Chapter 1

Uma Introdução à Visualização Volumétrica para Diagnóstico de Lesões Cerebrais

Shin-Ting Wu e Clarissa Lin Yasuda

Abstract

Following the trend of using images to aid the diagnose of any disease in non-invasive way, the number of applications that support physicians in processing, manipulating and interpreting them is growing continuously. Some of images even play the role of a second expert by e.g. automatically detecting suspicious nodules in mammograms and on chest X-rays. For having a complex structure with varying individual anatomies, neuroimaging applications with high sensibility and high specificity are still open problems. The expert interventions are therefore indispensable to reach correct final interpretation. An interactive volumetric visualization may help a doctor to precisely locate a subtle lesion in the native brain space and may assist a neurosurgeon in surgical planning. In this lecture, we firstly present a diagnosis protocol for detection of brain lesion with focus on the expectations of the neuroscientists on an interactive volume visualization system. Next, we present the state-of-the art technique for interactive volume rendering: a GPU-based ray-casting algorithm. We then show how we can integrate voxel-based 3D interaction tools to this algorithm. We conclude the lecture with open problems and suggestions for future projects.

Resumo

Acompanhando a tendência do uso de imagens para diagnóstico não-invasivo de uma eventual doença, cresce continuamente o número de aplicativos que assistem os médicos no seu processamento. Alguns deles chegam até desempenhar o papel de um segundo especialista identificando automaticamente, por exemplo, nódulos nos exames mamográficos e radiológicos de tórax. No entanto, por apresentarem uma estrutura complexa, algoritmos de alta sensibilidade e de alta especificidade para análise de imagens do encéfalo ou cérebro são ainda objetos de pesquisa. A intervenção de especialistas é, portanto, imprescindível para interpretá-las corretamente. Uma visualização volumétrica interativa

dessas neuroimagens pode auxiliar os médicos a detectarem com maior precisão lesões sutis no espaço nativo do cérebro e facilitar planejamento cirúrgico. Introduzimos este tutorial com uma introdução ao protocolo utilizado pelos médicos no diagnóstico de uma lesão cerebral específica, com foco em suas expectativas em relação a um visualizador interativo de neuroimagens. Em seguida, apresentamos o estado-da-arte da visualização volumétrica exploratória: o algoritmo de ray-casting implementado em unidades de processamento gráfico (GPUs). Mostramos ainda como este algoritmo pode ser modificado para integrar as ferramentas de interação na granularidade de voxels. Concluiremos o tutorial com os problemas em aberto e recomendações para projetos futuros.

1.1. Introdução

Com a rápida evolução das técnicas de aquisição não-invasiva das imagens das estruturas internas de um paciente, neuroimagens têm sido empregadas rotineiramente no processo diagnóstico em neurologia. Entre as tecnologias mais utilizadas tem-se tomografia computadorizada (TC), ressonância magnética (RM), tomografia por emissão de pósitrons (PET), e tomografia por emissão de fóton único (SPECT). A figura 1.1 ilustra as imagens adquiridas pelos escaneadores de modalidades distintas. Observe que TC mostra a estrutura óssea, RM capta a estrutura anatômica, e uma nuvem com contorno difuso é imageada por PET e SPECT. Esta nuvem contém informação funcional do cérebro. Atuando de forma complementar, estas distintas modalidades são de grande valia para investigação de anormalidades cerebrais, fornecendo com grande acurácia informações anatômicas, metabólicas e funcionais.

Na sua versão original, em formato DICOM – acrônimo de *Digital Imaging and Communications in Medicine* [39], tais modalidades são de difícil compreensão e manuseio para os médicos e neurocientistas. Para subsidiar melhor as tomadas de decisões diagnósticas, elas são processadas, interpretadas e transformadas em imagens exibíveis num monitor 2D. Vários esforços têm sido investidos no desenvolvimento de aplicativos que detectem automaticamente uma lesão. Entretanto, a complexidade da estrutura cerebral e as diferenças anatômicas individuais constituem ainda um desafio para tal automatização. Os algoritmos existentes ainda apresentam baixa sensibilidade e baixa especificidade para neuroimagens [6, 19]. A intervenção de especialistas pode ser imprescindível até na análise e na interpretação primárias das amostras. Assim, a assistência dos computadores pode se dar no nível de **visualização exploratória**, auxiliando os médicos na organização, catalogação e visualização dos dados relevantes ao longo da sua **investigação** no espaço em que os dados foram escaneados, ou seja, no seu espaço nativo. É desejável que os dados apresentados na tela sejam elementos ativos, capazes de se desdobrar em mais detalhes ou em outras representações com maior ou menor contraste, de forma a ajudar efetivamente na busca pelas áreas com lesões sutis. Neste capítulo dividimos a apresentação da visualização exploratória de lesões cerebrais em três partes.

Na primeira parte, seção 1.2, apresentamos uma visão médica do papel das neuroimagens no protocolo de diagnóstico. Sem perda de generalidade, utilizamos como referência neste capítulo uma classe de lesões cerebrais: displasia cortical focal [63]. São comentadas as principais ferramentas de processamento utilizadas nos principais laboratórios nacionais e as dificuldades encontradas pelos neurologistas na detecção de lesões sutis. Fechamos esta primeira parte com uma lista de requisitos funcionais, de usabilidade

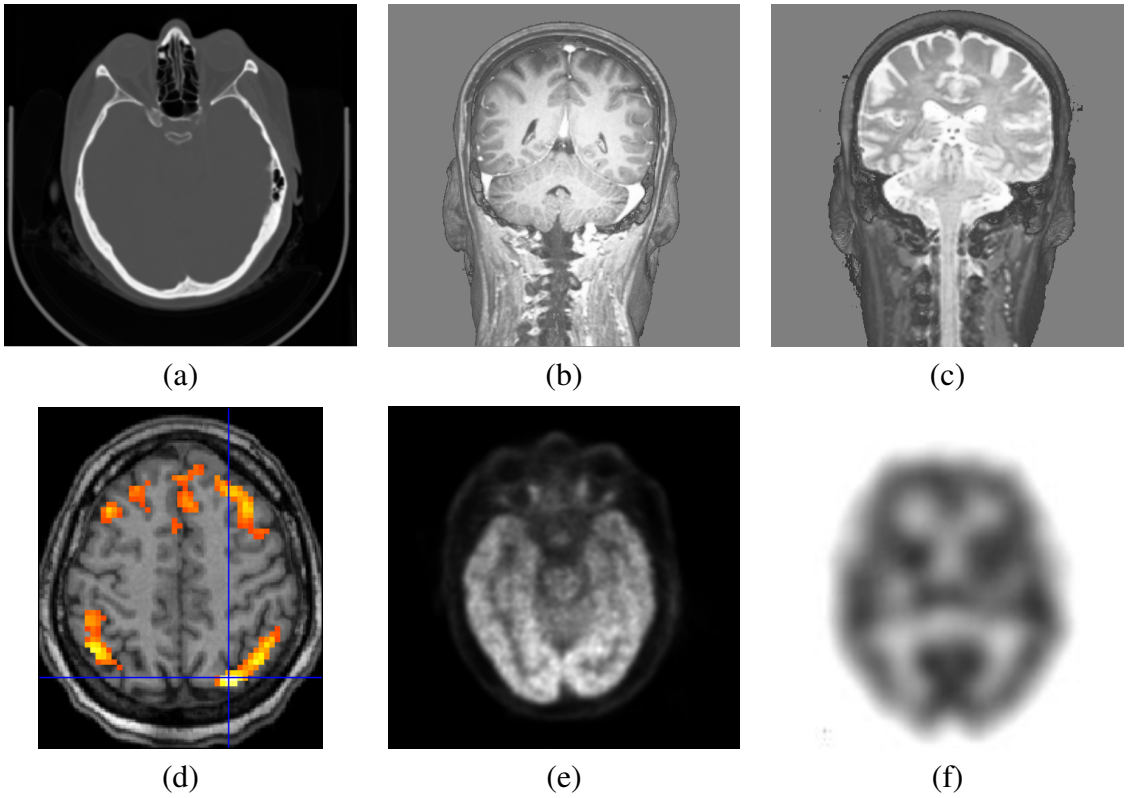


Figura 1.1. Neuroimagens: (a) TC; (b) RM com taxa de relaxamento T1; (c) RM com taxa de relaxamento T2; (d) RM funcional, (e) PET; e (f) SPECT.

e dos dados.

Na segunda parte, seção 1.3, após uma breve introdução às principais técnicas aplicadas na visualização das imagens médicas, focamos na técnica de renderização volumétrica direta baseada em lançamento de raios, em inglês *ray-casting based direct volume rendering*. Além de ser considerada a mais promissora para exploração de estruturas anatômicas complexas [64], mostramos que ela é compatível com a arquitetura paralela das unidades de processamento gráfico que se encontram em acelerada evolução. Apresentamos ainda um fluxo básico de visualização dos dados, desde os gerados pelos equipamentos de aquisição no formato DICOM até as imagens 2D exibidas nos monitores.

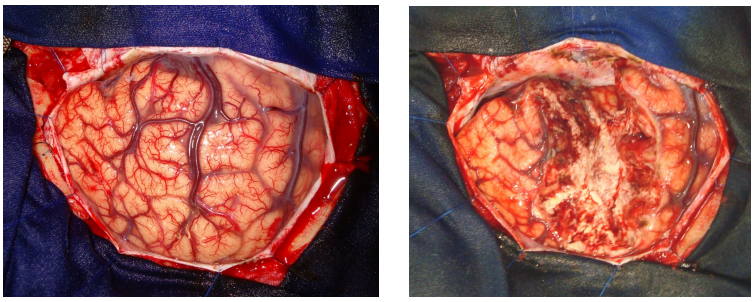
Posteriormente, na terceira parte, seção 1.4, explicamos como podemos tirar proveito do fluxo de visualização baseado em técnica de lançamento de raios para transformar todos os *voxels* exibidos em elementos ativos, capazes de responder condizentemente às ações de um usuário. Isso permitiria que um especialista, com base nos seus conhecimentos e nas suas experiências, analisasse os dados e interferisse assincronamente no seu fluxo de processamento.

Finalmente, concluímos este capítulo com os potenciais projetos de pesquisa na seção 1.5.

1.2. Neuroimagens em Diagnóstico de Lesões Cerebrais

Sem perda de generalidade, tomamos como base o curso de tratamento e procedimento adotado pelos médicos e neurocientistas do Hospital das Clínicas da Unicamp, ou simplesmente HC-Unicamp, no diagnóstico de displasia cortical focal, que é uma das formas mais frequentes de malformações do desenvolvimento cortical, estando intimamente relacionada com epilepsia de difícil controle em crianças e adultos.

A epilepsia é uma das patologias cerebrais responsável por cerca de 1% do ônus global gerado por doenças [59]. No Brasil, um estudo realizado no interior do estado de São Paulo mostrou uma prevalência estimada de 18,6/1000 pessoas [8]. Uma das suas formas mais devastadoras é a epilepsia parcial [12]. Diferentemente das epilepsias generalizadas que afetam ambos os hemisférios cerebrais, as crises que acompanham a epilepsia parcial têm origem numa região do cérebro, denominada foco epileptogênico. Tais crises são denominadas crises focais ou parciais. As crises parciais que alteram o estado de consciência são chamadas de crises parciais complexas. As epilepsias parciais acompanhadas destas crises representam aproximadamente 40% de todos os tipos de epilepsia em adultos. Elas são frequentemente resistentes às drogas antiepilépticas (DAEs) e associadas a uma maior morbidade, mortalidade [38], incidência de distúrbios psiquiátricos, além de déficits cognitivos e de memória. Apenas cerca de 40% dos adultos com crises parciais complexas apresentam um controle completo das crises com uma farmacoterapia adequada. O restante dos pacientes só consegue alcançar um melhor controle das suas crises através de ressecção ou desconexão do foco epileptogênico. A figura 1.2 ilustra o resultado da ressecção de uma área cerebral.



(a) (b)

Figura 1.2. Ressecção de uma área: (a) antes e (b) depois.

Desde o início dos anos 1970 com a invenção de tomografia computadorizada (TC) e, em torno de uma década depois, da ressonância magnética (RM), elas passaram por um desenvolvimento constante que impactou de forma decisiva na previsão dos resultados de tratamento clínico e cirúrgico em epilepsia. Por permitir observar variações na anatomia cerebral a olho nu e de forma não-invasiva, técnicas morfométricas, como VBM (*voxel-based morphometry*) e DWI (*diffusion-weighted imaging*), foram desenvolvidas para medir as estruturas cerebrais e suas alterações. Com isso, as imagens RM têm contribuído tanto para detectar esclerose hipocampal e elucidar as etiologias estruturais da epilepsia, como também para estudos prognósticos da doença de um paciente e selecionar alternativas terapêuticas mais apropriadas. A figura 1.3 ilustra uma lesão displásica extensa na região parietal direita, distinguível pela mudança nos níveis de cinza: muito mais

clara na imagem superior esquerda obtida pela RM com a sequência FLAIR que anula o sinal de líquor, e mais escura nas outras imagens de RM com taxa de relaxamento T1. Vale observar que a lesão indicada pela seta vermelha aparenta estar no lado esquerdo, porque a lateralização neste exame segue a convenção radiológica, a ser explicada na seção 1.3.1.

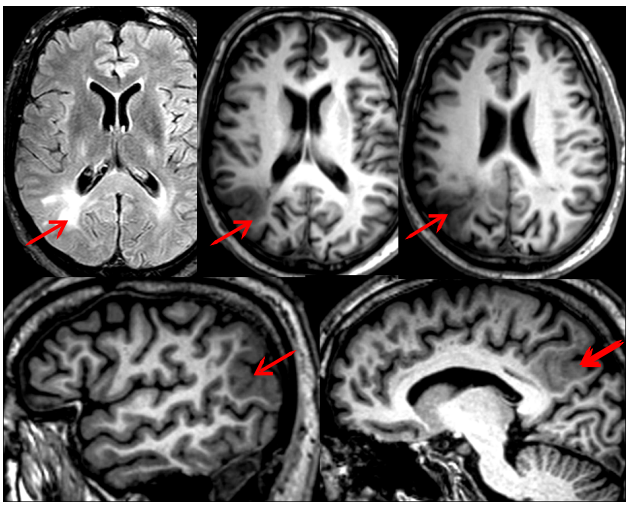
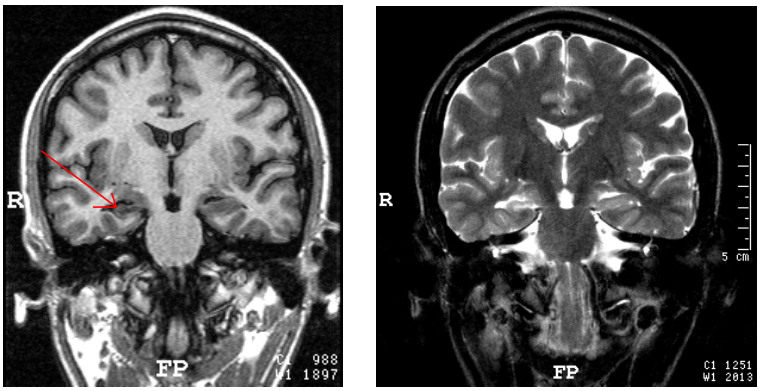


Figura 1.3. Região displásica revelada em imagens de RM com taxas de relaxamento T1 e com a sequência FLAIR.

Em geral, quando as propriedades cerebrais de interesse são mensuráveis em unidade de amostras adquiridas, como a área e o volume, ou as propriedades são passíveis às análises estatísticas, as imagens de RM vem sendo decisivas na identificação de algumas doenças neurológicas. Isso se deve ao fato de que as imagens desta modalidade apresentam alta resolução espacial e as alterações anatômicas podem fornecer alguns indícios de disfunção. Por exemplo, a perda de volume do hipocampo e a alteração do seu formato em imagens de RM com taxa de relaxamento T1, associadas à assimetria no corno temporal e ao hipersinal (região visualmente mais clara) em RM com taxa de relaxamento T2, como ilustra a figura 1.4, é um indício de esclerose hipocampal [28]. Na imagem, a região suspeita é indicada com uma seta vermelha.

Por outro lado, pesquisas têm revelado que um terço dos potenciais candidatos à cirurgia, especialmente aqueles com suspeita de displasia cortical focal, tem as suas imagens de RM pré-operativas classificadas como normais. Para esses indivíduos, além de história clínica detalhada e video-eletroencefalograma (vídeo-EEG), ou telemetria, são necessários exames complementares, tais como tomografia por emissão de pósitrons (PET) e tomografia por emissão de fóton único (SPECT) [20]. Durante o exame de telemetria, o paciente permanece internado por dias, gravando simultaneamente o exame de eletroencefalograma e filmando seus movimentos para registrar uma crise epiléptica desde o seu início. Imagens de outras modalidades, como ressonância magnética funcional (RMf) e tensor de difusão por ressonância magnética (DTI), permitem ainda prognosticar os riscos de uma intervenção cirúrgica. Resultados de pesquisa apontam ainda que os exames de RMf realizados simultaneamente com EEG podem auxiliar na detecção do foco epileptogênico.



(a) (b)

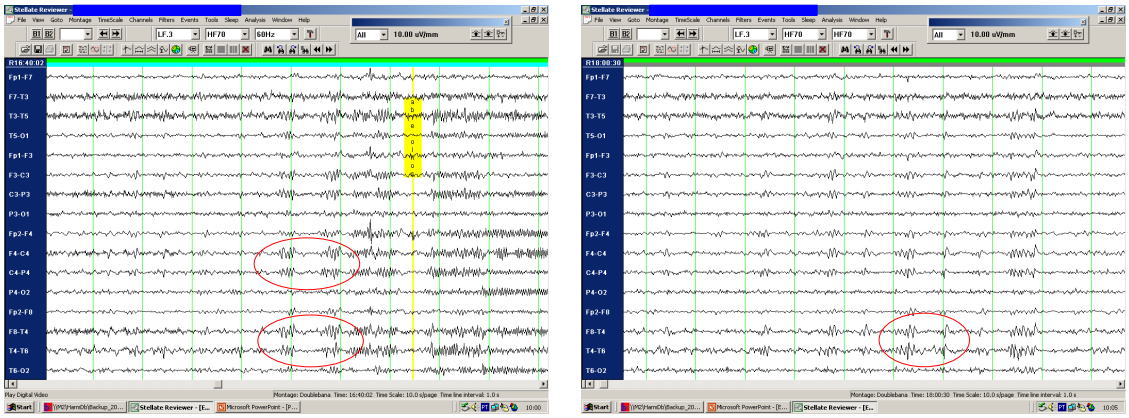
Figura 1.4. Exames combinados de RM com: (a) taxa de relaxamento T1 e (b) taxa de relaxamento T2.

Na seção 1.2.1 é feita uma síntese do protocolo de diagnósticos usualmente adotado pelos médicos na detecção de uma displasia cortical focal. Em seguida, mostramos na seção 1.2.2 como a ferramenta MATLAB-SPM8 [22] e os visualizadores de neuroimagens fornecidos pelos fabricantes dos escaneadores de ressonância magnética têm auxiliado os neuroespecialistas no estudo das neuroimagens de distintas modalidades tanto para pesquisa como para diagnóstico de displasia [43, 63]. E, apresentamos na seção 1.2.3 uma lista dos requisitos médicos quanto à visualização das neuroimagens para propósitos diagnósticos.

1.2.1. Protocolo de Diagnósticos de Displasia Cortical Focal

A epilepsia é uma disfunção crônica caracterizada por repetidas crises causadas por descargas elétricas excessivas dos neurônios. O seu diagnóstico depende, sobretudo, da análise conjunta de uma série de informações, incluindo características clínicas, eletroencefalográficas e de neuroimagens [12]. O EEG registra, por meio dos eletrodos colocados no couro cabeludo, as linhas de evolução temporal de potenciais elétricos desenvolvidos no encéfalo que chegam à superfície do escalpo. Os achados do EEG podem estabelecer o diagnóstico de epilepsia, ajudar no diagnóstico diferencial entre epilepsias generalizadas e parciais, bem como revelar padrões característicos que apontam para um determinado tipo de crise ou síndrome epiléptica. Os traçados do EEG podem identificar também o início de uma crise epiléptica. A figura 1.5 mostra o eletroencefalograma de um paciente no período ictal, ou seja no decurso de uma crise, e no período inter-ictal, isto é fora da crise. Observe as alterações nos sinais destacadas.

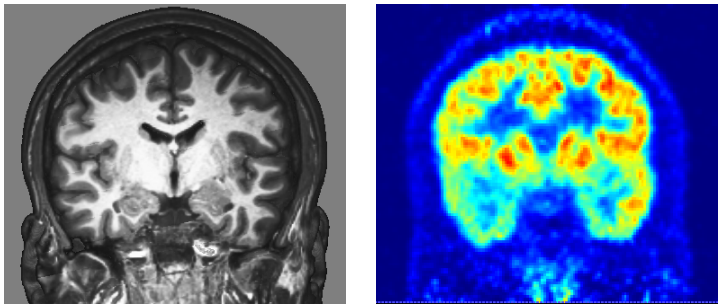
Uma limitação do EEG é que ele não fornece, com precisão, a localização do foco epileptogênico. Para pacientes que necessitem de um tratamento cirúrgico, é necessário uma série de exames complementares. Por serem as neuroimagens de alta resolução espacial adquiridas sem uso de radiotraçadores, um conjunto de exames de RM é utilizado clinicamente para investigar, a olho nu, possíveis anormalidades anatômicas, como derrames, mal-formações e tumores. Como há indicações de que a concentração dos metabólitos no encéfalo pode diferenciar classes de epilepsias, a espectroscopia por ressonância magnética (RME) é utilizada para avaliar esta concentração. Este exame pode ser



(a) (b)

Figura 1.5. Eletroencefalograma de um paciente no período: (a) ictal e (b) inter-ictal.

utilizado para análise de uma variedade de átomos. Do ponto de vista clínico, a espectroscopia mais utilizada é a de hidrogênio, devido à abundância deste átomo no organismo. Em vez de imagens como ocorre em RM, os dados de RME são representados por um gráfico que mostra picos de concentração dos metabólitos em diferentes radiofreqüências e intensidades.



(a) (b)

Figura 1.6. Exames combinados de: (a) RM e (b) FDG-PET.

Se houver discordância ou indefinição diagnóstica na análise comparativa entre os exames EEG, RM e RME, é recomendado o procedimento radioisotópico para detectar focos epileptogênicos. São, então, realizados os exames PET e SPECT ictal e interictal [32]. Em relação ao EEG, esses exames têm a vantagem de não serem diretamente dependentes da atividade elétrica do cérebro para fornecer a localização dos possíveis focos. E, comparado com RM, os exames de PET e SPECT permitem identificar os neuroreceptores e os neurotransmissores através da seleção de radiofármacos. Vale, porém, ressaltar que, para realizar os exames da modalidade SPECT ictal, os traçados do EEG são ainda necessários na identificação do início de uma crise epiléptica e na determinação do momento apropriado para administração endovenosa do radiofármaco. A figura 1.6 ilustra a imagem da modalidade RM e a imagem da modalidade PET com o radiofármaco 18F-fluorodeoxiglicose (18F-FDG), em inglês *F18-fluorodeoxyglucose*, de um mesmo paciente com crises epilépticas. Observe a região de hipoativação (em cores mais frias) no lobo temporal direito na imagem de PET, enquanto a região correspondente na ima-

gem de RM aparenta ser normal. Foi adotada a lateralização de convenção radiológica nas duas imagens.

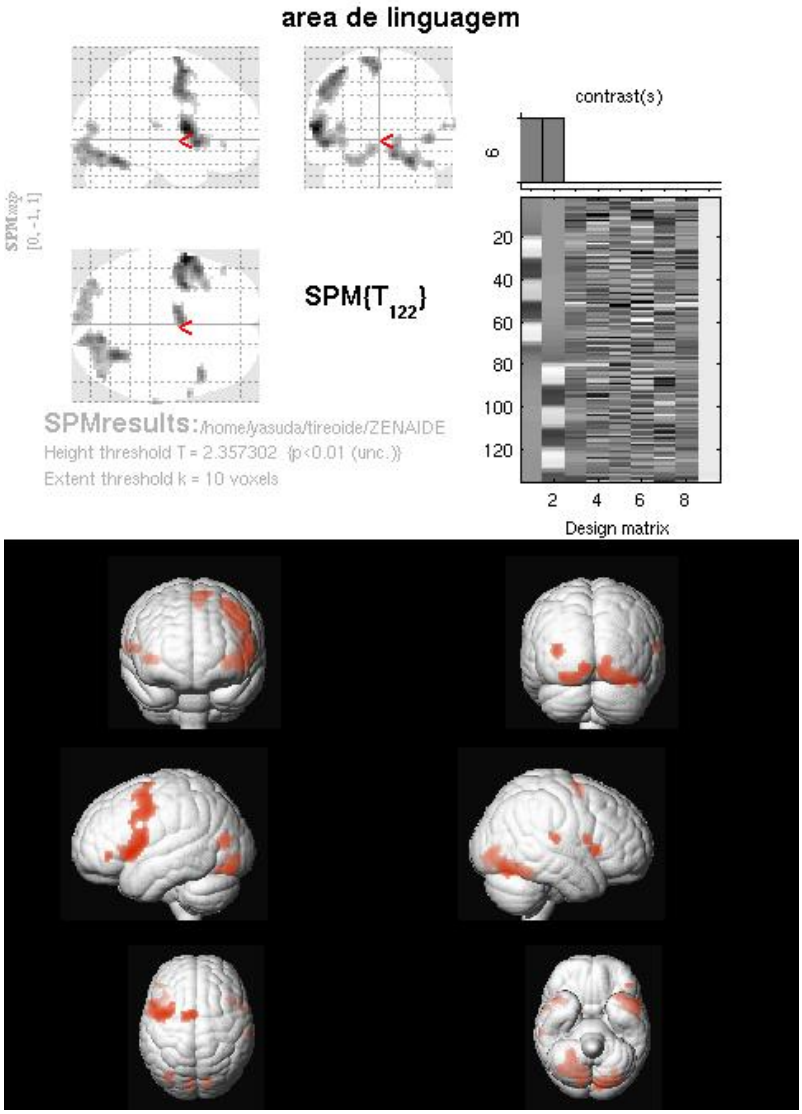
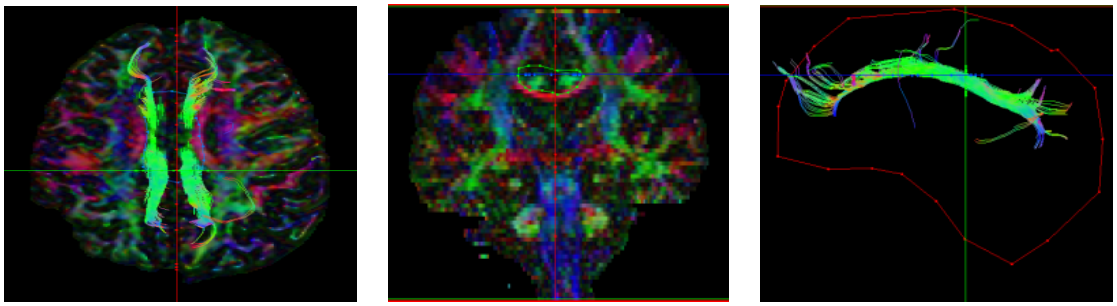


Figura 1.7. Exemplo de ativação cerebral em áreas de linguagem, durante execução de paradigma de geração de palavras.

Por ser não-invasiva, uma área promissora de pesquisa visa aplicações clínicas dos exames de RMf, avaliando o seu potencial em planejamento cirúrgico. Os exames de RMf identificam o nível de atividade cerebral através da quantificação do sinal BOLD, acrônimo de *Blood Oxygenation Level Dependent*, uma vez que a deoxi-hemoglobina, hemoglobina pobre em oxigênio, é afetada por um campo magnético de forma diferente da oxi-hemoglobina, hemoglobina rica em oxigênio. Pacientes candidatos a neurocirurgia podem ser submetidos a testes realizados através de RMf, a fim de mapear as regiões cerebrais responsáveis por funções primárias sensório-motoras, por funções de linguagem e memória ou por outras funções, visando minimizar os riscos de déficits funcionais pós-operatórios [16]. A figura 1.7 ilustra a ativação cerebral em áreas de linguagem durante

um experimento de geração de palavras. Neste experimento, o voluntário saudável deve pensar, ou gerar mentalmente, o máximo de palavras que iniciam com a letra apresentada a ele. Na parte superior esquerda da figura é mostrada a representação gráfica 2D, em três vistas, do resultado da compilação dos dados do exame; à sua direita é apresentada a matriz de delineamento do estudo, em inglês *design matrix*. E na parte inferior da figura é exibida uma visualização gráfica 3D dos resultados num espaço normalizado.



(a) (b)

Figura 1.8. DTI: (a) fatia axial, (b) giro do cíngulo circulado com linha pontilhada verde e (c) giro do cíngulo segmentado.

Outra modalidade de neuroimagem que permite avaliar os riscos de uma intervenção cirúrgica é a tractografia, pois ela permite visualizar a direção dos feixes nervosos da substância branca do cérebro que pode ser deslocada com a presença de tumores ou outras lesões. A figura 1.8 ilustra as imagens DTI onde o feixe de fibras brancas está colorido em verde, com destaque do delineamento do giro do cíngulo nas figuras 1.8.(b) e (c) em dois pontos de vista.

O uso combinado de EEG e RMf é um dos objetos de grande interesse de pesquisa. As alterações de potenciais elétricos, registrados pelo EEG, podem ser medidas com alta precisão temporal a uma baixa resolução espacial. A RMf, por sua vez, não tem grande resolução temporal, mas permite produzir mapas da atividade cerebral de boa resolução espacial. A aplicação simultânea dessas duas técnicas num único experimento permite correlacionar o estado cerebral, por exemplo a análise conjunta de descargas no período interictal com as alterações hemodinâmicas em crises epiléticas, apontando para um aumento do fluxo sanguíneo na área de suspeição. Está em investigação o valor clínico destes resultados na localização precisa de um foco epileptogênico. A figura 1.9 é um exemplo combinado de EEG e RMf para detecção de um foco epileptogênico. Com base na análise estatística dos dados dos dois exames, essencialmente os instantes de ocorrência dos eventos e as alterações hemodinâmicas, foi possível construir um mapa de área de ativação, destacada com cores. Nesse caso, ela se localiza no fundo do sulco frontal superior direito. Note que a lateralização nestas imagens segue a convenção neurológica.

A importância de um conjunto de exames para identificação do foco epileptogênico e avaliação de prognóstico é reconhecida pela Liga International contra Epilepsia, em inglês *International Ligue Against Epilepsy*, ILAE. Numa revisão recente do papel da neuroimagem no prognóstico da epilepsia [63], pudemos observar que:

- Uma boa resposta ao tratamento medicamentoso está associada a ausência (ou al-

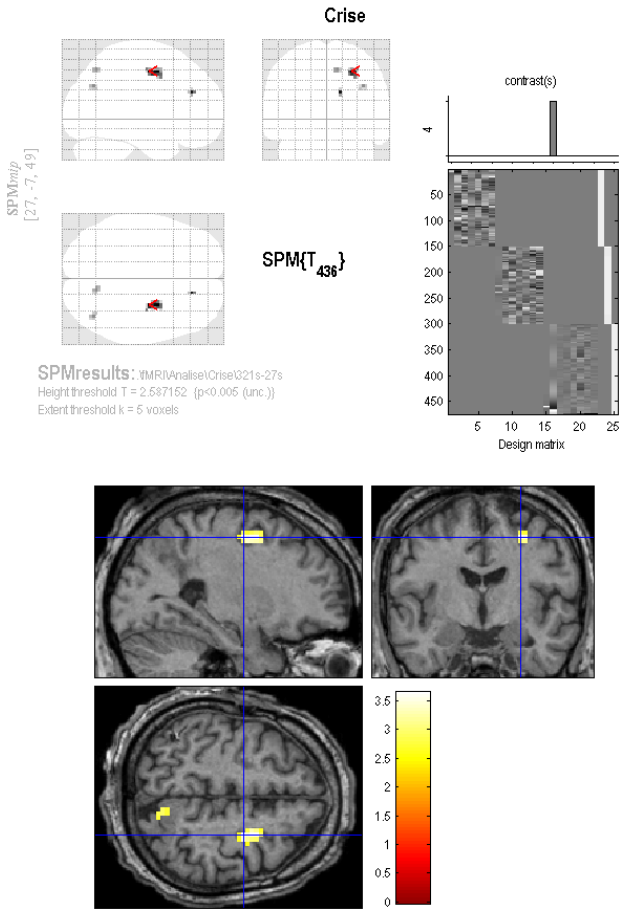


Figura 1.9. Detecção de um foco epileptogênico através de um eletroencefalograma associado às imagens de ressonância magnética funcional: foco em 3 mapas de vista (superior esquerdo), matriz de alinhamento (superior direito) e visualização do foco em imagens RM normalizadas (inferior).

teração muito sutis) de anormalidades no exame de ressonância cerebral; algumas anormalidades tais como displasia cortical focal e esclerose hipocampal indicam resposta ruim às DAEs.

- Existe um risco elevado de recorrência de crises após retirada de medicação para pacientes que apresentam atrofia hipocampal, quando comparados a pacientes com volumes normais de hipocampo.

- A melhora de qualidade dos exames estruturais de ressonância magnética tem facilitado a identificação de possíveis candidatos ao tratamento cirúrgico, uma vez que a identificação de uma lesão bem delimitada e acessível são marcadores de bom prognóstico cirúrgico.

- Uma pior resposta ao tratamento cirúrgico e medicamentoso tem sido associada à presença de áreas de anormalidades que se estendem além do foco epileptogênico,

identificadas tanto com exames funcionais (PET, EEG-RMf, RME) quanto estruturais (RM com taxa de relaxamento T1 e T2).

- Os estudos com imagens de ressonância funcional e estrutural, incluindo imagens de tensor de difusão, têm ajudado tanto na avaliação de possíveis déficits pós-operatórios quanto na identificação de áreas cerebrais eloquentes para o planejamento cirúrgico.

1.2.2. Ferramentas de Análise Quantitativa e Qualitativa

As epilepsias extratemporais sem lesões evidentes na ressonância estrutural constituem um grande desafio no campo da epileptologia. Em geral, elas estão associadas a quadros refratários graves, com várias crises num mesmo dia que geram uma limitação na vida dos pacientes e seus familiares. Como já foi dito anteriormente, um conjunto de dados é necessário para identificação de uma possível área de displasia. Os dados clínicos e de EEG inicialmente podem apontar para uma possível área suspeita no cérebro, porém muitas vezes revelam disfunção cerebral bilateral, exigindo a complementação da investigação com outros exames.

A partir dos dados clínicos e de EEG pode-se criar uma hipótese de localização e lateralização. E com base nesta hipótese, parte-se para uma investigação visual minuciosa na região alvo em imagens estruturais, buscando encontrar os sinais discretos sugestivos de displasia cortical focal. Entre os indícios mais frequentes, podemos citar alteração da morfologia dos sulcos e giros, borramento na transição entre substância branca e cinzenta e sinal "transmanto", que consiste em uma alteração do sinal na imagem RM com taxa de relaxamento T2, em direção ao ventrículo.

Uma ferramenta, de uso clínico, que suporta tais exames minuciosos de tecidos é a reconstrução multiplanar, a ser explicada na seção 1.3.2, disponível nos aplicativos proprietários fornecidos pelos próprios fabricantes dos escaneadores de RM. No hospital HC-Unicamp usamos o escaneador Philips 3T-Achieva e os seus aplicativos para examinar as imagens multimodais e para realizar a reconstrução multiplanar da imagem em alta resolução, ou seja, para "navegar" em todas as direções no meio dos dados até encontrar alguma das alterações citadas. A figura 1.10 mostra a funcionalidade de reformatação multiplanar disponível no aplicativo fornecido pelo fabricante Philips através da qual foi possível observarmos uma discreta alteração na profundidade do sulco frontal superior direito, sugerindo uma área de displasia focal.

Conforme a indicação da ILAE (Seção 1.2.1), se lesões bem delimitadas e visíveis forem encontradas em neuroimagens estruturais, teremos um prognóstico cirúrgico favorável. Os exames das modalidades PET e SPECT fazem parte da investigação pré-cirúrgica de indivíduos com imagens de RM aparentemente normais. O exame com 18F-FDG-PET é realizado no período inter-ictal e pode revelar uma área de hipometabolismo associado ao foco epileptogênico. O exame de SPECT tem papel complementar na análise clínica das lesões, uma vez que permite captar através de imagem as alterações de fluxo que ocorrem durante uma crise epiléptica, com base nas alterações metabólicas e de perfusão cerebrais. São conduzidos dois exames: um no período inter-ictal e outro ictal. A subtração entre a imagem ictal e a imagem inter-ictal resulta numa terceira imagem que pode indicar, a olho nu, a localização da área epileptogênica. Os aplicativos

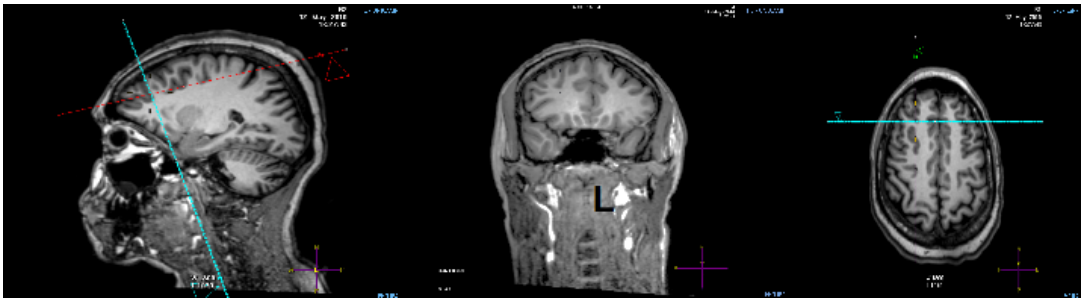


Figura 1.10. Visualizador de imagens em cortes multiplanares da Philips revela uma lesão sutil.

fornecidos junto com os escaneadores das imagens destas duas modalidades usualmente proporcionam estas análises visuais qualitativas.

É importante ressaltar que, como exames complementares, é desejável que as imagens funcionais com marcadores de lesões sejam co-registradas com as imagens estruturais quando o prognóstico indica uma intervenção cirúrgica. A figura 1.11 mostra o resultado de uma análise das imagens SPECT ictal, que confirmam o foco de crises na região frontal direita, co-registradas com as imagens de RM, a fim de determinar a localização precisa do foco epileptogênico em termos de giros e sulcos. A convenção neurológica foi adotada na exibição das três vistas da imagem.

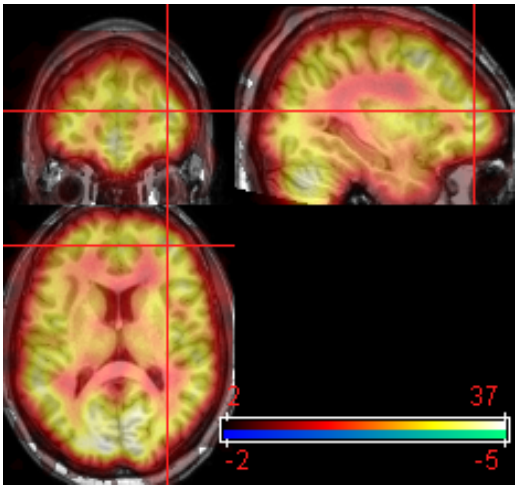
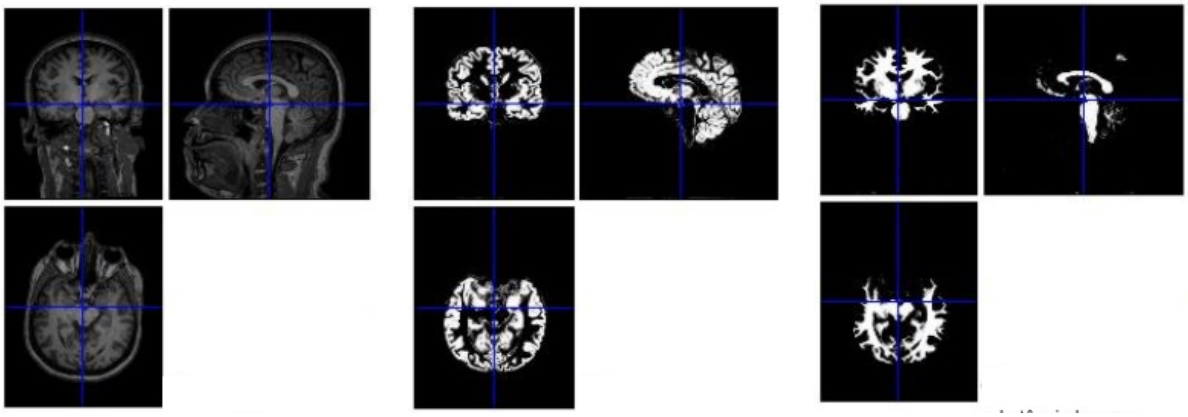


Figura 1.11. Exame de uma imagem SPECT ictal co-registrada sobre a imagem 3D de RM.

Infelizmente, os exames clínicos usuais não são infalíveis. Além disso, alguns deles são de natureza invasiva e trazem riscos e desconforto para pacientes. Métodos estatísticos, embora voltado principalmente para comparações estatísticas entre grupos de indivíduos, tem sido vastamente aplicados nas pesquisas para correlacionar melhor a ativação neuronal e a fisiologia cerebral, com a perspectiva de criar novos padrões para se avaliar pessoas com disfunções neurológicas. Com o espírito científico investigativo, usamos no Laboratório de Neuroimagens da Faculdade de Medicina da Unicamp alguns aplicativos desenvolvidos para análise das neuroimagens, mais especificamente, MATLAB-SPM8 [22], FSL [29], AFNI [41] e Freesurfer [23]. As modalidades estrutu-

rais que temos estudados são RM-T1, RM-T2 e DTI, enquanto as modalidades funcionais analisadas englobam RMf, SPECT, PET e EEG-RMf.

O SPM8 é o mais utilizado para estudo de diversas modalidades entre grupos. É uma ferramenta que requer a instalação da versão 7.1 da plataforma MATLAB, ou superior. Deve ser, porém, ressaltado que apesar de permitir a análise individual de exames, ele não é considerado ainda um aplicativo para diagnóstico clínico. Por suas funções serem baseadas em comparações por métodos estatísticos, as imagens devem ser pré-processadas de forma que a estrutura superficial destas imagens seja co-registrada com um modelo padrão, em inglês *template*. O processo é conhecido como normalização espacial, com relação a um espaço padrão. SPM8 permite também, com alguns ajustes, a análise de imagens funcionais no espaço nativo do indivíduo, cujos resultados podem ter valores clínicos quando devidamente explorados.



(a) (b) (c)

Figura 1.12. Segmentação por VBM de (a) uma imagem de RM-T1 em: (b) substância cinzenta e (c) substância branca.

Uma das ferramentas do SPM8 bastante conhecida é a segmentação das imagens de RM-T1 de alta resolução (*voxel* de no mínimo 1x1x1mm^3) em substância branca, cinzenta e líquor. Esta técnica é conhecida como VBM, sigla de *Voxel Based Morphometry*. A partir da normalização no espaço padrão e a segmentação dos diferentes tecidos, obtemos mapas individuais de substância branca e cinzenta, conforme mostra a figura 1.12. Tais mapas são usados tanto para estudos comparativos entre grupos de pacientes e controles, como também para análise de regressão com dados clínicos. A análise estatística pode ser, por exemplo, uma investigação de áreas de atrofia de substância branca e/ou cinzenta num grupo de pacientes, quando comparado com controles ou outro subgrupo de pacientes. Esse tipo de investigação é importante para o avanço no entendimento de diversas patologias cerebrais que não apresentam lesões macroscópicas, porém podem apresentar lesões microscópicas que são impercetíveis à análise visual. Sob esse ponto de vista, o aplicativo, embora não tenha ainda uma aplicação diagnóstica, representa uma ferramenta extremamente robusta para análises estatísticas na granularidade de *voxels*. Um exemplo desse tipo de análise pode ser visto na figura 1.13, que mostra as áreas de atrofia de substância branca e cinzenta de um grupo de pacientes quando comparados com controles.

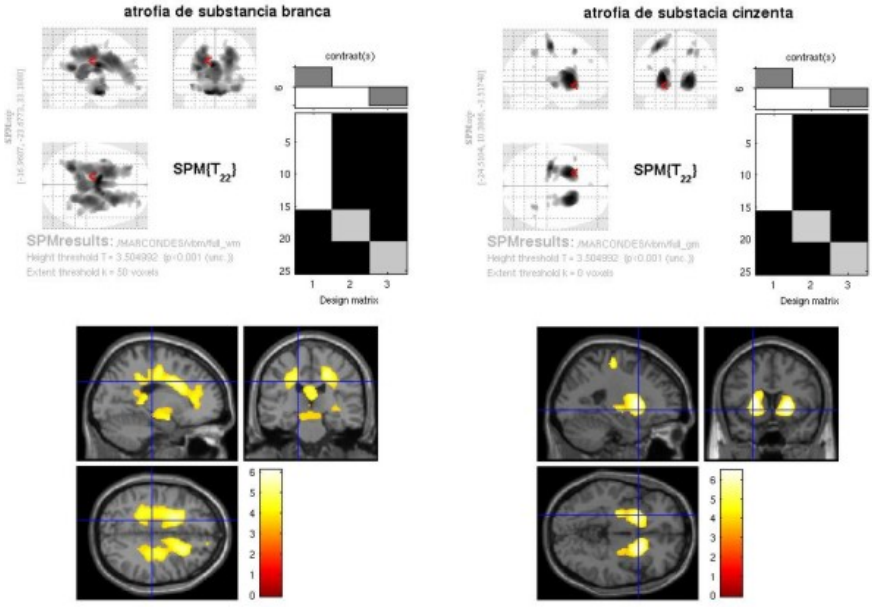


Figura 1.13. Identificação de áreas de atrofia por análises estatísticas.

1.2.3. Requisitos

Pelo exposto até agora, é possível perceber a complexidade no processo de detecção dos focos epileptogênicos, envolvendo diversos exames tecnológicos e procedimentos distintos de análise. Usualmente os neuroespecialistas são equipados de vários aplicativos para estudar comparativamente todos os exames disponíveis antes de traçar um plano de tratamento. Neste capítulo especificamente, o nosso foco é um visualizador interativo que propicie análise qualitativa das neuroimagens. Portanto, são apresentados, sob o ponto de vista médico, os requisitos funcionais, de dados e de usabilidade deste visualizador.

Entre os requisitos funcionais destacamos o controle do brilho e do contraste das imagens, a filtragem de ruídos e a reformatação multiplanar, ou a reformatação em qualquer outra geometria, que permitam perscrutar, na granularidade de *voxel*, o interior do cérebro do paciente até encontrar os focos suspeitos. É também desejável que tais operações sejam realizadas no espaço nativo do indivíduo, e não no espaço normalizado, a fim de evitar distorção anatômica e permitir medições da extensão das lesões e o planejamento cirúrgico. Vale ressaltar que é importante que os neurocientistas tenham total controle nos parâmetros de exibição das amostras, na filtragem e nos cortes para experimentar diferentes alternativas antes de emitir o laudo.

Outra funcionalidade de extrema utilidade é o processamento integrado das imagens de distintas modalidades, uma vez que nenhum dos exames de imagem é 100% eficaz. Analisar exames em diferentes plataformas e estabelecer correlações com base na avaliação visual, e subjetiva, do especialista é mais propenso a discordâncias. Portanto, é desejável que o visualizador seja multimodal, incluindo a funcionalidade de co-registro de imagens de modalidades distintas e as técnicas de fusão. Além disso, neste visualizador multimodal é desejável que os dados de diferentes modalidades sejam distinguidos

visualmente por diferentes atributos gráficos. Embora o uso de cores tenha sido uma prática comum, é interessante que o sistema dê um suporte melhor na escolha das cores para destacar áreas clinicamente relevantes.

Dos requisitos em relação aos dados, é imprescindível que o estado destes dados seja preservado o mais próximo possível do seu estado original e que os dados sejam visualizados na forma como os médicos estão familiarizados para evitar elementos novos que possam induzir conclusões falsamente negativas ou positivas. É interessante que os dados estatísticos pré-processados, como estudos comparativos com os dos controles, sejam aproveitados para aumentar o conhecimento da informação contida nas imagens.

Quanto aos requisitos de usabilidade, podemos mencionar a preferência pela disposição justaposta de janelas e não sobreposta, a fim de evitar perda por algum detalhe relevante. As funções devem ser de fácil interação com usuário, já que a maioria não tem muita experiência com computadores. Quando possível, os atalhos às funções mais comuns devam ser providos. É essencial visualização simultânea dos planos coronal, axial e sagital, por estes serem familiares aos médicos. E, acima de tudo, a interface deve se aproximar de um ambiente clínico usual, pois demandaria menos tempo de aprendizagem e geraria menos frustrações nos primeiros contatos.

Finalmente, ressaltamos que o desenvolvimento de uma ferramenta com as características listadas tem o potencial de evoluir para uma ferramenta mais avançada, o "neuronavegador". O neuronavegador é um instrumento utilizado por neurocirurgiões para se "guiar" em cirurgias de estruturas profundas no cérebro.

1.3. Visualização de Dados Médicos

Acompanhando a acelerada evolução dos equipamentos, os dados escaneados passaram de 2D (chapas de raio X) para 3D (volumes de *voxels*, imagens multiplanares ou imagens 3D), e até 4D (uma série de volumes em distintos instantes da linha de tempo). Os algoritmos de visualização destes dados também progrediram do paradigma baseado em malhas poligonais [42] para técnicas orientadas diretamente ao volume [27]. Com isso, conseguiu-se não só contornar o problema da extração de iso-superfícies de topologia complexa [10], como também atender a demanda de exibir simultaneamente a estrutura externa e a interna de uma imagem 3D.

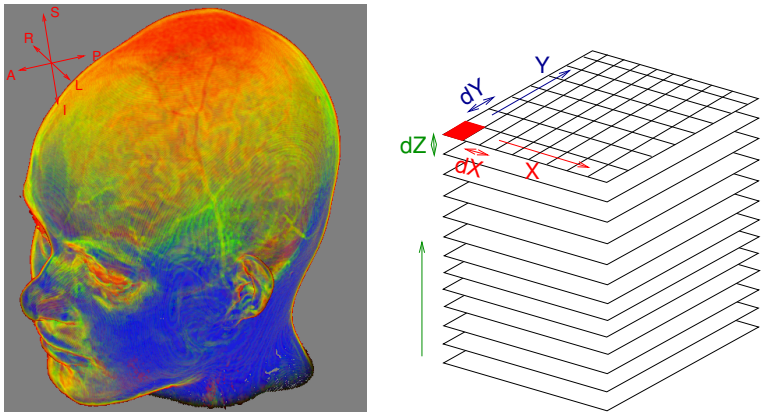
Hoje em dia, ambas as técnicas clássicas de renderização, baseadas em superfícies e orientadas diretamente ao volume, podem ser encontradas nos aplicativos médicos [64]. No entanto, a tendência é prevalecer a segunda classe de algoritmos. Um dos fatores que contribuiu significativamente para isso foi o desenvolvimento das GPUs, cuja arquitetura SIMD, sigla de *Single Instruction, Multiple Data*, é compatível com a natureza paralela desses algoritmos.

Iniciamos esta seção com uma breve introdução à representação das amostras adquiridas pelos escaneadores de distintas modalidades e às principais classes de algoritmos aplicados na visualização de dados médicos. Em seguida, focamos no estado-da-arte: técnica baseada em lançamento de raios que explora o potencial das GPUs. E finalizamos esta seção com a apresentação de um fluxo típico de visualização de dados médicos.

1.3.1. Neuroimagens

A grosso modo, podemos considerar uma imagem médica digital como um mapa de respostas dos tecidos da área de interesse a uma específica excitação identificada pela modalidade da imagem. Uma imagem de modalidade ultrassonografia é constituída de respostas a ultrassom emitido pelos aparelhos. Uma imagem de modalidade TC é, por sua vez, obtida com os raios X, enquanto uma imagem de modalidade RM registra respostas sob campo eletromagnético e uma imagem de modalidade PET ou de SPECT está relacionada com os materiais radioativos injetados no paciente.

O equipamento de aquisição varre a região de interesse seguindo uma ordem predefinida em relação a um referencial fixado no paciente: eixo esquerda(L)–direita(R), eixo anterior(A)–posterior(P) e eixo inferior(I)–superior(S) (Figura 1.14.(a)). Usualmente, adota-se uma amostragem espacialmente uniforme em cada eixo. A ordem de varredura é coluna por coluna, na direção da seta vermelha na figura 1.14.(b), progredindo linha por linha, seguindo a seta azul da figura 1.14.(b). Após finalizar a varredura de um plano, passa-se para o plano seguinte na direção da seta verde mostrada na figura 1.14.(b). O resultado é um reticulado, ou um volume, de amostras e cada amostra é chamada *voxel*, acrônimo de *volume element* [17]. Os dados das amostras variam de acordo com a modalidade de aquisição. Por exemplo, nas imagens da modalidade TC eles são as densidades dos tecidos e nas imagens da modalidade DTI, as direções de difusão das moléculas.



(a) (b)

Figura 1.14. Neuroimagens: (a) possíveis vetores-base de um referencial orientado a paciente; e (a) ordem de escaneamento das amostras.

Sob o ponto de vista de implementação de um visualizador, o que se interessa é a representação digital do volume de amostras adquiridas. A representação deve ser tal que se possa recuperar não só a informação dos tecidos como também a sua localização espacial. Tipicamente, os dados das amostras são codificados em valores escalares ou vetoriais, e estruturados em arranjos. A localização de cada amostra é, portanto, recuperável com a agregação dos seguintes parâmetros ao arranjo: a localização da primeira amostra escaneada (*voxel* em vermelho na figura 1.14.(b)), as direções de escaneamento da primeira linha e da primeira coluna, X e Y, e o espaçamento entre os elementos adjacentes em cada direção, dX, dY e dZ.

As informações vetoriais, como posição e direções, requerem, por sua vez, a es-

pecificação de um referencial. Este é o ponto mais confuso. Embora haja certo consenso em utilizar o paciente como referencial, conforme mostra a figura 1.14.(a), não são padronizados quais três dos seis vetores, $\vec{L}$, $\vec{R}$, $\vec{A}$, $\vec{P}$, $\vec{S}$ e $\vec{I}$, e, mesmo estabelecidos os três vetores, não há conformidade na ordem de escaneamento [57]. Os referenciais mais populares entre os médicos são: convenção neurológica ou **RAS** definido pelos vetores $\vec{R}$, $\vec{A}$ e $\vec{S}$; e convenção radiológica ou **LAS** constituído pelos vetores $\vec{L}$, $\vec{A}$ e $\vec{S}$. Para evitar troca de lados de um paciente na visualização, é imprescindível conhecer o referencial, ou a lateralização, utilizado.

Em face à diversidade dos equipamentos de aquisição, tanto em tecnologia quanto em procedência, houve um esforço muito grande da comunidade médica em padronizar o formato das informações médicas para facilitar a sua troca entre distintos equipamentos digitais. Desde a sua primeira publicação em 1993, o padrão DICOM tem ganhado cada vez mais adeptos [39]. Hoje em dia, todos os modernos sistemas médicos suportam este padrão. Ele inclui na sua especificação solução para dois grandes problemas: formato das informações dos pacientes e o protocolo de comunicação entre os sistemas. O arquivo das informações dos pacientes contém, além das imagens, os dados do paciente, os dados de aquisição, e o contexto em que a imagem será usada para estudos. Entre os dados de aquisição estão as direções de escaneamento, os espaçamentos, a localização da amostra no canto superior esquerdo, e o referencial utilizado. Quando omitido, assume-se que **LPS** seja o referencial, isto é, que as amostras sejam escaneadas primeiro em $\vec{L}$, depois em $\vec{P}$ linha por linha, e em $\vec{S}$ plano por plano [9].

1.3.2. Técnicas de Renderização

De acordo com Zhang *et al.*, há três principais classes de algoritmos para visualizar os volumes de dados médicos: reformatação multiplanar (MPR), renderização baseada em superfície ou renderização indireta (ISR) e renderização baseada em dados volumétricos ou renderização direta (DVR) [64].

A reformatação multiplanar (MPR) é uma técnica que permite visualizar cortes de um volume de dados. O procedimento consiste em reamostrar os dados da região de corte a partir das amostras obtidas pelo equipamento de aquisição e transformá-los numa imagem. A figura 1.15 ilustra os três planos de reformatação mais comuns em aplicações médicas: sagital, coronal e axial. Algumas estruturas, como o hipocampo e a amígdala, são melhor investigados em cortes oblíquos, perpendiculares ao eixo longo do hipocampo [13]. Planos de corte em outras angulações, como mostra a figura 1.16, foram propostos [30, 18]. Kanitsar et al. argumentaram, porém, que tais cortes oblíquos não são apropriados para investigar estruturas tubulares longas, como sistemas vasculares, tubos digestivos e arcadas dentárias [31]. Eles propuseram uma variante de MPR e a denominaram reformatação multiplanar curva, em inglês *curved planar reformation*. Ao invés de cortes planares, utilizaram-se cortes ao longo das linhas mediais destas estruturas e projetaram as novas amostras sobre o plano de imagem.

Visando tanto à visualização de complexas fraturas acetabulares, anomalias craniofaciais e estruturas intra-cranianas, como também às interações com as superfícies dos órgãos, surgiram demandas por visualização 3D. Com a disponibilidade das tecnologias de renderização orientada à superfície na década de 90, popularizou a proposta de visu-

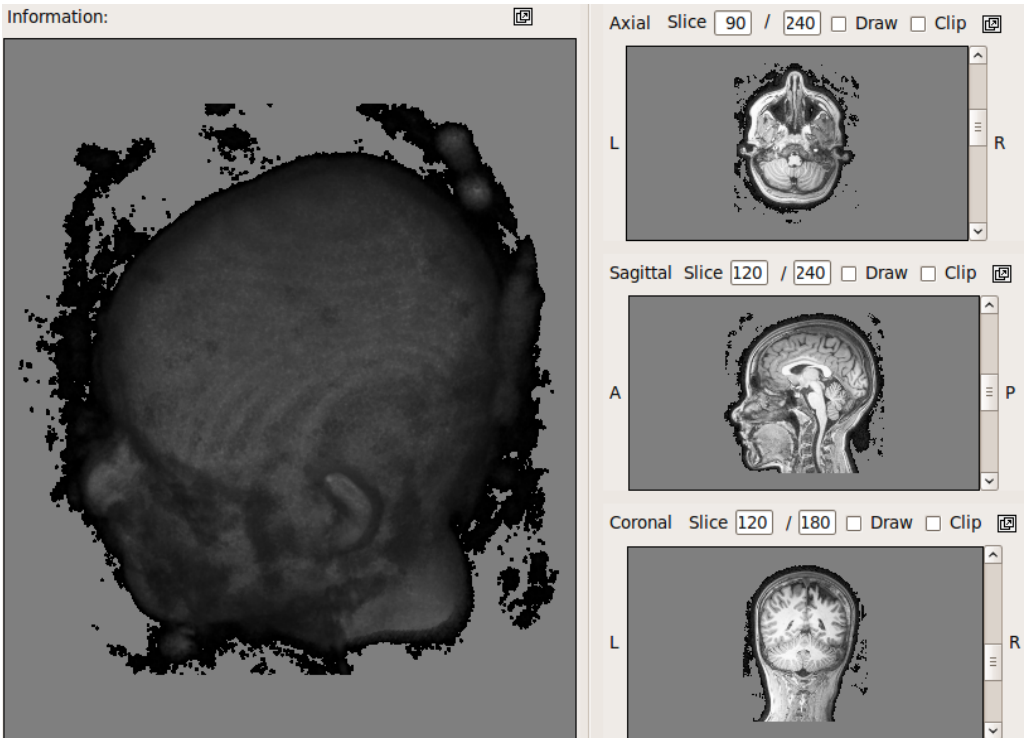


Figura 1.15. MPR: volume de dados constituído de uma sequência de fatias axiais obtidas por um escaneador RM (esquerda) e sua visualização em plano de corte axial, sagital e coronal (direita).

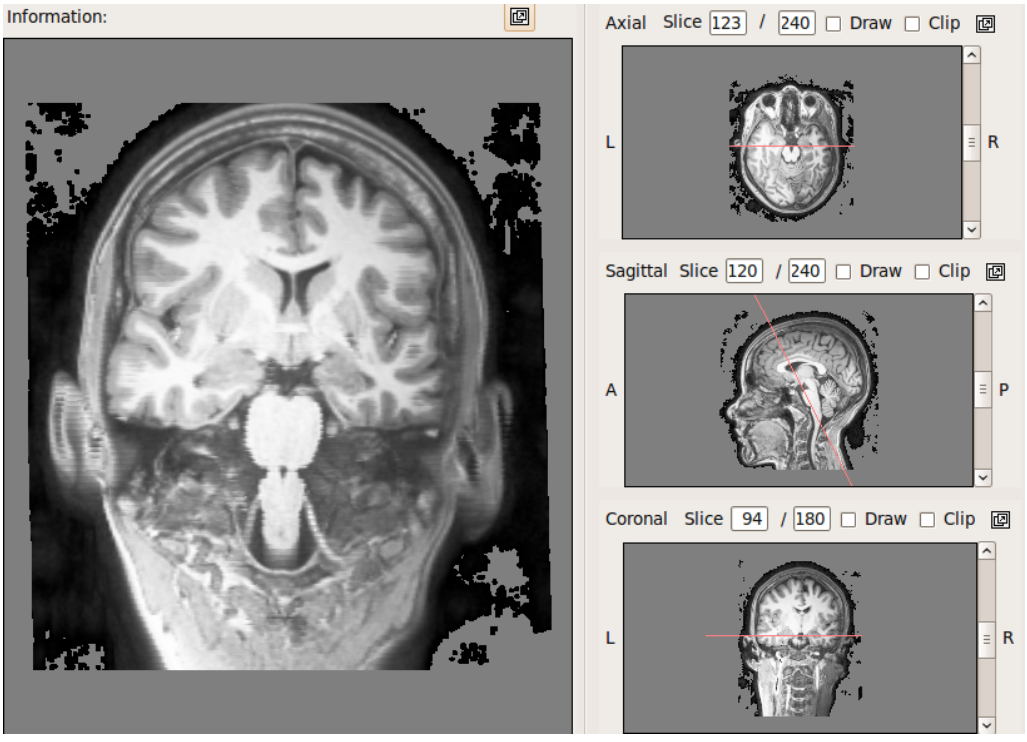


Figura 1.16. MPR: corte oblíquo de um volume de dados da figura 1.15 (esquerda) e as três vistas do plano de corte (direita).

alizar indiretamente os valores escalares das amostras correspondentes às superfícies de interesse via malhas triangulares. Esta técnica é, portanto, conhecida por renderização indireta baseada em superfície (ISR). Como tecidos de mesma natureza apresentam respostas similares, concentraram-se, então, esforços no desenvolvimento de algoritmos de construção de malhas triangulares a partir das amostras de mesma densidade, ou seja, construção de iso-superfícies. O algoritmo de construção mais popular é o algoritmo de *marching cubes* proposto por Lorensen e Cline [35]. Um resultado desta técnica pode ser visualizado na figura 1.17.(a).

Figura 1.17. Volume da figura 1.15: (a) Renderização de superfícies de interesse baseada indiretamente em malha triangular (ISR); (b) baseada diretamente em volume de dados (DSR) e (c) renderização direta do volume (DVR).

Somente com a evolução da arquitetura SIMD em unidades de processamento gráfico, foi possível nos anos de 2000 adotar o paradigma de *ray-casting*, proposto por Arthur Appel em 1968 [4]. Tornou viável processar paralelamente os dados do volume, sem a etapa intermediária de construção de malhas triangulares. O processamento baseado no espaço 3D onde se encontra a cabeça do paciente cedeu, portanto, lugar ao processamento baseado no espaço de imagem onde a cabeça seria projetada. Para cada raio de projeção lançado de um *pixel* é aplicado o modelo de iluminação na primeira amostra da iso-superfície, ou o fragmento, que ele intercepta. O vetor normal da iso-superfície em cada amostra é estimado com base nos gradientes das densidades das amostras.

Este procedimento gerou resultados aceitáveis para imagens de modalidade TC, mas não para a modalidade RM, por este último apresentar alta sensibilidade em tecidos moles. Observe as falhas em decorrência da falta de amostras para o intervalo de densidades pré-definido na figura 1.17.(b). Uma alternativa para contornar tais falhas é compormos a contribuição de todas as amostras, ou seja todos os fragmentos, ao longo do raio na cor final de cada *pixel* como mostra a figura 1.17.(c) [50]. E uma outra variante de DVR, amplamente difundida em aplicações médicas, é a técnica de projeção de intensidade máxima (MIP), ou seja, visualizar somente a amostra de maior intensidade ao longo de cada raio lançado [54].

Sendo a técnica de renderização volumétrica o estado-da-arte para visualização de volume de dados médicos, exporemos na seção 1.3.3 alguns detalhes da sua implementação em GPUs.

1.3.3. DVR baseado em Lançamento de Raios

Esta técnica consiste essencialmente em lançar a partir de cada *pixel* um raio, no mínimo, e compor os atributos gráficos dos *voxels* que o raio intercepta. Apresentamos os principais passos deste algoritmo e, antes de apresentar a sua implementação em GPUs e o procedimento necessário para executá-lo em GPUs, fazemos uma breve introdução à arquitetura da GPU e ao seu modelo de programação. Mostramos ainda brevemente algumas alternativas para determinar os atributos gráficos em cada *voxel* por meio de funções de transferência [21, 5].

1.3.3.1. DVR

O principal objetivo da técnica de renderização volumétrica direta, e suas variantes, é visualizar um bloco de *voxels* translúcidos de forma direta. De acordo com o princípio de transporte de energia, um feixe luminoso que atravessa este bloco sofre absorções, reflexões e refrações, de forma que a cor que perceberíamos no outro lado seria a "integração das atenuações" que o feixe original sofreu.

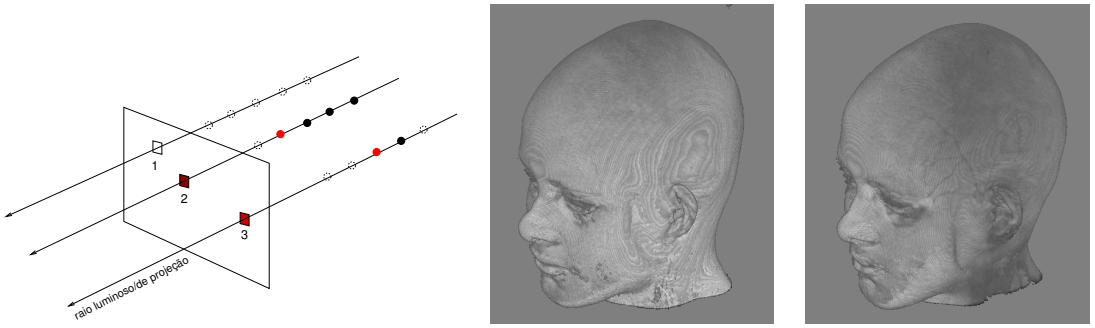
Representando a propriedade óptica de cada *voxel* i pela dupla C_i (cor) e α_i (opacidade), podemos aproximar a integração das contribuições luminosas do primeiro *voxel* ($i = 0$) até o último ($i = n$), ao longo de um raio de projeção, através da recorrência [27]:

$$\mathscr{C}_i = (1-\alpha_{i-1})C_i + \mathscr{C}_{i-1} \qquad \mathscr{O}_i = (1-\alpha_{i-1})\alpha_i + \mathscr{O}_{i-1} \tag{1}$$

com $\mathscr{C}_0 = C_0$ e $\mathscr{O}_0 = \alpha_0$, e atribuir ao *pixel* de onde partiu o raio a cor $\mathscr{C}_n$. Ou integrar do último *voxel* até o primeiro, iniciando com $\mathscr{C}_n = C_n$ e $\mathscr{O}_n = \alpha_n$, através da recorrência

$$\mathscr{C}_i = C_i + (1-\alpha_i)\mathscr{C}_{i+1} \qquad \mathscr{O}_i = \alpha_i + (1-\alpha_i)\mathscr{O}_{i+1}, \tag{2}$$

e atribuir a cor $\mathscr{C}_0$ ao *pixel*.

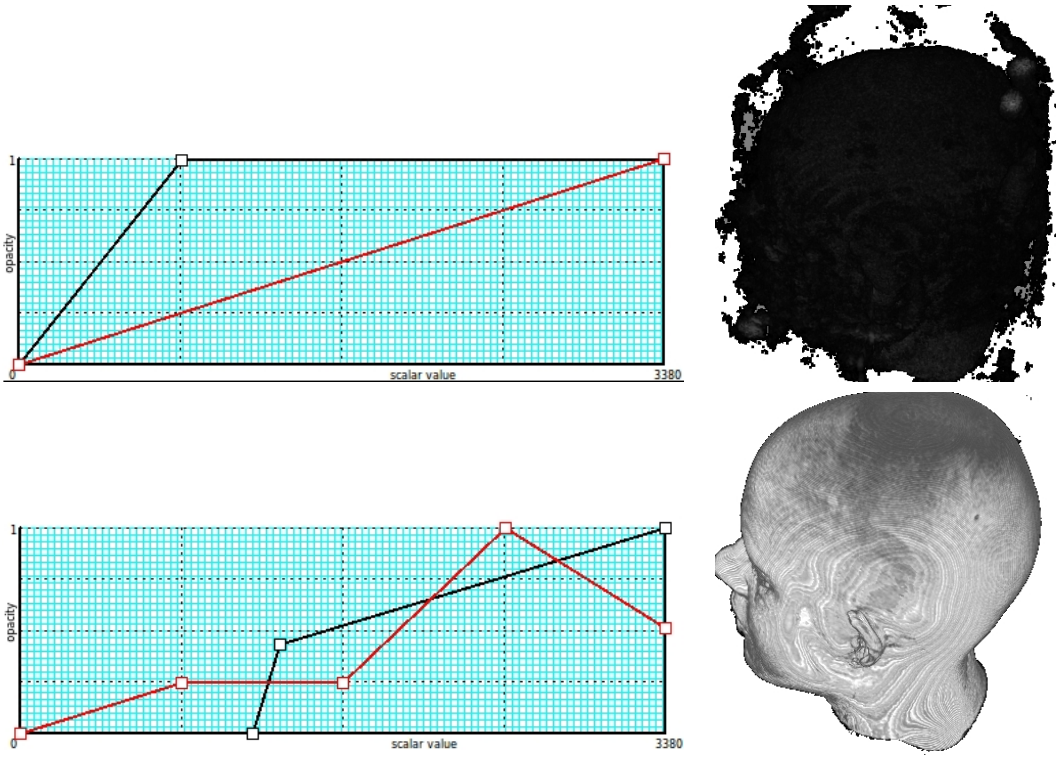


(a) (b) (c)

Figura 1.18. Renderização direta de volume: (a) traçado de um raio; (b) de frente para trás; e (c) de trás para frente.

Se amostrarmos um raio do feixe luminoso em cada *pixel* e aplicarmos nele um desses modelos de composição (Figura 1.18.(a)), teremos como resultado uma aproximação do que a visão humana percebe. As figuras 1.18.(b) e 1.18.(c) mostram, respectivamente, resultado da composição de trás para frente e de frente para trás. Observe que, neste caso específico, a diferença entre os dois modos de composição é muito sutil.

Ao compararmos os dados utilizados na composição da cor em cada *pixel* com os dados obtidos pelos equipamentos de aquisição descritos na Seção 1.3.1, podemos perceber imediatamente que são de natureza bem distinta. Os primeiros são os atributos gráficos que colorem cada *pixel* e os últimos são as respostas dos equipamentos em relação aos tecidos. É natural perguntar-se qual é a função de transferência que relaciona os dois conjuntos de valores: de um lado, as propriedades ópticas e,do outro, as densidades das amostras. Baseado no fato de que as colorações em imagens médicas, com as quais os médicos estão familiares, são mais por convenção do que por reprodução [15], é comum utilizar como tal função de transferência uma tabela de correspondência para mapear os dois conjuntos. A figura 1.19 mostra resultado de duas funções de transferência distintas aplicadas num mesmo volume de dados. Vale, porém, ressaltar que a expectativa dos médicos é que as funções de transferência sejam tais que as regiões de interesse, como lesões sutis ou malformações, se destaquem automaticamente. Para isso, uma das tendências é construir função de transferência com uso dos algoritmos de segmentação [48].



(a) (b)

Figura 1.19. Função de Transferência: (a) correspondência densidade×nível de cinza (linha vermelha) e densidade×opacidade (linha preta); (b) imagem obtida.

Sintetizando, os principais passos de um algoritmo de renderização volumétrica direta são:

1. Determine o ponto P de entrada do raio de projeção no volume
2. Determine a direção $\vec{r}$ do raio

while P está dentro do volume **do**

3.1. Acesse a densidade da amostra

3.2. Acesse as propriedades ópticas correspondentes

3.3. Acumule as propriedades ópticas com as anteriores
3.4. Avance/Retroceda P de um passo ao longo de $\vec{r}$
end while

Embora não seja de interesse para aplicações médicas, apresentamos uma diversidade de imagens que podem ser geradas com as variantes do algoritmo DVR para ilustrar a sua versatilidade. O aplicativo que utilizamos é uma versão adaptada do Spvolren desenvolvido pelo grupo de pesquisa liderado pelo Prof. Thomas Ertl na Universidade de Stuttgart [50]. Nas figuras 1.20.(a–e) foi adotado como o critério de parada no percurso ao longo de cada raio o encontro com uma amostra de densidade pré-definida. Note na figura 1.20.(e) as discontinuidades nas amostras. Os distintos efeitos visuais observados nas figuras 1.20.(a–e) foram resultados das diversas combinações dos parâmetros do modelo de iluminação. Na figura 1.20.(f) são exibidas apenas as amostras de maior densidade ao longo de cada raio de projeção, ou seja, a imagem é um resultado da técnica MIP [21].

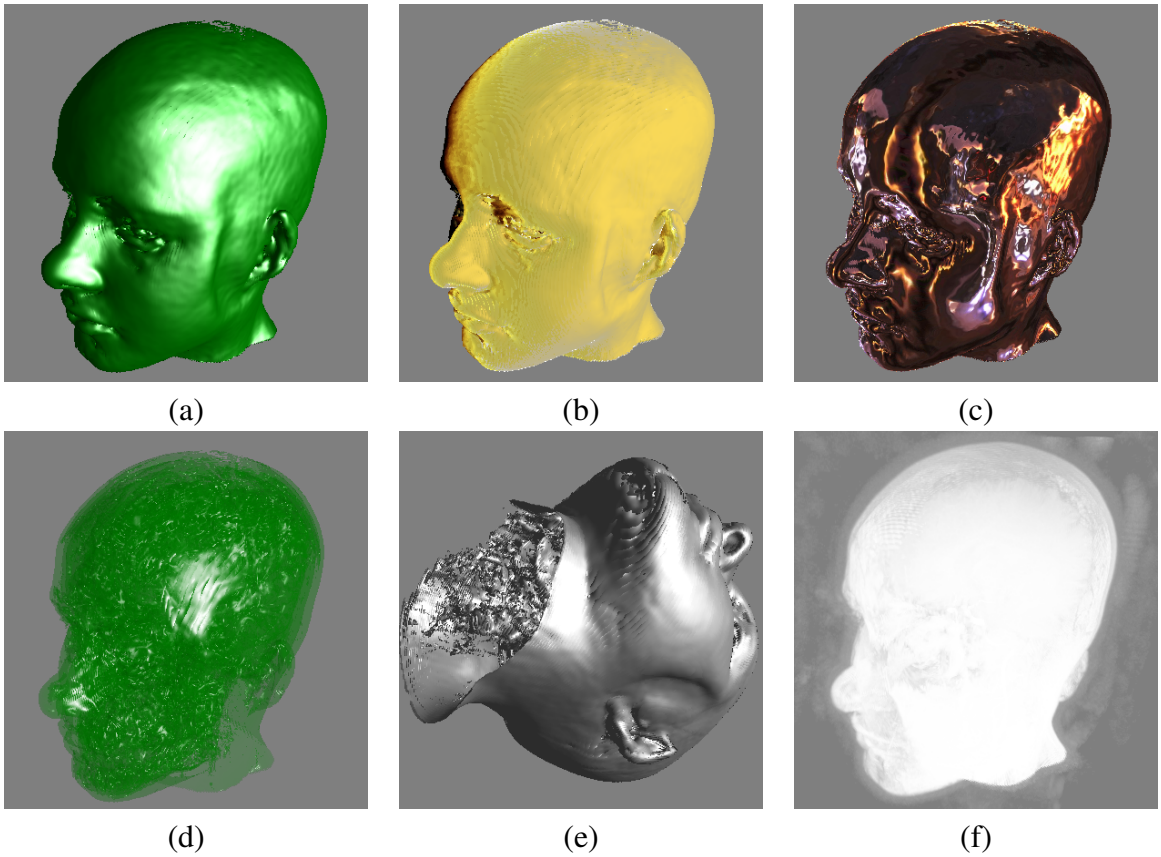


Figura 1.20. Variantes do DVR: (a–e) com distintas combinações dos parâmetros do modelo de iluminação, (f) MIP.

1.3.3.2. Unidades de Processamento Gráfico

Motivada pela maximização do desempenho em processamentos em ponto flutuante, a concepção da arquitetura da GPU difere muito da filosofia de projeto das clássicas arquiteturas paralelas da CPU. Algoritmos paralelos de alto desempenho em CPUs podem não

o ser em GPUs. Portanto, para tirar melhor proveito das funcionalidades disponíveis nas GPUs, é fundamental que se tenha uma noção sobre o seu modelo de processamento, o seu modelo de memória e o seu modelo de programação.

As GPUs são unidades multiprocessadoras constituídas por diversos processadores de fluxo, em inglês *stream processors*, que executam simultaneamente uma mesma sequência de instruções sobre os elementos de um fluxo de dados do mesmo tipo, seguindo o modelo de processamento paralelo SIMD. Ou seja, cada elemento é associado a uma linha de execução, denominada *thread*, que é executada num processador de fluxo. As linhas que são executadas em paralelo formam um *warp*. Para "esconder" a alta latência da memória de vídeo, em inglês *latency hiding*, os processadores de fluxo podem chavear de um *warp* para o outro, quando o fluxo se estagna em algum ponto de um *warp*. Só por curiosidade, ao invés de SIMD, nVidia usou SIMT, sigla de *Single Instruction and Multiple Threads*, para alcunhar a arquitetura paralela das suas GPUs.

Originalmente desenhadas para síntese de imagens, destacam-se dois fluxos de dados ao longo dos seus processadores: o fluxo de vértices e o fluxo de fragmentos. O fluxo de vértices inicia com a transferência dos vértices para GPU por um aplicativo executado na CPU. Sobre o fluxo de vértices são aplicadas as transformações geométricas e os cálculos de cores [42]. Após a reestruturação das primitivas geométricas em malhas triangulares, as facetas triangulares são rasterizadas e transformadas num fluxo de fragmentos. A principal operação executada neste fluxo é texturização, que consiste essencialmente em processar imagens pré-carregadas como texturas e "cobri-las" sobre os fragmentos. Após as apropriadas operações de composição destes fragmentos, a filtragem e os testes de descarte, o resultado é então escrito na memória de quadro, em inglês *framebuffer* e mais especificamente *window-system-provided framebuffer*. Na figura 1.21 este fluxo de dados é representado pela linha cheia.

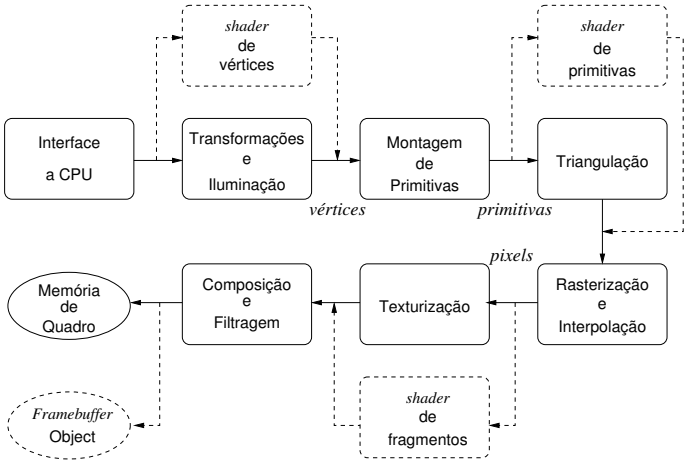


Figura 1.21. Fluxo de processamento em GPUs.

As primeiras GPUs foram projetadas como aceleradores gráficos, suportando somente funções fixas. Circuitos dedicados foram projetados: processadores do fluxo de vértices (*vertex shader*), processadores do fluxo de primitivas gráficas (*geometry shader*), processadores do fluxo de fragmentos (*fragment shaders*), unidades de mapeamento de textura (*texture mapping units*) e unidade de saída (*render output unit*). Desde a série

GeForce3 da Nvidia, os processadores do fluxo de vértices e do fluxo de fragmentos passaram a ser programáveis [58]. E, com a unificação dos processadores a partir da série GeForce8 da Nvidia [11], o processamento do fluxo de primitivas gráficas se tornou também programável. Os códigos de programação são denominados *shaders*. Possíveis alternativas ao fluxo fixo são mostradas na figura 1.21 pela linha pontilhada. Observe que, ao optar por um desvio do fluxo fixo, é da responsabilidade do programador suprir todas as funcionalidades do estágio substituído e assegurar que os dados necessários para processamento cheguem nele. Ressaltamos ainda que os resultados do fluxo de renderização não são necessariamente armazenados em memória de quadro. É possível redirecionar estes resultados para uma área de memória especificada pelo usuário, denominada *framebuffer object*, para serem reutilizados na síntese de novas imagens [3].

Além de um modelo de processamento favorável à execução maciça de dados, as GPUs apresentam uma organização de memória que propicia tal processamento. Visando a processamentos paralelos e contínuos dos elementos de um fluxo de dados, tomou como premissa que as instruções em cada linha de execução envolvam mínimos acessos a memória. Portanto, as GPUs têm, em relação a CPUs, menos registradores[1] e menos memória *cache*[2]. Há uma memória compartilhada entre os processadores de fluxo dentro de cada multiprocessador, e uma memória global, compartilhável por todos os multiprocessadores, como ilustra a figura 1.22. Embora mais baratas e com uma largura de banda maior[3], o tempo de acesso a essas duas últimas memórias é da ordem de centenas de ciclos.

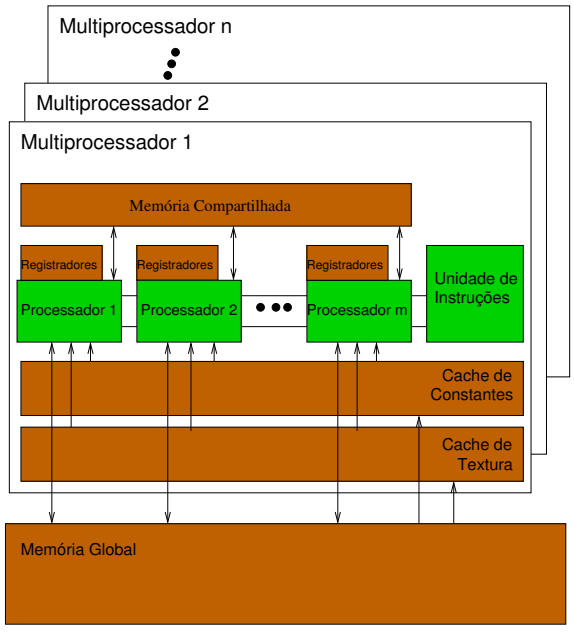


Figura 1.22. Sistema de memória em GPU.

[1]Unidades de memória mais rápidas e caras, tipicamente utilizadas como dispositivo de armazenamento temporário nos processadores.

[2]Unidades de memória de acesso rápido que serve de intermediário entre um processador de alto desempenho e uma memória lenta, e de maior capacidade de armazenamento.

[3]Taxa de transferência máxima teórica da memória.

Para poder acompanhar o ritmo dos múltiplos processadores de fluxo, ou seja aumentar a largura de banda, adotou-se a opção de barramento de centenas trilhas permitindo que a GPU transfira mais *bits* em cada acesso. No entanto, na prática, os dados requeridos por processador de fluxo são de dezenas *bits*, levando a uma baixa ocupação do barramento. Por isso, foi introduzida desde a série GeForce3 da Nvidia [58] a tecnologia de barramento cruzado, em que as trilhas do barramento são comutáveis entre os processadores e as unidades de memória. Mesmo com uma larga banda de barramento, a taxa de preenchimento dos *pixels*, em inglês *fillrate*, pode ser comprometida no estágio de texturização quando há uma demanda maciça de dados. Soluções adicionais foram incorporadas como acessos contíguos aos elementos de textura, em inglês *texture elements* cujo acrônimo é *texels*, espacialmente adjacentes, pré-processamento destes elementos e uso de memória *cache*.

O sistems de memória em GPUs opera de forma complementar ao sistema de memória do computador hospedeiro, com um espaço de endereçamento próprio como ilustra a figura 1.23. Os dados devem ser transferidos entre eles para que sejam processados pelos respectivos processadores. Mesmo com a tecnologia de barramento PCI-Express, tais transferências podem resultar em degradação no desempenho do sistema. Funções e diretivas são providas nas interfaces de programação das GPUs não só para o programador otimizar a transferência dos dados entre os dois processadores, como também para especificar a frequencia e o tipo de acessos a serem realizados nos dados, permitindo que o programador aloque locais da memória mais apropriados.

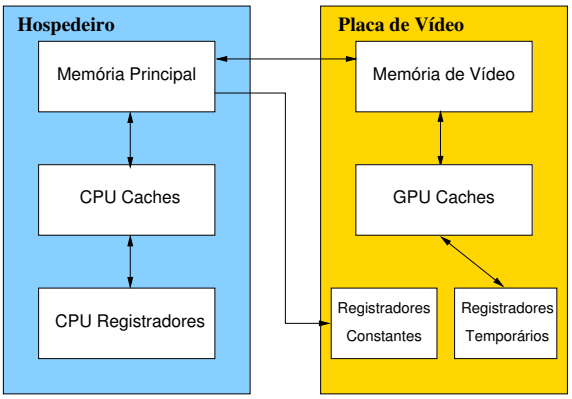


Figura 1.23. Sistemas de memória no computador hospedeiro e em GPUs dedicadas às aplicações gráficas.

A organização dos dois processadores, CPU e GPU, impacta diretamente no modelo de programação. O paradigma de programação distribuída cliente-servidor é aplicado. Na CPU, que é o cliente, é executado o programa principal que envia tarefas de renderização para GPUs, os servidores, através da sua interface de programação. Como os dois tipos de processadores têm espaços de endereçamento distintos, os dados envolvidos devem ser transferidos da CPU para a GPU. Mesmo no fluxo fixo de renderização, é necessário transferir as primitivas gráficas e seus atributos. Com exceção das texturas, que devem ser previamente carregadas na memória global, todos os outros elementos gráficos podem ser transferidos no modo imediato. A fim de tirar maior proveito do paralelismo e da largura de banda, recomenda-se, no entanto, transferir os dados das primitivas grá-

ficas organizados em bloco. Se quisermos alterar algum estágio do fluxo fixo, devemos ainda, antes de iniciar o processo de renderização, compilar, ligar e transferir as instruções organizadas em *shaders*.

Foram desenvolvidas algumas linguagens de programação de alto nível, similares à linguagem C, para GPUs. As mais conhecidas são: Cg (*C for Graphics*), desenvolvida pela NVidia, HLSL (*High Level Shading Language*), desenvolvida pelo Microsoft para ser compatível com DirectX, e GLSL (*OpenGL Shading Language*) incluída ao núcleo da interface de programação do aplicativo (API) OpenGL desde a versão 2.0 [42]. Sem perda de generalidade, utilizamos no restante deste capítulo a API OpenGL nos exemplos de código. Esta API, originalmente desenvolvida pela *Silicon Graphics Inc.* em 1992 e inspirada no modelo de programação cliente-servidor, é mantida hoje em dia pelo consórcio industrial Grupo Khronous.

Vale destacar aqui que algumas funcionalidades incluídas na API OpenGL ao longo da sua evolução proporcionaram a aplicativos maior controle sobre quando e onde os dados devem ser armazenados em GPUs:

1. a partir da OpenGL 1.1 é possível organizar um conjunto dos dados de vértices em arranjos, a fim de reduzir a quantidade de chamadas, e portanto melhorar o desempenho do sistema;

2. a partir da OpenGL 1.4 é possível transferir antecipadamente arranjos dos dados de vértices para a memória da GPU, denominados *vertex buffer objects*;

3. a partir da OpenGL 2.1 é possível transferir antecipadamente o arranjo dos dados de *pixels* para a memória da GPU, denominados *pixel buffer objects*; e

4. a partir da OpenGL 3.0 é incluído no seu núcleo os *framebuffer objects* que permitem escrever os resultados do fluxo de renderização em locais diferentes da memória de quadro.

A API OpenGL é revisada periodicamente para acompanhar a evolução das GPUs. No momento da elaboração deste capítulo, a versão mais atual de OpenGL é 4.3 [26]. Para os algoritmos que apresentaremos, as funcionalidades da versão OpenGL 2.0/GLSL 1.10 são suficientes. No entanto, como houve uma revisão mais radical na versão OpenGL 3.2/GLSL 1.50, utilizaremos esta versão para ilustrar o modelo de programação em GPUs modernas.

Todos os elementos envolvidos em fluxo de renderização são abstraídos em **objetos**. Existem, por exemplo, **objetos de arranjo** (*vertex array object*) para compactar os atributos das primitivas gráficas no cliente, **objetos de textura** (*texture object*) para representar todas as informações relacionadas com as imagens a serem utilizadas no processo de texturização, **objetos de** *buffer* (*buffer object*) para abstrair os dados armazenados na estrutura de memória gerenciada pelo servidor, e **objetos de programa** para abstrair o conjunto de instruções encapsuladas em **objetos de** *shader*.

Os tipos da dado básicos, GLubyte, GLbyte, GLushort, GLshort, GLuint, GLint, GLfloat, e GLdouble são similares aos tipos básicos da linguagem C. Adicionalmente,

OpenGL dispõe de alguns tipos próprios com uma semântica mais específica, como GLsizei que representa o tamanho de um espaço da memória. Todos os objetos são providos de nomes que podem ser gerados automaticamente com a instrução **glGen***, onde o asterisco corresponde ao tipo de objeto. Quando se deseja manipular um objeto específico, como carregar os dados nele, é necessário conectá-lo através da função glBind*, como glBindTexture, glBindVertexArray e glBindBuffer.

Os *shaders*, que substituem alguns estágios do fluxo de controle fixo, tem o seu modelo de programação restrito à arquitetura da GPU. Em GPUs modernas são no mínimo dois *shaders* para programá-las: um *shader* de vértices para mapear os atributos dos vértices, como cor, vetor normal e coordenadas de textura, em atributos de fragmentos; e um *shader* de fragmentos que "pinta" os fragmentos com os atributos recebidos e os transfere para um *buffer object* ou para a memória de quadro. Ao escrever o código de um *shader*, vale lembrar algumas suas peculiaridades [21]:

1. As instruções de um *shader* são aplicadas paralelamente em todos os elementos do fluxo. Deve-se maximizar o fluxo de dados.

2. O processamento de um *warp* só termina quando todas as suas linhas de execução tiverem sido concluídas. Deve-se minimizar e simplificar a sequência de instruções num *shader*.

3. A localidade espacial em termos de coordenadas da textura impacta diretamente no desempenho dos processadores. Deve-se organizar os dados adjacentes da textura em posições contíguas.

4. Através dos qualificadores dos dados e das funções de transferência de dados, o programador especifica em qual memória devem ser transferidos os dados para um melhor aproveitamento das funções do processamento. É da responsabilidade do programador balancear os processamentos e os acessos às memórias.

5. Não suporta estruturas complexas como pilha e fila, nem permite alocação dinâmica da memória. Deve-se antecipar a alocação dos espaços de memória e a transferência de dados necessários a um fluxo de renderização.

Finalmente, é importante frisar que, para ser totalmente independente do sistema computacional onde a GPU é instalada, a API OpenGL não inclui as funcionalidades do sistema de janelas nem o processamento dos eventos gerados pelo usuário. Estas funções, dependentes do sistema operacional, são executadas no espaço de endereços da CPU e gerenciadas através de uma interface gráfica de usuário, em inglês *Graphical User Interface* cuja sigla é GUI.

Como os serviços oferecidos pela GPU se limitam a processamentos numéricos relacionados a renderização, cabe ao seu cliente providenciar uma área na tela do computador a fim de exibir as imagens gravada por ela na memória de quadro. Uma GUI, por sua vez, encapsula em componentes denominados *widgets* todas as funcionalidades do sistema de janelas e as interações com os dispositivos de entrada, como *mouse* e teclado, e de saída, como o monitor. Portanto, sob o ponto de vista de programação, além de instruir a GPU o quê fazer, o cliente precisa preparar o ambiente de janelas via os métodos

oferecidos por uma GUI antes de iniciar o fluxo de renderização a fim de assegurar que os resultados possam ser exibidos na tela do monitor. A figura 1.24 esquematiza o fluxo completo dos dados, desde as instruções do cliente até a exibição das imagens na tela, envolvendo as duas unidades.

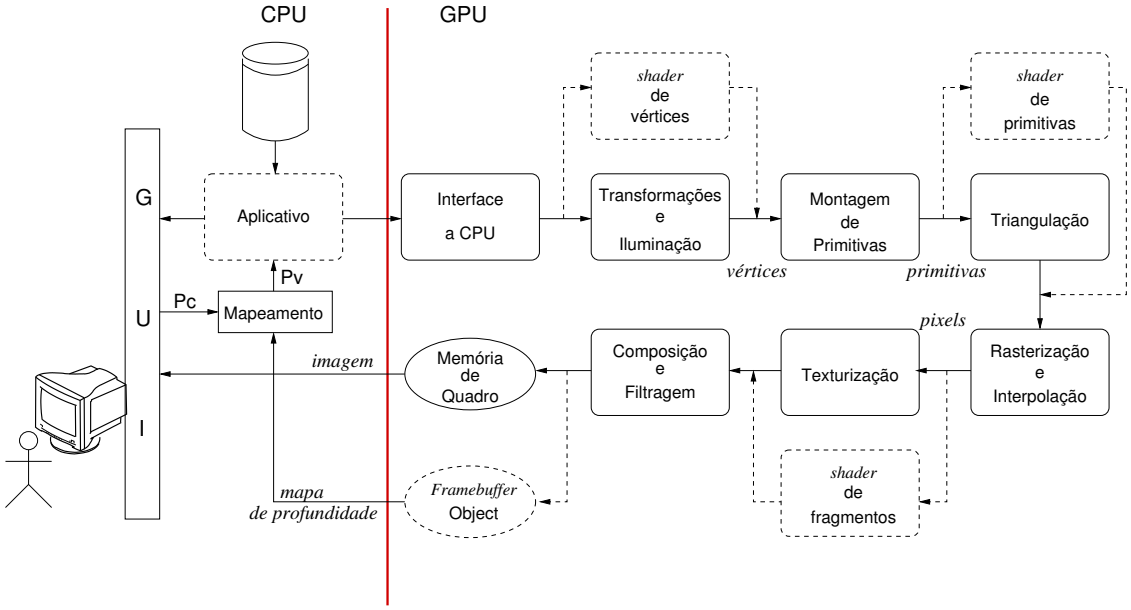


Figura 1.24. Fluxo de renderização com a saída na tela de um monitor.

Os componentes de GUI são facilmente encontrados em bibliotecas de desenvolvimento, conhecidas como *toolkit* [34]. Hoje em dia, todas do nosso conhecimento suportam conexão com a OpenGL. GLUT, acrônimo de *OpenGL Utility Toolkit* desenvolvido pelo Mark Kilgard, é a mais popular entre os iniciantes de OpenGL por sua simplicidade e por apresentar um conjunto mínimo de funções necessárias para gerenciamento de janelas e eventos [25]. FLTK, sigla de *Fast Light Toolkit*, é outra biblioteca simples, porém com mais funcionalidades para desenvolvimento de aplicativos de médio porte [46]. Exemplos de bibliotecas mais completas são Qt [45], GTK [51] e wxWidgets [2]. Por ser multi-plataforma, por usar os componentes nativos de cada plataforma, por ter uma licença menos restritiva e por suportar a linguagem C++, utilizaremos nos exemplos deste capítulo as funções de wxWidgets.

1.3.3.3. Uma Implementação em GPUs

Feita a introdução às GPUs, vamos apresentar nesta seção uma implementação em GPU do algoritmo DVR explicado na seção 1.3.3.1.

Para renderizar modelos 3D com as funcionalidades disponíveis no fluxo fixo das GPUs, é necessário transformar o volume de dados de interesse em facetas poligonais. Entre as técnicas que vimos na seção 1.3.2, tanto a técnica ISR quanto a técnica MPR são baseadas em facetas. Na ISR uma iso-superfície aproximada em malha poligonal é extraída do volume de dados e na MPR, um corte planar é separado. O recurso de textura

é explorado para "cobrir" estas primitivas gráficas com as densidades armazenadas em neuroimagens. E para visualizar um volume diretamente, como deve ser o procedimento?

Estendendo o paradigma de cortes planares, um volume de dados pode ser considerado como uma superfície fechada, ou seja um paralelepípedo de 6 faces conhecido como *proxy geometry*. Esta superfície e o seu interior são preenchidos com as densidades dos tecidos escaneados. No entanto, ao enviarmos esta representação para o fluxo fixo de uma GPU, somente as seis faces serão renderizadas e "coloridas" com os respectivos *voxels*. Isso não atenderá a expectativa esquematizada na figura 1.18.(a): o resultado desejado é a composição dos *voxels* ao longo do raio de projeção lançado a partir de cada *pixel* para podermos visualizar o interior do volume, e não somente os *voxels* da periferia do volume. Uma solução seria reprogramar o estágio de "texturização" com o algoritmo apresentado na seção 1.3.3.1, atribuindo a cada *pixel* a "cor" composta das densidades de todos os *voxels* ao longo do raio de projeção, e não a "cor" da densidade do *voxel* que cobre o fragmento visível. O seguinte *shader* dvr.frag é uma implementação do algoritmo da seção 1.3.3.1:

```
#version 150

uniform mat4 viewMatrixInv;               // matriz inversa de MODELVIEW
uniform float step;                       // passo de percurso
uniform sampler3D VOLUME;                 // densidades da neuroimagem 3D
uniform sampler1D TRANSFERFUNCTION;       // função de transferência
in vec3 texCoord;                         // coordenadas de textura
out vec4 FragColor;                       // cor do fragmento

void main(void){
    vec4 pos, geomDir, geomPos, dst, tex, src;
    float t, scalar;
    mat4 mvi = viewMatrixInv;

    geomPos = vec4(texCoord, 1.0);             // ponto de entrada do volume
    geomDir = normalize(mvi * vec4(0.0,0.0,-1.0,0.0)); // direção do raio
                                                 // paralelo ao eixo de projeção
    t = 2.0;
    pos = geomPos + geomDir*t;            // inicia com um ponto mais distante
    scalar = texture(VOLUME, pos.xyz).rgba;  // ler a densidade
    src = texture (TRANSFERFUNCTION, scalar).rgba; // ler o atributo gráfico
    src = clamp(src, 0.0, 1.0);          // truncar os valores entre [0.0,1.0]
    dst = src;                           // cor inicial
    t -= step;                           // avança um passo para frente
    do {
      pos = geomPos + geomDir*t;          //próxima amostra ao longo do raio
      scalar = texture(VOLUME, pos.xyz).rgba; // obter a densidade da amostra
      src = texture (TRANSFERFUNCTION, scalar).rgba; // obter prop. óptica
      src = clamp(src, 0.0, 1.0);  // truncar o valor para [0,1]
      scalar = src.a*0.6;          // atenuar o fator de composição
      dst = mix(dst, src, scalar); // composição
      dst = clamp (dst, 0.0, 1.0); // truncar o valor para [0.,1.]
      if (dst.a >= 1.0)            // 1a. cond. de parada: satura a opacidade
        break;
      t -= step;
    } while (t >= 0.0);              // até chegar na face visível do volume
    FragColor = dst;                 // escrever a cor final do fragmento
}
```

No cabeçalho deste código há a declaração de uma série de variáveis que são utilizadas ao longo das instruções. Mais especificamente, 4 variáveis globais do tipo

uniform, MODELVIEW, step, VOLUME e TRANSFERFUNCTION, e 2 variáveis de intercomunicação entre os estágios do fluxo de renderização, texCoord e FragColor.

A execução do *shader* dvr.frag depende dos acessos a dois conjuntos de dados definidos pelo cliente: as densidades das amostras VOLUME e a função de transferência TRANSFERFUNCTION. Para tirarmos proveito do mecanismo de filtragem e de interpolação no preenchimento das primitivas gráficas, estes dois conjuntos são encapsulados como objetos de textura na CPU e transferidos para GPU como duas unidades de textura distintas, 1 e 2:

```
void wxGLCanvasSubClass::transfereTexturas () {
  GLuint texId[2];             // IDs das 2 texturas
  GLubyte data[...];           // dados do volume de dimensoes <dimx*dimy*dimz>
  GLubyte tf[...];             // dados da funcao de transf de dimensao <dim>

  glGenTextures(2, texId);          // gerar dois nomes de textura

  data = carregueVolume();          // ler volume de dados
  glBindTexture(GL_TEXTURE_3D, texId[0]);
  glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA,//armazenar na GPU como uma textura 3D
     dimx, dimy, dimz, 0, GL_RGBA,  // de formato (R,G,B,A) e de dimensão
     GL_UNSIGNED_BYTE, data);       // <dimx x dimy x dimz> os elementos de
                                    // <data>, de tamanho GL_UNSIGNED_BYTE
                                    // e no formato (R,G,B,A)
  tf = carregueTF();                // ler a funcao de transferencia
  glBindTexture(GL_TEXTURE_1D, texId[1]);
  glTexImage1D(GL_TEXTURE_1D, 0, GL_LUMINANCE, // transferir para GPU uma
      dim, 0, GL_LUMINANCE,        //textura 1D de formato escalar e de dimensão
      GL_UNSIGNED_BYTE, tF);       // <dim> os elementos de <tf>, de tamanho
                                  // GL_UNSIGNED_BYTE e no formato escalar
}
```

Retornando ao programa dvr.frag, a variável de saída FragColor representa o local onde é armazenada a cor final do *pixel*, enquanto a variável de entrada texCoord indica as coordenadas de textura que o *shader* de fragmentos espera receber do *shader* de vértices. Como as coordenadas de textura são um dos atributos de vértices estabelecidos pelo cliente, a CPU, precisamos prover um *shader* de vértices que simplesmente passa este atributo da CPU para o *shader* de fragmentos. O seguinte código dvr.vert executa exatamente esta tarefa:

```
#version 150
                                  //Definidos pelo cliente:
uniform mat4 viewMatrix, projMatrix;//matrizes MODELVIEW e PROJECTION
in vec3 position;                 //coordenadas espaciais do vértice
in vec3 texCoord0;                //coordenadas de textura do vértice
out vec3 texCoord;

void main (void)
{
  vec4 tmp;
  tmp = vec4(position, 1.0);                    // converter para coord.homog.
  gl_Position = projMatrix * viewMatrix * tmp;  // projeção do ponto
  texCoord = texCoord0 ;                        // transferir coord. de textura
}
```

Observe que neste *shader* de vértices temos duas variáveis de entrada, position e texCoord0, cujos valores o *shader* espera receber da CPU. Como estes valores são passados da CPU para GPU? Qual é o mecanismo de vinculação dos endereços destes valores nos dois espaços de endereços?

Vimos na seção 1.3.3.2 que é desejável que todos os atributos dos vértices sejam transferidos para o espaço de endereços da GPU antes de iniciar o fluxo de renderização.

Podemos criar um *buffer object* para cada conjunto de atributos. Em particular, para o nosso exemplo, criamos três *buffer objects*: as coordenadas espaciais dos vértices agrupadas no objeto vbo[0], as coordenadas de textura associadas aos vértices encapsuladas no objeto vbo[1] e os índices dos vértices que definem as faces em vbo[2]. E para manipular todos estes atributos como um único objeto na CPU, podemos ainda associar estes *buffer objects* a um *vertex array object*, como mostra o seguinte trecho de código:

```
void wxGLCanvasSubClass::crieVertexArray () {
  float vertices[][3]={...};        // coordenadas espaciais dos vértices
  float texCoords0[][3]={...};      // coordenadas de textura dos vértices
  float quadIndices[24]={...};      // índices dos vértices para formar as faces
  GLuint vao, vbo[3];

  glGenVertexArrays(1, &vao);       // gerar o nome para vertex array object
  glBindVertexArray(vao);           // <vao> passa a ser objeto corrente
  glGenBuffers(3, vbo);             // gerar nome para os 3 buffer objects
  glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); // associar vbo[0] a <vao>
  glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
                                   // transferir seus dados para GPU
  glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); // associar vbo[1] a <vao>
  glBufferData(GL_ARRAY_BUFFER, sizeof(texCoords0), texCoords0, GL_STATIC_DRAW);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[2]); // associar vbo[2] a <vao>
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(quadIndices), quadIndices,
                                  GL_STATIC_DRAW);
  glBindVertexArray(0);           // disassociar <vao> como objeto corrente
}
```

Tanto vbo quanto texId são referências utilizadas na CPU para nomear espaços de memória gerenciados pela GPU. Ao executar um *shader*, por exemplo o *shader* de vértices, como este *shader* sabe que os valores da variável position estão no espaço designado por vbo[0]? Novamente, a CPU, o cliente, é quem intermedia o processo antes de iniciar o fluxo de renderização.

Primeiro, o cliente usa a função glGetAttribLocation ou a função glGetUniformLocation para obter as referências das variáveis dentro do objeto de programa prog, assim que forem compilados e ligados os objetos de *shader* associados a ele. A primeira função, glGetAttribLocation, é para ter as referências das variáveis específicas do *shader* de vértices, e a segunda para variáveis compartilhadas pelos *shaders* de vértices e de fragmentos, conforme o trecho de código que se segue:

```
void wxGLCanvasSubClass::instaleShaders () {
  GLuint vertexLoc, texoLoc, projMatrixLoc, viewMatrixLoc,
           viewMatrixInvLoc, stepLoc;

  prog = carregueShaders();             // carregue, compile e ligue shaders
  vertexLoc = glGetAttribLocation(prog,"position"); // obter as referências
  tex0Loc = glGetAttribLocation(prog,"texCoord0");  // das variaveis nos
  projMatrixLoc = glGetUniformLocation(prog, "projMatrix"); // shaders
  viewMatrixLoc = glGetUniformLocation(prog, "viewMatrix"); // dvr.frag e
  viewMatrixInvLoc = glGetUniformLocation(prog, "viewMatrixInv");// dvr.vert
  stepLoc = glGetUniformLocation(prog,"step");
  tex0Loc = glGetUniformLocation(prog,"VOLUME");
  tex1Loc = glGetUniformLocation(prog,"TRANSFERFUNCTION");
}
```

Segundo, o cliente atribui os valores às referências para serem utilizados nos *shaders*. Isso deve ser realizado depois de ativar o objeto de programa prog com glUseProgram e antes de iniciar o fluxo de renderização. Quando se trata da passagem dos valores que são compartilhados pelos dois *shaders*, utiliza-se a função glUniform*:

```
void wxGLCanvasSubClass::atribuaValores() {
  GLfloat  modelview[16], projection[16];
  GLfloat passo=0.5;

  glGetFloatv( GL_MODELVIEW_MATRIX, modelview);   // ler a matriz MODELVIEW
  glGetFloatv( GL_PROJECTION_MATRIX, projection); // ler a matriz PROJECTION
  inverseMatrix (modelview, inv_modelview);     // inverter a matriz MODELVIEW

  glUniformMatrix4fv(viewMatrixLoc, 1, GL_FALSE, modelview);// passar os valores
  glUniformMatrix4fv(projMatrixLoc, 1, GL_FALSE, projection);
  glUniformMatrix4fv(viewMatrixInvLoc, 1, GL_FALSE, inv_modelview);
  glUniform1f(stepLoc, passo);

  glActiveTexture(GL_TEXTURE0 + 1); // ativa unidade de textura 1
  glUniform1i(tex0Loc,1);           // passar referência da textura 3D
  glActiveTexture(GL_TEXTURE0 + 2); // ativa unidade de textura 2
  glUniform1i(tex1Loc,2);           // passar referência da textura 1D
}
```

E quando são os valores restritos ao *shader* de vértices e encapsulados num objeto de arranjo de vértices, a função glVertexAttribPointer é utilizada. Tendo tudo devidamente vinculado, a CPU só precisa chamar a função glDrawElements para desenhar as seis faces do *proxy geometry* a fim de obtermos imagens similares às mostradas nas figuras 1.18.(b) e 1.18.(c):

```
void wxGLCanvasSubClass::desenheProxyGeometry() {
  glUseProgram (prog);
  atribuaValores();
  glBindVertexArray(vao);                 // vincular a <vao>
  glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);  // associar vertexLoc com o ponteiro
  glEnableVertexAttribArray(vertexLoc);  // ao espaço do vbo[0] na GPU
  glVertexAttribPointer(vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, (const GLvoid *)0);
  glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);  // associar texoLoc com o ponteiro
  glEnableVertexAttribArray(tex0Loc);     // ao espaço do vbo[1] na GPU
  glVertexAttribPointer(tex0Loc, 3, GL_FLOAT, GL_FALSE, 0, (const GLvoid *)0);

  glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, (const GLvoid *)0);
                             // renderizar 6 facetas com 4 vértices em cada uma
                             // cujos atributos estão associados a <vao>
  glBindVertexArray(0);
}
```

Resta agora mostrar como alocar uma área na tela do monitor para exibir a imagem final com uso da GUI wxWidgets. Uma forma simples é instanciar um objeto do *widget* wxGLCanvas, conforme exemplifica o seguinte código adaptado dos exemplos disponíveis em [62]. Para um conhecedor da linguagem C/C++, o trecho de código pode causar uma certa estranheza por falta da tradicional função main(). Ela não foi removida (nem pode)! A função main() está simplesmente embutida no macro IMPLEMENT_APP(MyApp):

```
#include <wx/wx.h>
#include <wx/glcanvas.h>
#ifndef WIN32
#include <process.h>
#include <io.h>
#endif
class wxGLCanvasSubClass: public wxGLCanvas {
        void desenheProxyGeometry();
        void atribuaValores();
        void instaleShaders();
        void crieVertexArray();
        void transfereTexturas;
```

```
public:
   wxGLCanvasSubClass(wxFrame* parent);
   void Paintit(wxPaintEvent& event);
protected:
   GLuint prog;
   DECLARE_EVENT_TABLE()
};

BEGIN_EVENT_TABLE(wxGLCanvasSubClass, wxGLCanvas)
    EVT_PAINT    (wxGLCanvasSubClass::Paintit)
END_EVENT_TABLE()

wxGLCanvasSubClass::wxGLCanvasSubClass(wxFrame *parent)
:wxGLCanvas(parent, wxID_ANY,  wxDefaultPosition, wxDefaultSize, 0, wxT("GLCanvas")){
   int argc = 1;
   char* argv[1] = { wxString((wxTheApp->argv)[0]).char_str() };

   transfereTexturas();
   crieVertexArray();
   instaleShaders();
}
void wxGLCanvasSubClass::Paintit(wxPaintEvent& WXUNUSED(event)){
   desenheProxyGeometry();
}

class MyApp: public wxApp
{
   virtual bool OnInit();
   wxGLCanvas * MyGLCanvas;
};

IMPLEMENT_APP(MyApp)     // macro para inicializar wxWidgets inclusive main()

bool MyApp::OnInit()
{
    wxFrame *frame = new wxFrame((wxFrame *)NULL, -1,  wxT("Hello GL World"),
                                 wxPoint(50,50), wxSize(200,200));
    new wxGLCanvasSubClass(frame);

    frame->Show(TRUE);
    return TRUE;
}
```

Devido às limitações de espaço, omitimos vários trechos do código. Focamos somente no mecanismo pouco convencional para implementar um algoritmo de DVR: transferência de dados entre os dois espaços de endereçamento, a passagem dos valores ou dos endereços dos valores do cliente para o servidor e alocação de uma área de exibição na tela do monitor. À primeira vista, o procedimento é pouco intuitivo e desnecessariamente complicado. Este é o ônus pelo alto desempenho das GPUs. Tendo em mente a organização de memória em GPUs, cabe ao programador decidir onde e como armazenar os dados para que sejam otimizados os seus acessos. Para uma completa visão da estrutura de um aplicativo envolvendo CPU e GPU, como compilação e linkagem dos *shaders*, recomendamos a leitura da referência [24], mesmo que a GUI seja GLUT e a sintaxe dos seus códigos seja da API OpenGL 2.0/GLSL 1.10.

1.3.4. Fluxo de Visualização

Antes de serem renderizados, os dados gerados pelos equipamentos de aquisição podem passar ainda por três estágios de transformações antes de entrar no fluxo de renderização: importação, pré-processamento de imagens (filtragem e aprimoramento), e mapeamento [52]. A figura 1.25 mostra a relação destes estágios num fluxo de visualização de

imagens médicas.

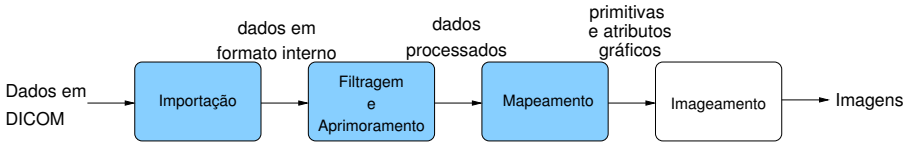


Figura 1.25. Fluxo de renderização integrado ao fluxo de visualização.

O estágio de importação de dados consiste essencialmente em ler os dados originais, tipicamente no formato DICOM, e estruturá-los no formato compatível com os métodos implementados no fluxo de visualização. Perda de dados neste processo pode comprometer não só a qualidade de imagens geradas como a qualidade de diagnóstico. Pois, no caso de imagens médicas, cada *voxel* é uma informação preciosa do interior do corpo humano que pode indiciar alguma lesão. Portanto, é importante que a conversão seja 1:1, preservando todos os dados originais, e deixar que os especialistas decidam o que visualizar.

A decisão do quê e de como visualizar os dados ocorre no estágio de pré-processamento. É neste estágio que são executados algoritmos específicos de processamento de imagens médicas para, por exemplo, filtrar ruídos, realçar regiões de interesse e segmentar órgãos em análise. Veremos na seção 1.4 que é nesta etapa em que se aplicam os algoritmos de exploração, a fim de integrar a inteligência humana em processamentos que são ainda considerados complexos para neuroimagens.

Decididos o quê ver dos dados originais e como vê-los, precisamos então adequá-los ao formato que o fluxo de renderização consegue processar. A este estágio chamamos mapeamento – mapeamento de dados selecionados em primitivas e atributos gráficos. Funções de transferência de densidades de tecidos para propriedades ópticas, introduzidas na seção 1.3.3.1, é um exemplo de mapeamento. Vimos ainda na seção 1.3.3.3 que, dependendo da técnica escolhida para renderização, os dados que são passados para o fluxo de renderização podem ser malhas poligonais ou volumes de amostras e conversões entre os formatos de dados podem ser necessárias. No nosso caso, optamos pela técnica DVR que tem diretamente os dados volumétricos como entrada.

1.4. Visualização Exploratória

Para que uma visualização seja efetiva em aplicações médicas, os dados visualizados devem ser elementos ativos no sentido de que cada *voxel* seja capaz de responder apropriadamente às ações de um médico sobre ele [37]. De acordo com Ware, uma visualização provida de interações em diferentes detalhes é conhecida como **visualização exploratória**. É um processo constituído de vários laços de realimentação que podem ser classificados, conforme o seu nível de abstração, em [55]:

laço de solução de problema, no qual o especialista complementa a alta capacidade de processamento do computador com o seu poder cognitivo;

laço de exploração e navegação, no qual o especialista complementa a capacidade do computador em síntese de imagens de alta qualidade com a sua alta capacidade de

reconhecimento e discernimento das principais características dos dados;

laço de manipulação direta, no qual o especialista complementa a capacidade do computador em renderizar um mesmo objeto sob distintos pontos de vista com a sua capacidade de manusear um objeto de interesse.

Um ingrediente fundamental em interações homem-máquina é a realimentação visual imediata das ações do usuário, pois ajuda-o a avaliar se suas ações são processadas conforme a sua expectativa. Isso proporciona maior senso de controle e aumenta a acurácia e a precisão na solução de um problema. Mostramos nesta seção uma forma eficiente de integrar a geometria do objeto de realimentação, como *cursor*, a um volume de dados com uso da API OpenGL 3.0 [21].

Nesta seção expomos ainda algumas tarefas de interação mais usuais em aplicações médicas [37]: edição de funções de transferência que mapeiam os valores escalares dos dados em atributos gráficos [33, 47]; alteração dos parâmetros como giro [49] e redimensionamento do volume de dados [7]; e cortes que revelam a estrutura interna do volume de dados [56, 14, 60]. Estas operações podem ser locais, específicas para um subconjunto de *voxels*; ou globais, envolvendo todos os dados do volume. Quando se trata de operações locais, é fundamental dispor de um procedimento de correspondência entre as coordenadas dos *pixels* e as coordenadas dos *voxels* projetados neles. Portanto, é dada antes uma noção das técnicas de mapeamento de 2D para 3D e as suas implementações via API OpenGL 3.0 realizadas pelo grupo de pesquisa da Faculdade de Engenharia Elétrica e de Computação da Unicamp [1].

A figura 1.26 esquematiza o ciclo de interação com um *mouse* convencional para controlar o movimento de um *cursor* 3D cuja posição $P_c = (x_c, y_c, z_c)$ é realimentada visualmente através de um objeto gráfico no espaço nativo do volume de dados. Este objeto gráfico é denominado objeto de realimentação. Veremos nas seções 1.4.1 e 1.4.2 que cada ciclo de interação envolve dois passos de fluxo de renderização. No primeiro passo o mapa de profundidade do objeto de realimentação é gerado e é armazenado no *framebuffer object*, e no segundo passo a imagem com o objeto de realimentação integrado é gerada e gravada na memória de quadro para ser exibida. Ao repetir este processo com uma taxa de quadros em torno de 30 quadros por segundo, ou simplesmente 30 fps, o sistema produz uma sensação de fluidez na sua resposta ao usuário.

1.4.1. Integração do Objeto de Realimentação

O objeto de realimentação mais conhecido em aplicativos 2D é o *cursor* que indica a posição do dispositivo de entrada sob controle do usuário na tela de um monitor. Uma extensão natural dele para aplicativos 3D é o *cursor 3D* que indica a posição do mesmo em uma cena 3D. Embora funcionalmente similares, há uma grande diferença na forma como a geometria dos dois objetos deva ser renderizada para evitar interpretações equivocadas. Enquanto um *cursor* convencional desliza sobre todos os objetos exibidos na tela, mantendo a mesma profundidade, a profundidade de um *cursor 3D* muda dinamicamente. Surge então a pergunta: "como integrar a sua geometria ao volume de dados preservando a percepção da sua profundidade?"

Engel e os seus colegas propuseram uma variante do algoritmo DVR que usa o

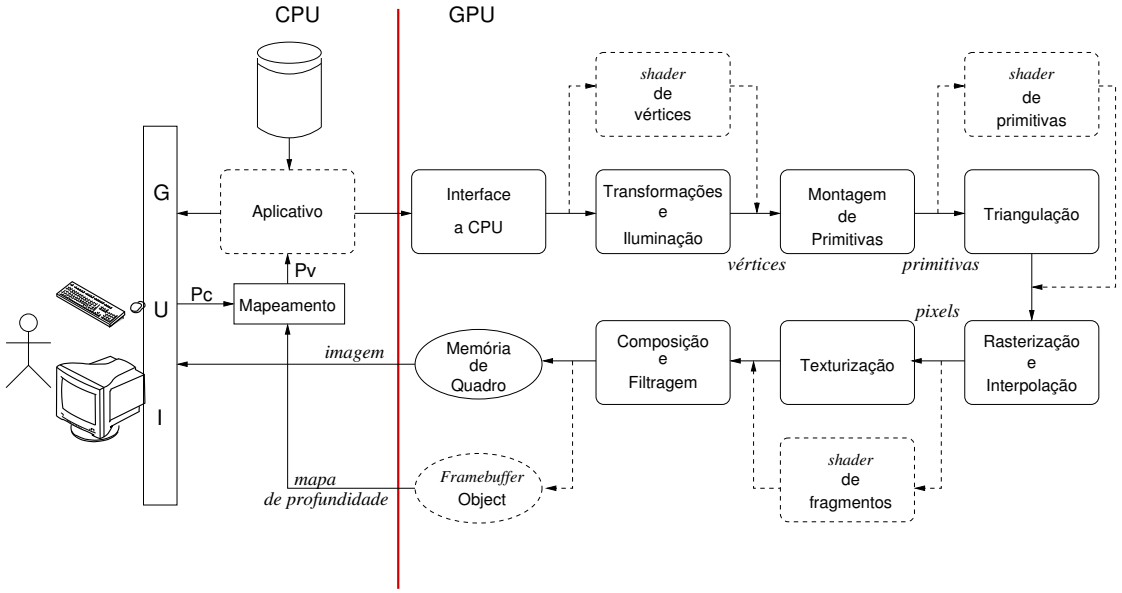


Figura 1.26. Fluxo de interação.

mapa de profundidade da geometria do objeto de realimentação para controlar o momento de parada no percurso dos raios de projeção [21]. Dadas as coordenadas do *cursor* $P_v = (x_V, y_V, z_V)$, é gerado o mapa de profundidade com o objeto de realimentação centrado em nele. Este mapa é passado como uma terceira textura DEPTH_MAP_OBJ ao *shader* dvr.frag:

```
uniform sampler2DRect DEPTH_MAP_OBJ;
```

e o valor z do objeto de realimentação em cada fragmento gl_FragCoord é acessado

```
z = texture(DEPTH_MAP_OBJ, gl_FragCoord.xy);
```

e incluído na condição de parada

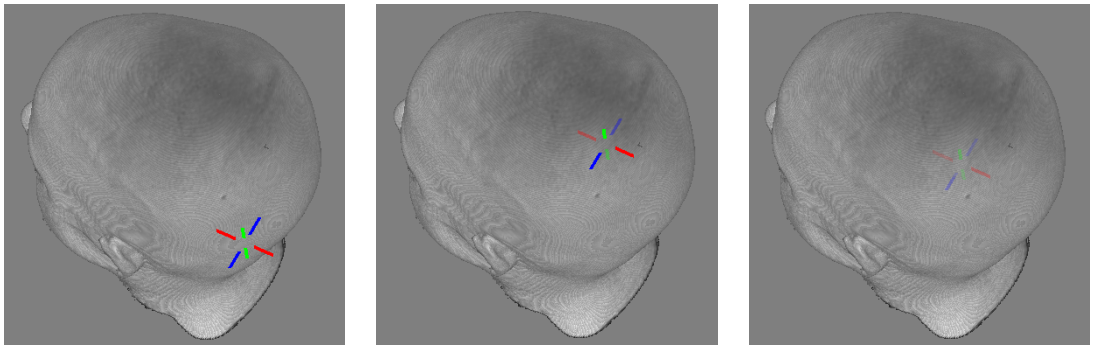
```
(t >= 0.0 && pos.z < z)
```

Quando o raio de projeção encontra o objeto de realimentação dentro do volume de dados, a sua cor (cor_obj.r, cor_obj.g, cor_obj.b) é misturada à cor do volume de dados acumulada dst com o fator de ponderação cor_obj.a, causando a percepção de "estar dentro":

```
dst = mix(dst, cor_obj, cor_obj.a);
```

A figura 1.27 ilustra o resultado visual do algoritmo aplicado em um *cursor 3D*, quando ele se encontra fora, sobre e dentro do volume de dados.

O mapa de profundidade do objeto de realimentação, por sua vez, pode ser gerado via o mecanismo de *offscreen rendering*. Para isso, é necessário alocar um espaço de memória na GPU, associá-lo a um *framebuffer object* fbo:



(a) (b) (c)

Figura 1.27. Realimentação visual de um *cursor* 3D: (a) fora, (b) sobre, e (c) dentro do volume de dados.

```
GLuint texture_obj[2];
GLuint VIEWPORT_HEIGHT, VIEWPORT_WIDTH; // dimensões da janela

// alocar 2 áreas de textura do tamanho da janela sem inicialização:
// color buffer (no mínimo, 1) e depth buffer
glGenTextures(2, texture_obj);
glBindTexture(GL_TEXTURE_RECTANGLE, texture_obj[0]);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_RECTANGLE, 0, GL_RGB,
     VIEWPORT_WIDTH, VIEWPORT_HEIGHT, 0, GL_RGB,
     GL_FLOAT, 0);
glBindTexture(GL_TEXTURE_RECTANGLE, texture_obj[1]);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_RECTANGLE, 0, GL_DEPTH_COMPONENT,
     VIEWPORT_WIDTH, VIEWPORT_HEIGHT, 0, GL_DEPTH_COMPONENT,
     GL_FLOAT, 0);

// associar o espaço a um framebuffer object
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
     GL_TEXTURE_RECTANGLE, texture_obj[0], 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
     GL_TEXTURE_RECTANGLE, texture_obj[1], 0);
glBindFramebuffer(GL_FRAMEBUFFER, 0); // retornar para o buffer convencional
```

e redirecionar o fluxo de renderização para fbo com a escrita do seu *buffer* de profundidade habilitada e a escrita nos seus *buffers* de cor desativada. Só depois dessa sequência de configuração é disparado o fluxo de renderização do objeto de realimentação para gerar o seu mapa de profundidade:

```
glEnable(GL_DEPTH_TEST);      // habilita a escrita do buffer de profundidade
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
glDrawBuffer(GL_NONE);       // desabilita a escrita dos buffers de cor
desenheObjetoRealimentacao();
glBindFramebuffer(GL_FRAMEBUFFER, 0);  // retornar à memória de quadro
```

A vinculação da unidade de textura GL_TEXTURE0 + texture_obj[1], que contém o mapa de profundidade do objeto de realimentação, com a variável DEPTH_MAP_OBJ declarada na nova versão de dvr.frag segue o mesmo procedimento apresentado na seção 1.3.3.3.

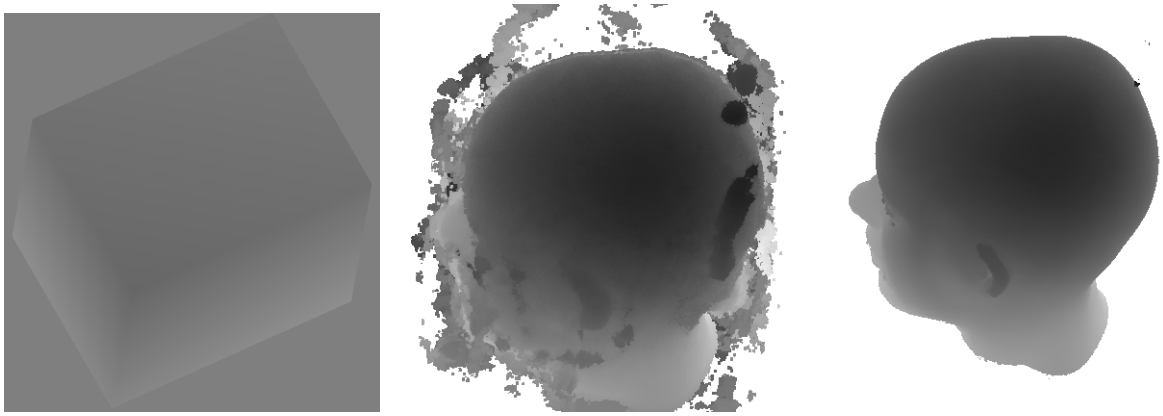
1.4.2. Mapeamento de 2D para 3D

O procedimento apresentado na seção 1.4.1 parte da premissa de que a posição do *cursor* seja dada em coordenadas 3D. No entanto, os dispositivos convencionais 2D provêem somente as coordenadas dos *pixels* sobre os quais eles deslizam. Precisamos, em vista das tarefas a serem realizadas, achar uma correspondência entre as coordenadas 2D do dispositivo de entrada, $P_c = (x_c, y_c)$, e as coordenadas 3D, $P_V = (x_V, y_V, z_V)$, do espaço em que os objetos do nosso interesse se encontram.

Em relação ao movimento do *cursor*, podemos considerar duas situações: (1) deslizar sobre a superfície do volume de dados; e (2) mover livremente no espaço onde o volume está inserido. Para a primeira situação, precisamos mapear as coordenadas do *pixel* P_c, sobre o qual o *cursor* está localizado, em coordenadas P_V da amostra da superfície do volume projetada neste *pixel*. E para a segunda situação, precisamos conceber uma forma que diferencie os movimentos em profundidade dos movimentos sobre o plano de imagem, a fim de que o usuário tenha a flexibilidade de "navegar" entre os *voxels*.

1.4.2.1. Movimentos Restritos sobre uma Superfície

De acordo com Wu *et al.* podemos mapear P_c numa amostra P_v da superfície do volume de dados, se gerarmos antecipadamente o mapa de profundidade da superfície visível [61]. Mas, com o procedimento apresentado na seção 1.4.1, que consiste em gerar o mapa de profundidade com a escrita de *buffer* de profundidade habilitada, teremos como resultado o mapa de profundidade do *proxy geometry* mostrada na figura 1.28.(a), e não do volume de dados correspondentes às amostras efetivas dos tecidos. O mapa obtido teria pouca utilidade para o nosso propósito. O nosso objetivo é manipular diretamente sobre a superfície das amostras visíveis sem alterar o seu interior, como as apresentadas na figura 1.28.(c). Para isso, precisamos alterar o mecanismo de geração do mapa de profundidade do fluxo fixo. Como o estágio de teste de profundidade não é programável, propomos desabilitá-lo e gerar o nosso mapa de profundidade via o *buffer* de cor texture_obj[0]:



(a) (b) (c)

Figura 1.28. Mapa de profundidade de: (a) *proxy geometry*; **(b) volume de dados com ruído; e (c) volume de dados sem ruído.**

```
glDisable(GL_DEPTH_TEST);                // não escreve no z-buffer
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
glDrawBuffer(GL_COLOR_ATTACHMENT0);      // escrever no color buffer 0
desenheProxyGeometry();
glDrawBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

O *shader* dvr.frag deve ser também ligeiramente modificado. Ao invés de retornar a composição das cores das amostras ao longo de cada raio de projeção, ele retorna a coordenada z da primeira amostra com densidade diferente de zero. Os valores de profundidade escritos no *framebuffer object* são então transferidos para o espaço de endereços da CPU:

```
GLfloat depthmap[VIEWPORT_WIDTH*VIEWPORT_HEIGHT];

glBindFramebuffer(GL_FRAMEBUFFER, fbo);
glReadBuffer(GL_COLOR_ATTACHMENT0);      // transferir o conteúdo do color buffer
glReadPixels(0, 0, VIEWPORT_WIDTH, VIEWPORT_HEIGHT, GL_RED, GL_FLOAT, depthmap );
                                         // para o endereço depthmap da CPU
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Para cada *pixel* $P_c = (x_c, y_c)$ acessamos a profundidade z_c = depthmap$[x_c, y_c]$ da primeira amostra projetada nele, reconstruindo assim as suas coordenadas 3D no espaço do dispositivo. E, com uso das matrizes GL_MODELVIEW_MATRIX, GL_PROJECTION_MATRIX e GL_VIEWPORT, podemos recuparar as coordenadas P_v no espaço do volume de dados através da função gluUnproject:

```
void mapa_2D_3D (float x_c, float y_c, float z_c, float *x_v, float *y_v, float *z_c) {
  double modelview[16], projection[16];
  int viewport[4];

  glGetDoublev( GL_PROJECTION_MATRIX, projection );
  glGetDoublev( GL_MODELVIEW_MATRIX, modelview );
  glGetIntegerv( GL_VIEWPORT, viewport );

  gluUnProject( x_c, viewport[3]-y_c, z_c, modelview,
                projection, viewport, x_v, y_v, z_v );
}
```

Vale observar aqui que há muito ruído nas imagens originais. Se aplicássemos diretamente o procedimento descrito sem pré-processamento, obteríamos mapas de profundidade similares aos apresentados na figura 1.28.(b). Por isso, adicionamos no dvr.frag um limiar controlável pelo usuário, somente acima do qual inicia-se o processo de composição. A maioria das imagens mostradas neste capítulo é, de fato, gerada com este limiar.

1.4.2.2. Movimentos Espaciais

A forma mais conhecida de posicionar o *cursor* num espaço 3D é através da decomposição deste espaço em três vistas ortogonais, xy, yz e xz. Nielson and Olsen apresentam uma técnica de manipulação direta no espaço 3D, particionando a tela em 6 regiões correspondentes aos 6 semi-eixos do referencial. Os deslocamentos do *cursor* afetarão somente a

coordenada correspondente à região em que ele se encontra [40]. Movimentos bruscos podem gerar resultados pouco intuitivos.

Mesquita e Wu decompõem o movimento do *cursor* em dois planos, *xy* e *xz*, e sugerem o uso de uma tecla para diferenciar os dois planos de movimento [36]. Quando os movimentos forem no plano *xy*, considera-se como a coordenada z_c do *cursor* o valor da iteração anterior. E quando os movimentos forem no plano *xz*, aplicamos a técnica proposta por Vidalón para estimar a variação da coordenada Δz_c em termos das variações das coordenadas Δx_c e Δy_c em relação à iteração anterior [53]

$$\Delta z_c = \frac{\sqrt{\Delta x_c{}^2 + \Delta y_c{}^2}}{4H},$$

sendo H o diagonal da janela de exibição.

Obtidas as coordenadas do *cursor* no espaço do dispositivo, utilizamos a função mapa_2D_3D, dada na seção 1.4.2.1, para recuperar as suas coordenadas no espaço nativo do volume de dados.

1.4.3. Tarefas de Interação

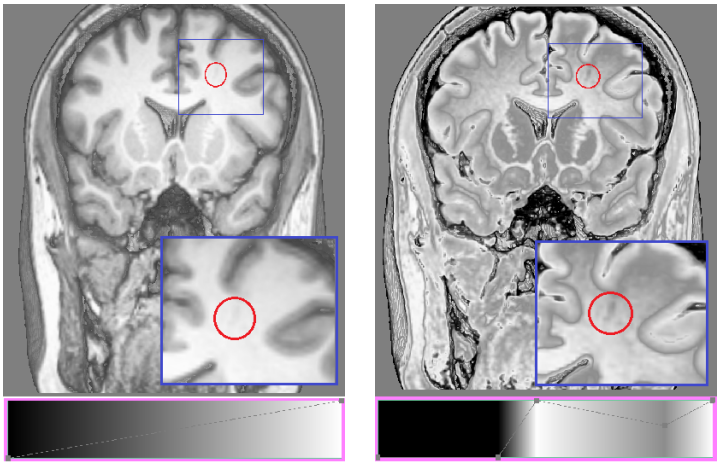
Dispondo do procedimento de renderização dos objetos de interação, de um procedimento de mapeamento de 2D para 3D e de uma biblioteca de GUI, a implementação de uma série de tarefas de interação que requer como referência alguns *voxels* do volume de dados se torna simples. Ilustramos nesta seção a implementação de três tarefas de interação com uso das funções da biblioteca wxWidgets [2] e da API OpenGL 3.0: o editor de funções de transferência, as transformações geométricas do volume de dados e reformatação multiplanar.

1.4.3.1. Editor de Funções de Transferência

Conforme vimos na seção 1.3.3.1, a função de transferência é responsável pelo mapeamento das densidades das amostras em níveis de cinza ou cores. Dentre as quatro classes de funções de transferência listadas pelo Pfister e seus colegas [44], Vidalón mostrou no seu trabalho que a técnica por tentativa-e-erro aplicada em funções de transferência 1D não-monotônicas consegue ajudar os médicos a distinguir lesões sutis em imagens de ressonância magnética [53]. Isso se deve principalmente ao fato do sinal de resposta da ressonância magnética não depender apenas da densidade de prótons como em tomografias computadorizadas, mas de outras características do tecido. Por um lado, as imagens de RM permitem distinguir diferentes tecidos moles. Por outro, esta alta sensibilidade dificulta a aplicação de funções de transferência multi-dimensionais na segmentação de distintas estruturas do cérebro como ocorre com as imagens de tomografia computadorizada. Além disso, como a percepção visual é algo subjetiva, ajustes interativos permitem que o usuário varie continuamente, a critérios individuais, os contrastes e os tons de cinza nas regiões suspeitas até que as lesões sutis fiquem perceptíveis.

A figura 1.29 ilustra o caso de uma lesão cortical focal sutil. Na figura 1.29(a) foi marcada uma pequena região suspeita de displasia cortical focal quase imperceptível com uma função de transferência monotônica. Podemos, no entanto, aumentar o contraste da

imagem utilizando uma função de transferência 1D não-monotônica. Com isso, detalhes da região lesionada tornaram-se mais perceptíveis, conforme a figura 1.29(b).



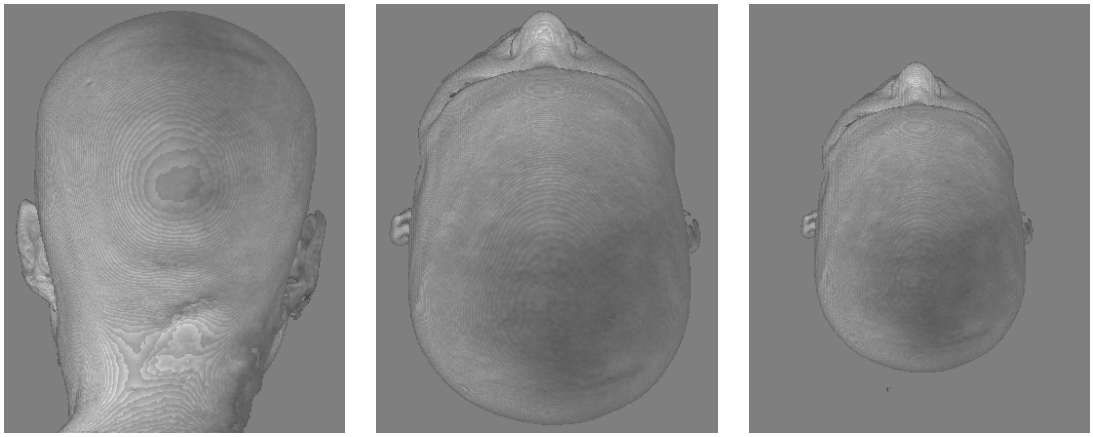
(a) (b)

Figura 1.29. Visualização de uma lesão sutil com: (a) uma função de transferência monotônica e (b) uma função de transferência não-monotônica.

As funções de transferência mostradas na figura 1.29 são funções lineares por parte, onde os valores na abscissa correspondem aos valores de densidade e os valores na ordenada aos níveis de cinza. Assim, as primitivas gráficas 2D, pontos e linhas, são suficientes como objeto de realimentação. Utilizamos as coordenadas $P_c = (x_c, y_c)$ do *cursor* para selecionar essas primitivas e manipulá-las. Os movimentos do *mouse* geram o evento EVT_MOTION da biblioteca wxWidgets e as coordenadas do seu *cursor* podem ser obtidas através do método wxMouseEvent::GetPosition da mesma biblioteca. A partir destas coordenadas, não só o nível de cinza y_c associado à densidade x_c pode ser alterado, como também os dois trechos da função de transferência que tem o ponto (x_c, y_c) como ponto-extremo podem ser modificados. E, de acordo com o fluxo da figura 1.26, uma nova imagem da função de transferência e uma do volume de dados são automaticamente geradas pelo Aplicativo após atualizar os atributos gráficos do objeto de realimentação e a textura correspondente à função de transferência. Como consequência desta realimentação, a consistência das informações visualizadas com as ações do usuário é assegurada.

1.4.3.2. Transformações Geométricas

Além da edição interativa das funções de transferência, rotação e mudança do fator de escala uniforme dos volumes de dado, como ilustra a figura 1.30, são também operações frequentes durante análise das neuroimagens. Quando a renderização de uma cena for muito lenta, é comum utilizar figuras geométricas mais simples como objetos de realimentação durante as interações. Como este não é o caso quando utilizamos o fluxo de renderização apresentado na seção 1.3.3.3, sugerimos utilizar o próprio volume de dados como o objeto de realimentação, poluindo assim menos a tela de exibição.



(a) (b) (c)

Figura 1.30. Transformações geométricas aplicadas em (a): (b) rotação em torno do eixo x e (c) redução do tamanho.

Uma forma de entrar o fator de escala s é através da roda do *mouse* cujo movimento é capturado como o evento EVT_MOUSEWHEEL da biblioteca wxWidgets. A direção e o ângulo relativo do giro podem ser lidos pela função wxMouseEvent::GetWheelRotation da mesma biblioteca. Com base nestes valores, o Aplicativo mostrado no fluxo da figura 1.26 determina a variação do fator de escala Δs, atualiza o fator através da expressão $s = s + \Delta s$ e dispara o fluxo de renderização para gerar a imagem do volume de dados com o novo fator de escala.

Quanto ao giro do volume de dados, a forma mais intuitiva de realizá-lo, sem histerese, é a técnica *arcball* proposta por Shoemake [49]. Esta técnica consiste em gerar, a partir de dois pontos v_0 e v_1 sobre uma esfera unitária de centro C, o quatérnio unitário $q = (x,y,z,w) = (\vec{v}sen\theta, cos\theta)$, onde $\vec{v}$ e 2θ são o eixo e o ângulo de rotação, respectivamente. A relação entre os pontos e o quatérnio é dada por

$$\begin{aligned} \vec{v}sen\theta &= (v_0 - C) \times (v_1 - C) \\ cos\theta &= (v_0 - C)\dot{(}v_1 - C). \end{aligned}$$

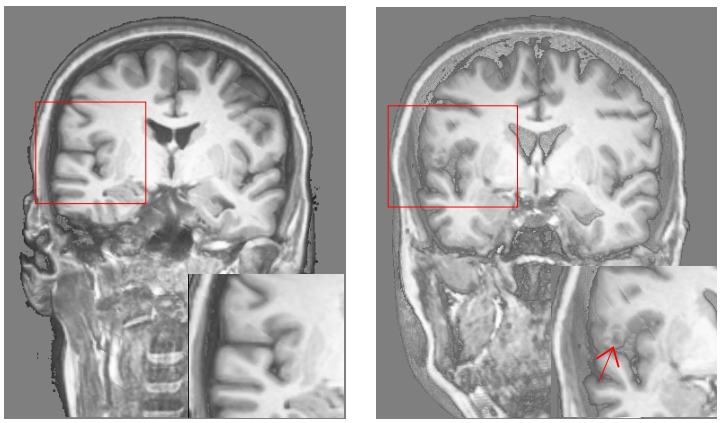
O quatérnio, por sua vez, pode ser convertido em uma matriz 4×4:

$$M = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw & 0 \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw & 0 \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Se utilizamos a estratégia descrita na seção 1.4.2.1 para mapearmos as duas posições subsequentes do *cursor* sobre a esfera unitária centrada no centróide do *proxy geometry*, temos os dois pontos requeridos pelo procedimento, v_0 e v_1. E o Aplicativo, indicado na figura 1.26, se encarregará de realizar os cálculos necessários para construir a matriz M e de disparar o fluxo de renderização de forma a gerar uma nova imagem com esta nova matriz de transformação geométrica. Ao fazer isso sucessivamente, ter-se-á a sensação de que o volume de dados gire junto com o movimento da mão do usuário.

1.4.3.3. Reformatação Multiplanar

Reformatação multiplanar é outra técnica amplamente aplicada na análise das neuroimagens. Como já explicada na seção 1.3.2, sob o ponto de vista clínico ela consiste em cortar o volume de dados com uma série de planos paralelos, permitindo investigar sob diversos ângulos de visão o interior do volume de dados de fatia em fatia. Este recurso pode ser decisivo no achado de lesões sutis, pois existem tecidos alterados que só são visíveis em um ângulo de visão específico, como ilustra a figura 1.31. Para um mesmo volume de dados, não se percebe nenhuma lesão no ângulo de visão da figura 1.31.(a), enquanto no ângulo mostrado pela figura 1.31.(b) uma lesão sutil no lobo temporal esquerdo torna aparente.



(a) (b)

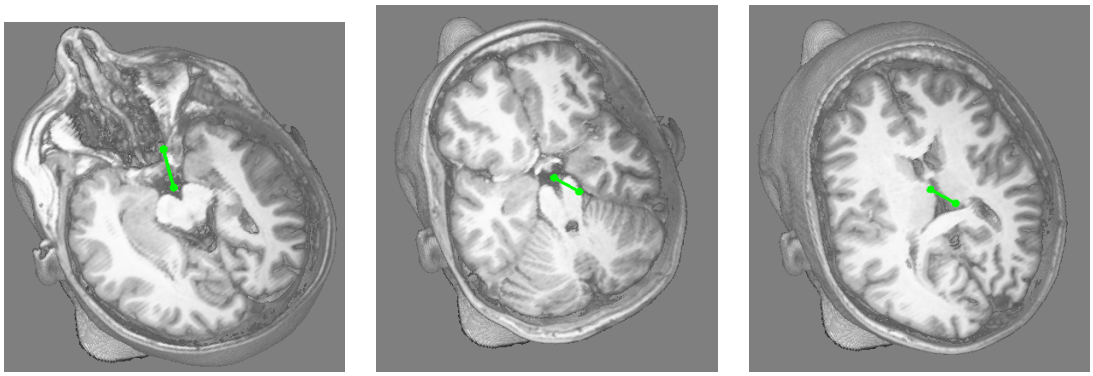
Figura 1.31. Visualização de uma lesão sutil em: (a) um ângulo de visão coronal e (b) um ângulo de visão ligeiramente oblíquo em relação às fatias coronais e axiais.

Para gerar este efeito visual com o *shader* dvr.frag, basta adicionar uma restrição: só entra na composição de cores da Eq. 1 ou da Eq. 2 as amostras que satisfazem a condição $n_x x+n_y y+n_z z+d<0$, sendo $n_x x+n_y y+n_z z+d=0$ o plano de corte. Isso reduz o nosso problema de reformatação multiplanar em especificação de quatro parâmetros para GPU: n_x, n_y, n_z e d.

Como (n_x,n_y,n_z) definem um vetor espacial, reutilizamos a técnica de rotação apresentada na seção 1.4.3.2 para girá-lo arbitrariamente em torno da intersecção P entre o plano e a reta paralela a (n_x,n_y,n_z) que passa pelo centróide do *proxy geometry*. Ou seja, as posições subsequentes do *cursor* são mapeadas sobre os pontos v_0 e v_1 de uma esfera de raio unitário com centro em P e a partir destes pontos construímos a matriz de rotação M. A cada nova orientação, o Aplicativo que aparece no fluxo da figura 1.26 dispara o fluxo de renderização para exibir uma outra imagem com os novos valores, e assim sucessivamente até o usuário parar de movimentar o *mouse*.

Em relação ao parâmetro d, podemos alterá-lo com a roda do *mouse* de maneira similar à variação do fator de escala explicada na seção 1.4.3.2. Na figura 1.32 apresentamos três tomadas da cabeça de um paciente para ilustrar a diferença visual gerada com a

variação em (n_x, n_y, n_z) e com a variação em d. Observe como o objeto de realimentação visual, representado pelo segmento em verde, é atualizado com as interações.



(a) (b) (c)

Figura 1.32. Reformatação multiplanar: variação de (n_x, n_y, n_z) entre (a) e (b); e variação de d entre (b) e (c).

1.5. Reflexões Finais

O cérebro é o órgão mais complexo e menos conhecido do corpo humano. Apesar de representar somente 2% da massa corporal, é dele que saem todos os sinais de controle das funções vitais. Pequenas lesões em certas áreas funcionais podem comprometer de forma drástica a qualidade de vida de um indivíduo e dos seus familiares. Apesar da sua importância, só com os avanços tecnológicos no final do século XX, os cientistas passaram a ter acesso ao interior dessa máquina poderosa *in vivo* e de forma não-invasiva, e analisar todos os seus detalhes de funcionamento. Doenças neurológicas antes incuráveis passaram a ter tratamento e um prognóstico animador.

Sem perda de generalidade, focamos neste capítulo uma anormalidade cerebral específica que afeta acerca 1% da população mundial – displasia cortical focal. Mostramos que, mesmo com todos os recursos tecnológicos que dispomos, é ainda uma tarefa desafiadora encontrar um foco epileptogênico quase imperceptível. O atual protocolo de diagnóstico envolve não só exames clínicos e tecnológicos, como também eventuais internações para realização de exames ictais. Muitas vezes, o resultado é muito aquém da expectativa. Esforços multidisciplinares são empenhados para melhorar este panorama.

Com o aumento da resolução espacial dos escaneadores, e consequente melhora na qualidade dos exames anatômicos e funcionais, ILAE reconhece a importância do estudo conjunto de imagens de diferentes modalidades no diagnóstico de um foco epileptogênico e na avaliação de prognóstico. Passa a ser um desafio para engenheiros como exibir de forma legível este grande volume de dados com o objetivo de ajudar neurocientistas a formular novas hipóteses e construir novos experimentos para verificá-las. Assim, espera-se desenvolver novas metodologias para diagnosticar com maior precisão o tipo da epilepsia e elaborar o seu plano de tratamento.

Nesta fase de experimentação e exploração, acreditamos que um eficiente visualizador interativo aumenta a fluidez dos trabalhos investigativos. Pois, ao permitir que um

neuroespecialista acompanhe os resultados intermediários de um processamento e, com base na sua experiência nem sempre programável, alterar o fluxo de controle de um processamento quando pertinente, o cientista pode rapidamente testar diferentes hipóteses e consolidá-las em procedimentos clínicos mais eficazes.

Mostramos neste capítulo alguns resultados promissores na área de visualização interativa para imagens médicas. Tais resultados só foram possíveis com o advento das GPUs, que provocaram grandes mudanças na concepção de programação em relação à época em que tínhamos somente CPUs para programar. Para ser auto-contido, foi dada uma breve introdução a GPUs e apresentamos alguns algoritmos de visualização implementados nestas unidades.

A área de pesquisa em visualização interativa para dados volumétricos médicos está ainda na sua infância. Há ainda muitos problemas em aberto. De acordo com Duncan [20], confiável integração de dados estruturais e funcionais em um sistema de neuronavegação promete ser uma forma promissora às intervenções cirúrgicas de epilepsia no futuro.

Para chegar lá, há ainda uma série de problemas a serem resolvidos. Em relação à visualização multimodal, são tópicos de pesquisa

- a melhor forma de combinar as informações oriundas de cada modalidade sem distorcê-las;
- uma maneira de contornar a limitação da memória de vídeo para renderizar "simultaneamente" diversos volumes de dados;
- a escolha de códigos de cor que sejam familiares aos médicos e, ao mesmo tempo, capazes de distinguir uma modalidade da outra; e
- aprimorar a fidelidade das imagens exibidas em relação aos valores das amostras mantendo a sua qualidade.

Em relação à interatividade do sistema, é fundamental que os métodos envolvidos apresentem um tempo de execução, que junto com a realimentação visual, esteja dentro da faixa classificada como interativa. Isso pode implicar na re-implementação dos algoritmos existentes adequando-os às tecnologias do estado-da-arte, entre as quais figuram as GPUs.

Finalmente, queremos frisar que os resultados almejados só conseguem ser alcançados com a sinergia entre as competências de formação bem distinta, com modo de trabalho e uso de termos técnicos diferentes. Esta multidisciplinaridade demanda das partes envolvidas um esforço adicional na troca de ideias e na adequação dos ritmos de trabalho. Por outro lado, saber que estes esforços possam vir a proporcionar maior conforto e bem-estar aos pacientes neurológicos é extremamente motivador. Além disso, os frutos colhidos, envolvendo imagens do interior do cérebro sob todos os ângulos, são fascinantes.

Agradecimentos As autoras agradecem ao Prof. Fernando Cendes pelo incentivo à elaboração deste texto, aos seus colegas pelo ambiente frutífero de trabalho e aos

seus orientandos pelo bom acolhimento que deram à versão original deste texto. Os trabalhos relatados neste texto contaram com o apoio financeiro da Fapesp (09/51425-6 e 2011/02351-0), do CNPq e da CAPES.

Referências

[1] MTK–Manipulation ToolKit. http://www.dca.fee.unicamp.br/projects/mtk, acessado em Março de 2013.

[2] wxWidgets, Cross-Plataform GUI Library, 2011. http://wxwidgets.org, acessado em Março de 2013.

[3] Song Ho Ahn. Opengl Framebuffer Object (FBO). http://www.songho.ca/opengl/gl_fbo.html, acessado em Março de 2013.

[4] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS'68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[5] Stephan Arens and Gitta Domik. A survey of transfer functions suitable for volume rendering. In *Volume Graphics*, pages 77–83, 2010.

[6] Isaac N. Bankman, editor. *Handbook of Medical Image Processing and Analysis*. Elsevier Inc., 2a. edition, 2010.

[7] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[8] M. A. Borges, L. L. Min, and C.A. Guerreiro. Urban prevalence of epilepsy: populational study in São José do Rio Preto, a medium-sized city in brazil. *Arquivos de Neuro-Psiquiatria*, (62):199–204, 2004.

[9] Matthew Brett and Michael Hanke. Defining the DICOM orientation. http://nipy.sourceforge.net/nibabel/dicom/dicom_orientation.html, acessado em Março de 2013.

[10] Hamish Carr and Nelson Max. Subdivision analysis of the trilinear interpolant. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):533–547, 2010.

[11] Loyd Case. Geforce 8800 GTX: 3D Architecture Overview, Novembro 2006. http://www.pcmag.com/article2/0,2817,2054120,00.asp, acessado em Março de 2013.

[12] Fernando Cendes. Neuroimagem na investigação das síndromes hemisféricas. Technical report, Latin-American Summer School on Epilepsy. http://www.lasse.med.br/mat_didatico/lasse1/textos/fernando02.html, acessado em Março de 2013.

[13] Fernando Cendes. Volumetry and diagnostic MRI evaluation of hippocampal sclerosis. Technical report, Latin-American Summer School on Epilepsy. http://www.lasse.med.br/mat_didatico/lasse1/textos/fernando04.html, acessado em Março de 2013.

[14] Hungi-Li Jason Chen, Faramarz F. Samavati, and Mario Costa Sousa. GPU-based point radiation for interactive volume sculpting and segmentation. *The Visual Computer*, 24(7):689–698, 2008.

[15] Markus Christen, Deborah A Vitacco, Lara Huber, Julie Harboe, Sara I Fabrikant, and Peter Brugger. Colorful brains: 14 years of display practice in functional neuroimaging. *Neuroimage*, 2013. http://www.biomedsearch.com/nih/Colorful-brains-14-years-display/23403183.html, acessado em Março de 2013.

[16] Roberto Covolan, Dráulio B. de Araújo, Antonio Carlos dos Santos, and Fernando Cendes. Ressonância magnética funcional: as funções do cérebro reveladas por spins nucleares. *Ciência e Cultura*, 56:40 – 42, 01 2004.

[17] Antonio Bittencourt de Almeida. Usando o computador para processamento de imagens médicas. *Informática Médica*, 1998. http://www.informaticamedica.org.br/informaticamedica/n0106/imagens.htm, acessado em Març de 2013.

[18] Luciano de Souza Queiroz and Maria Júlia Marques. Atlas de anatomia do cérebro em 12 cortes coronais ou frontais. http://www.fcm.unicamp.br/deptos/anatomia/bineucerebrocoronalindice.html, acessado em Março de 2013.

[19] G. Dougherty. *Digital Image Processing for Medical Applications*. Cambridge University Press, 2009.

[20] James S. Duncan. Imaging in the surgical treatment of epilepsy. *Nature reviews. Neurology*, 6(10):537–550, 2010.

[21] Klaus Engel, Markus Hadwiger, Joe Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-Time Volume Graphics*. AK Peters, 2006.

[22] Guillaume Flandin and Karl J. Friston. SPM2–Statistical Parametric Mapping. http://www.fil.ion.ucl.ac.uk/spm/software/spm8/, acessado em Março de 2013.

[23] Athinoula A. Martinos Center for Biomedical Imaging. FreeSurfer. http://surfer.nmr.mgh.harvard.edu/, acessado em Março de 2013.

[24] Joe Groff. An intro to modern OpenGL. http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Table-of-Contents.html, acessado em Março de 2013.

[25] The Khronos Group. GLUT–The Opengl Utility Toolkit. http://www.opengl.org/resources/libraries/glut/, acessado em Março de 20013.

[26] The Khronos Group. What's New in OpenGL 4.3. http://www.opengl.org/documentation/current_version/, acessado em Março de 2013.

[27] Milan Ikits, Joe M. Kniss, Aaron Lefohn, and Charles D. Hansen. Volume rendering techniques. In Randima Fernando, editor, *GPU Gems II: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 667–6692, Boston, MA, 2004. Addison Wesley.

[28] G. D. Jackson, S. F. Berkovic, B. M. Tress, R. M. Kalnins, G. C. Fabinyi, and P. F. Bladin. Hippocampal sclerosis can be reliably detected by magnetic resonance imaging. *Neurology*, 40(12):1869–75, 1990.

[29] M. Jenkinson, C.F. Beckmann, T.E. Behrens, M.W. Woolrich, and S.M. Smith. FSL. *NeuroImage*, 62:782–790, 2012. http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/, acessado em Março de 2013.

[30] Henrique Carrete Júnior. Avaliação de Epilepsia por Neuroimagem. Protocolos. *Revista Neurociências*, 10(2):94–98, 2002.

[31] Armin Kanitsar, Dominik Fleischmann, Rainer Wegenkittl, Petr Felkel, and Meister Eduard Gröller. CPR - Curved Planar Reformation. Technical Report TR-186-2-02-06, Institute of Computer Graphics and Algorithms, Vienna University of Technology, March 2002. http://www.cg.tuwien.ac.at/research/publications/2002/kanitsar-2002-CPRX/, acessado em Março de 2013.

[32] Sunhee Kim and James M. Mountz. SPECT Imaging of Epilepsy: An Overview and Comparison with F-18 FDG PET. *International Journal of Molecular Imaging*, 2011, 2011.

[33] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, Nov 2002.

[34] Li-Cheng. The Gui Toolkit, Framework Page. http://www.atai.org/guitool/, acessado em Março de 2013.

[35] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[36] Leonardo A. G. Mesquita and Shin-Ting Wu. Cursores 3D com uso de Dispositivos 2D. Technical report, Faculdade de Engenheria Elétrica e de Computação, Universidade Estadual de Campinas, 2001. ftp://ftp.dca.fee.unicamp.br/pub/docs/techrep/2001/lmesq-2001-cursor3d.pdf, acessado em Março de 2013.

[37] Konrad Mühler, Christian Tietjen, Felix Ritter, and Bernhard Preim. The Medical Exploration Toolkit: An Efficient Support for Visual Computing in Surgical Planning and Training. *IEEE Transactions on Visualization and Computer Graphics*, 16:133–146, January 2010.

[38] L. Nashef, D.R. Fish, Sander J.W., and Shorvon S.D. Incidence of sudden unexpected death in an adult outpatient cohort with epilepsy at a tertiary referral centre. *Journal of Neurology, Neurosurgery & Psychiatry*, (58):462–464, 1995.

[39] NEMA. Digital Imaging and COmmunications in Medicine, 2012. http://medical.nema.org/, acessado em Março de 2013.

[40] Gregory M. Nielson and Dan R. Olsen, Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, I3D'86, pages 175–182, New York, NY, USA, 1987. ACM.

[41] NIMH. Analysis of Functional NeuroImages. http://afni.nimh.nih.gov, acessado em Março de 2013.

[42] OpenGL. OpenGL, the cross-platform graphics API. http://www.opengl.org, acessado em Março de 2013.

[43] A. Palmini, I. Najm, G. Avanzini, T. Babb, R. Guerrini, N. Foldvary-Schaefer, G. Jackson, H.O. Lüders, R. Prayson, R. Spreafico, and H.V. Vinters. Terminology and classification of the cortical dysplasias. *Neurology*, 62(6 Suppl 3):S2–8, Março 2004.

[44] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L.S. Avila, K.M. Raghu, R. Machiraju, and Jinho Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16 –22, 2001.

[45] Digia plc. Qt Project. http://qt-project.org/, acessado em Março de 2013.

[46] Easy Software Products. Fltk – fast light toolkit. accessed in March 2013.

[47] Christof Rezk Salama, Maik Keller, and Peter Kohlmann. High-Level User Interfaces for Transfer Function Design with Semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12:1021–1028, Setembro 2006.

[48] Stefan Roettger, Michael Bauer, and Marc Stamminger. Spatialized Transfer Functions. In *EuroVis05: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 271–278, 2005.

[49] Ken Shoemake. Arcball rotation control. In *Graphics Gems IV*, pages 175–192, San Diego, CA, USA, 1994. Academic Press Professional, Inc.

[50] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics, 2005. Fourth International Workshop on*, pages 187–241, Junho 2005. http://cumbia.informatik.uni-stuttgart.de/eng/research/fields/current/spvolren/, acessado em Março de 2013.

[51] The GTK+ Team. The GTK+ Project. http://www.gtk.org/, acessado em Março de 2013.

[52] Alexandru C. Telea. *Data Visualization: Principles and Practice*. A K Peters, Ltd., Wellesley, MA, USA, 2008.

[53] José Elias Yauri Vidalón. Ferramentas interativas de auxílio a diagnóstico por neuroimagens 3d. Master's thesis, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2012.

[54] J.W. Wallis, T.R. Miller, C.A. Lerner, and E.C. Kleerup. Three-dimensional display in nuclear medicine. *IEEE Transactions on Medical Imaging*, 8(4):297–230, dec 1989.

[55] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, second edition, 2004.

[56] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):298–312, 2003.

[57] Graham Wideman. Orientation and Voxel-Order Terminology: RAS, LAS, LPI, RPI, XYZ and All That. http://www.grahamwideman.com/gw/brain/orientation/orientterms.htm, acessado em Março de 2013.

[58] Wikipedia. Comparison of nvidia graphics processing units. http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units, acessado em Março de 2013.

[59] World Health Organization. *Atlas: Epilepsy Care in the World*, Geneva, 2005.

[60] Shin-Ting Wu, Clarissa Lin Yasuda, and Fernando Cendes. Interactive curvilinear reformatting in native space. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):299–308, Fevereiro 2012.

[61] Shin-Ting Wu, José Elias Yauri Vidalón, and Lionis de Souza Watanabe. Snapping a Cursor on Volume Data. In *Proceedings of 24th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 109–116, Agosto 2011.

[62] wxWiki. WxGLCanvas. http://wiki.wxwidgets.org/WxGLCanvas, acessado em Março de 2013.

[63] Clarissa Lin Yasuda and Fernando Cendes. Neuroimaging for the prediction of response to medical and surgical treatment in epilepsy. *Expert Opinion on Medical Diagnostics*, 6(4):295–308, 2012.

[64] Qi Zhang, Roy Eagleson, and Terry M. Peters. Volume visualization: A technical overview with a focus on medical applications. *Journal of Digital Imaging*, 24(4):640–664, 2011.