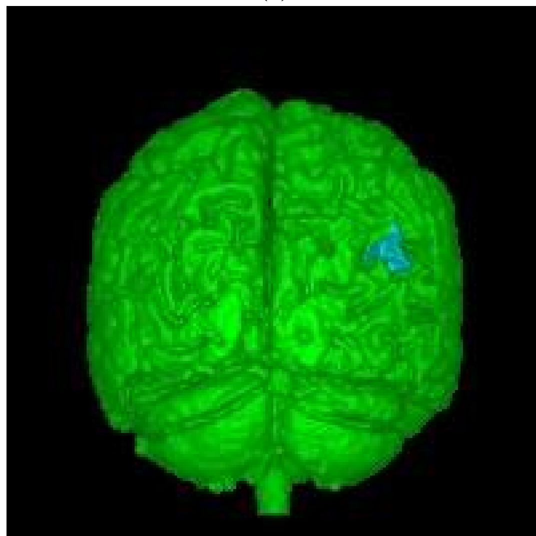
Figura E.4: Segmentação da cena “*brain*”: passos 1 a 6 (**a** a **f**). Visualização segundo o plano axial (vistas de baixo)



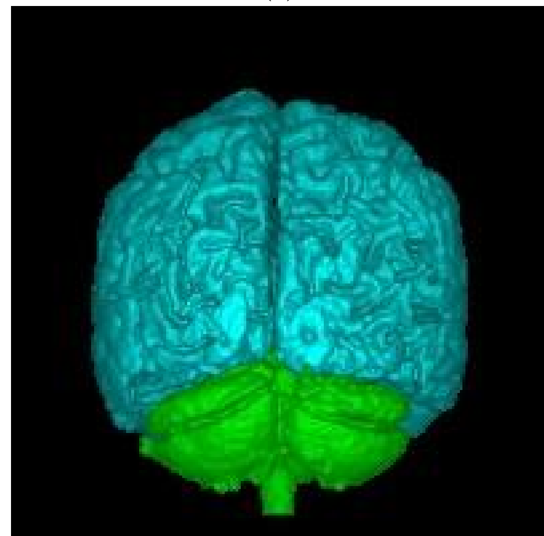
(a)



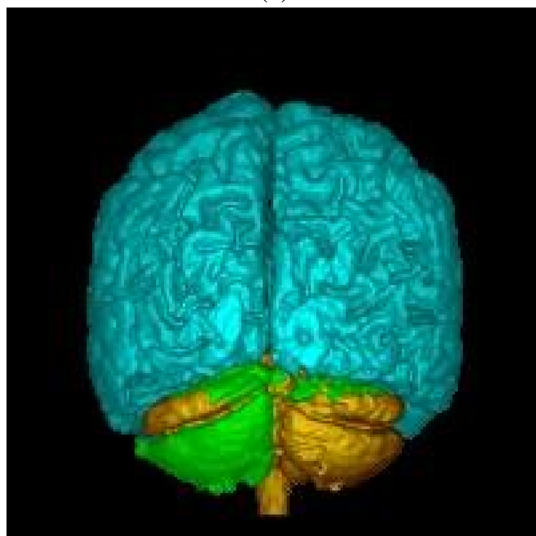
(b)



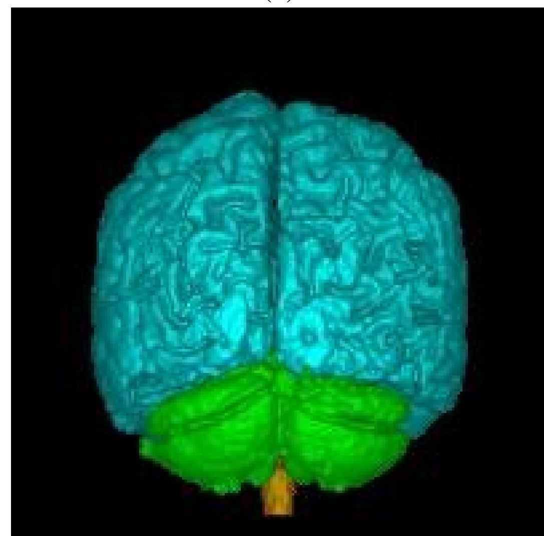
(c)



(d)

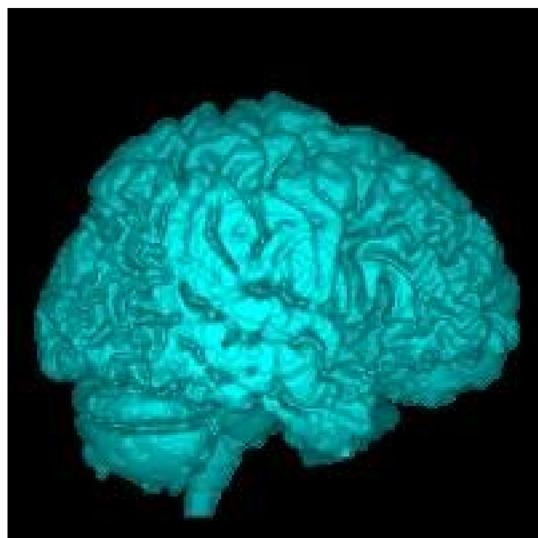


(e)

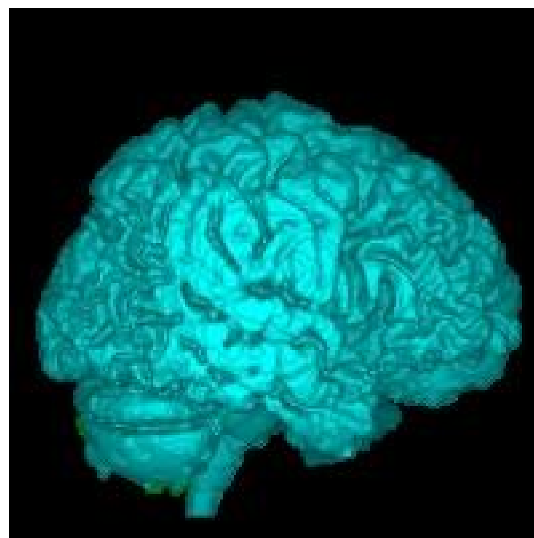


(f)

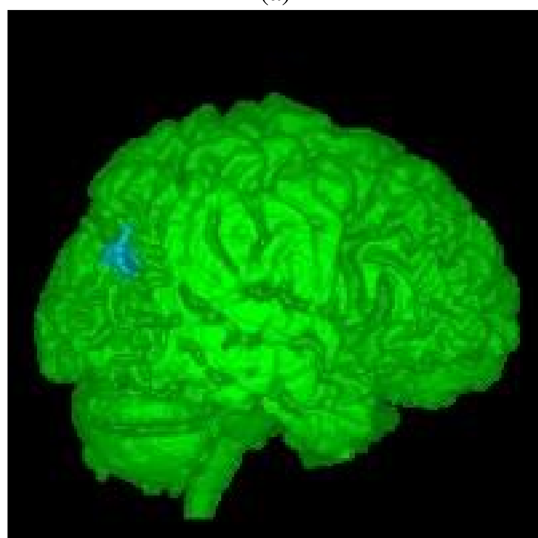
Figura E.5: Segmentação da cena "*brain*": passos 1 a 6 (**a** a **f**). Visualização segundo o plano coronal (vistas de trás)



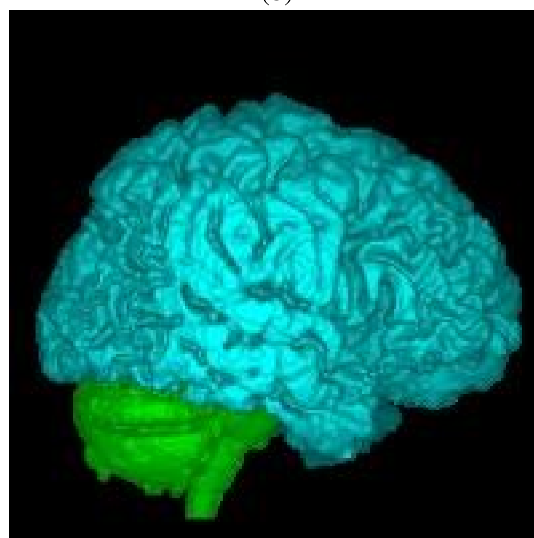
(a)



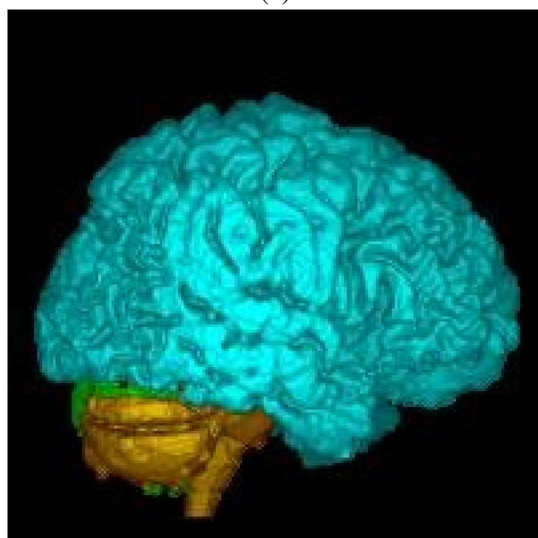
(b)



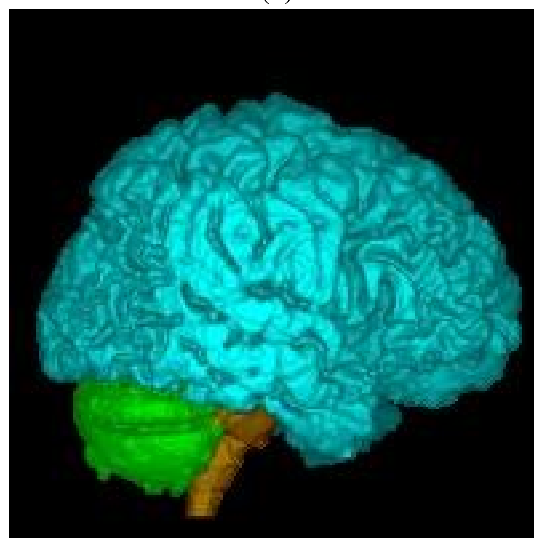
(c)



(d)



(e)



(f)

Figura E.6: Segmentação da cena "*brain*": passos 1 a 6 (**a** a **f**). Visualização segundo o plano sagital (perfis direitos)

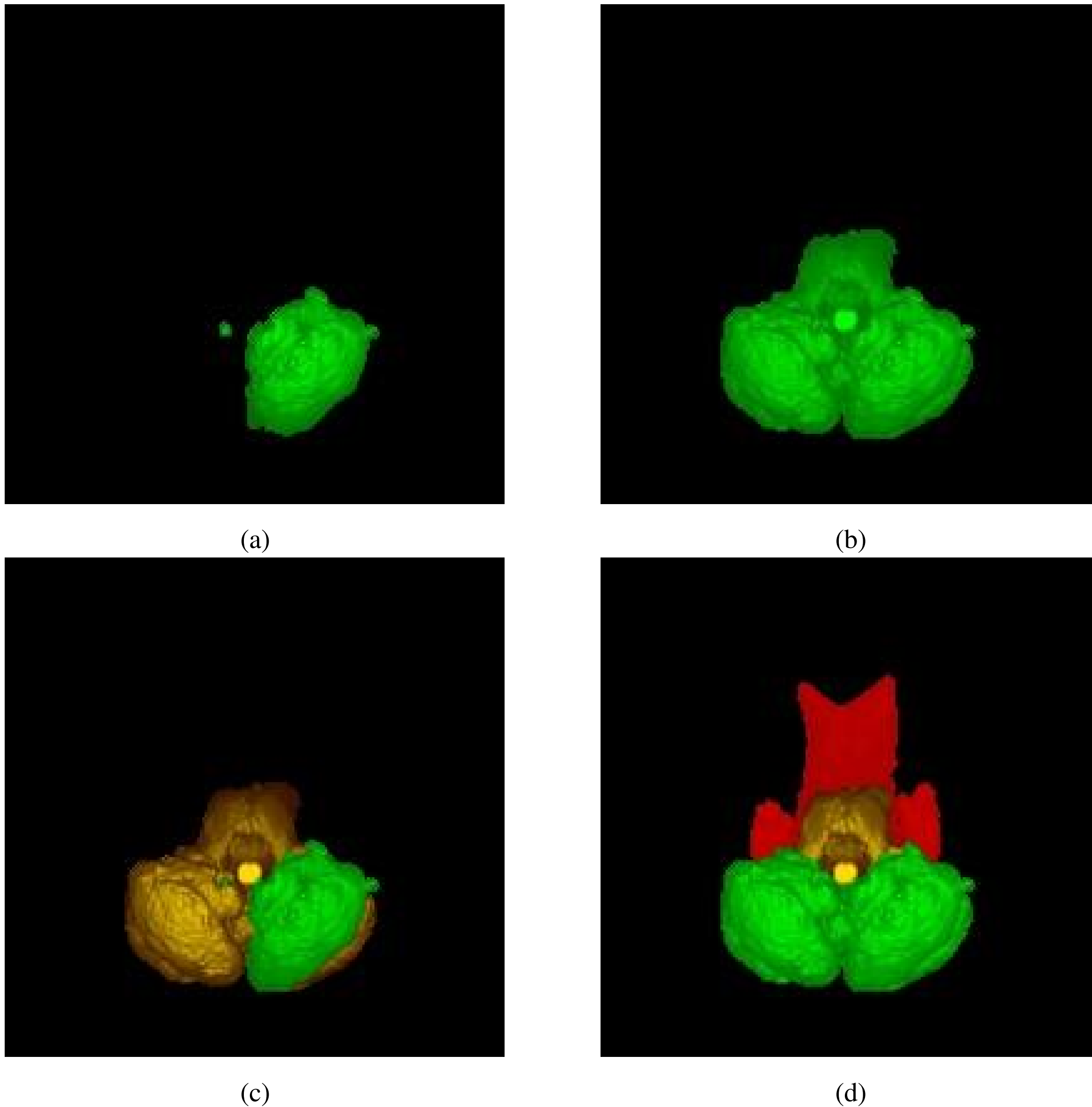


Figura E.7: Segmentação da cena “*brain3*”: passos 1 a 4 (**a** a **d**). Visualização segundo o plano axial (vistas de baixo). Os objetos de interesse são o cerebelo (de verde), a medula (dourada) e os ventrículos laterais (de vermelho).

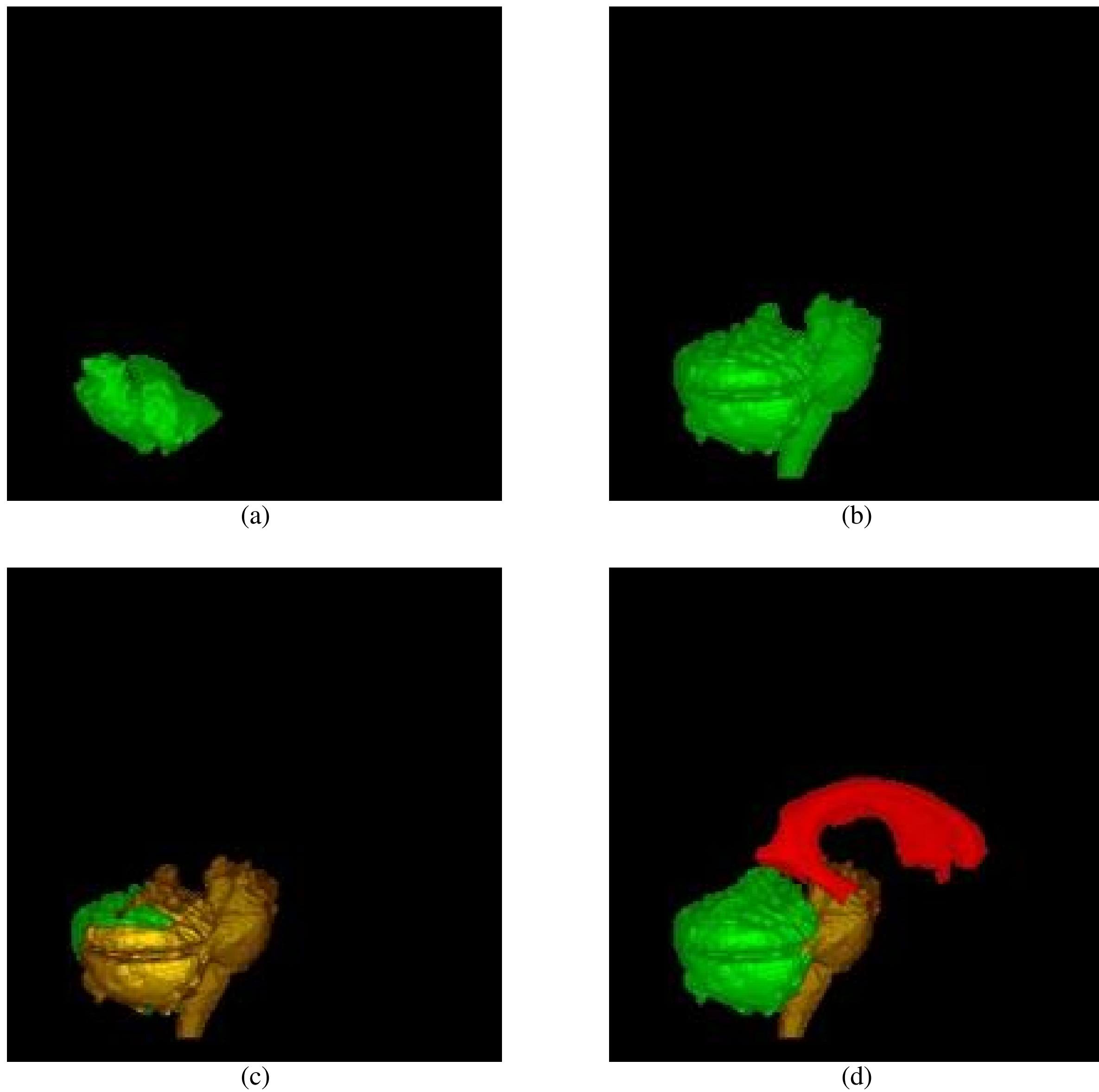


Figura E.8: Segmentação da cena “*brain3*”: passos 1 a 4 (**a** a **d**). Visualização segundo o plano sagital (perfis direitos). Os objetos de interesse são o cerebelo (de verde), a medula (dourada) e os ventrículos laterais (de vermelho).