Shell Rendering **com Fatoração** *Shear-Warp*

Leonardo Marques Rocha

Dissertação de Mestrado

Shell Rendering **com Fatoração** *Shear-Warp*

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Leonardo Marques Rocha e aprovada pela Banca Examinadora.

Campinas, 30 de agosto de 2002.

Prof. Dr. Alexandre Xavier Falcão
Instituto de Computação - Universidade Estadual de Campinas (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Rocha, Leonardo Marques

R582s Shell rendering com fatoração shear-warp / Leonardo Marques Rocha -- Campinas, [S.P. :s.n.], 2002.

Orientador : Alexandre Xavier Falcão

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Visualização. 2. Processamento de imagens. 3. Computação gráfica. 4. Fatoração matemática. I. Falcão, Alexandre Xavier. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Instituto de Computação
Universidade Estadual de Campinas

Shell Rendering com Fatoração *Shear-Warp*

Leonardo Marques Rocha

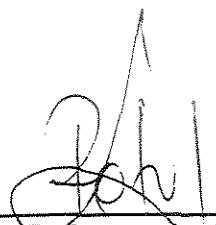
Julho de 2002

Banca Examinadora:

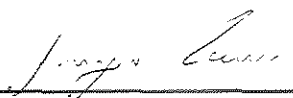
- Prof. Dr. Alexandre Xavier Falcão
  Instituto de Computação - Universidade Estadual de Campinas (Orientador)
- Prof. Dr. Roberto de Alencar Lotufo
  Faculdade de Engenharia Elétrica e Computação - Universidade Estadual de Campinas
- Prof. Dr. Sérgio Shiguemi Furuie
  InCor - Instituto do Coração da Universidade de São Paulo
- Prof. Dr. Fernando Cendes
  Faculdade de Ciências Médicas - Universidade Estadual de Campinas
- Prof. Dr. Neucimar J. Leite (Suplente)
  Instituto de Computação - Universidade Estadual de Campinas

TERMO DE APROVAÇÃO

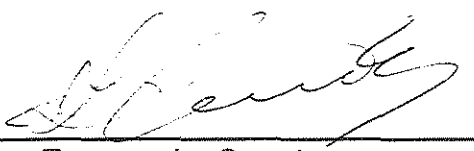
Tese defendida e aprovada em 30 de agosto de 2002, pela Banca Examinadora composta pelos Professores Doutores:



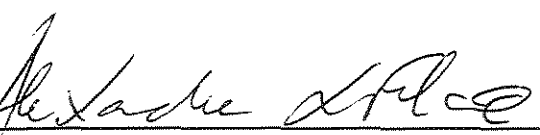
Prof. Dr. Roberto de Alencar Lotufo
FEEC - UNICAMP



Prof. Dr. Sérgio Shiguemi Furuie
InCor-USP



Prof. Dr. Fernando Cendes
FCM-UNICAMP



Prof. Dr. Alexandre Xavier Falcão
IC - UNICAMP

Agradecimentos

Gostaria de expressar meus agradecimentos a todos que, de uma forma ou de outra, contribuíram para a conclusão deste trabalho, que significou um grande passo na caminhada de minha vida:

Ao meu Pai, por ser um grande amigo e ter me proporcionado chegar onde estou.

À minha Mãe, pela sua dedicação e seu amor.

À minha Tia Vera, por seu carinho e preocupação.

Aos meus irmãos, por sua compreensão.

Ao meu orientador, Alexandre Xavier Falcão, por seu incentivo, dedicação, paciência e amizade.

Ao professor Roberto Lotufo pelo incentivo e colaboração.

Aos meus amigos do LCA e do DECOM da FEE/UNICAMP.

Aos antigos e novos amigos com quem convivi em Campinas.

Aos meus saudosos amigos em Fortaleza.

Ao Dr. Jayaram Udupa, *Medical Image Processing Group, Dept of Radiology, Univ of Pennsylvania*, pelo material fornecido e colaboração.

Resumo

Em visualização de volumes, *shell rendering* e *shear-warp rendering* estão entre os métodos mais eficientes. *Shell rendering* utiliza uma representação mais compacta dos dados volumétricos do que *shear-warp rendering*. Porém *shear-warp rendering* permite maior eficiência no *rendering* por explorar a fatoração *shear-warp*, da matriz de visualização. Este trabalho propõe uma extensão da estrutura de dados do *shell rendering* para permitir fatoração *shear-warp* da matriz de visualização. O novo método, denominado *shell rendering* com fatoração *shear-warp* provê o melhor *custo* × *benefício* entre os métodos, considerando três aspectos: qualidade de imagem, velocidade de *rendering* e espaço em memória. Esta dissertação apresenta uma revisão sobre os principais conceitos em visualização de volumes, descreve os métodos *shell rendering*, *shear-warp rendering* e *shell rendering* com fatoração *shear-warp* apontando suas diferenças nos aspectos computacionais, e apresenta uma análise comparativa entre os três métodos. Uma outra contribuição importante é que esta dissertação aponta um erro conceitual do *shell rendering*, o qual afeta a corretude do *rendering* e por conseqüência, a qualidade da imagem gerada.

Abstract

In volume visualization, shell rendering and shear-warp rendering are among the most efficient methods. Shell rendering utilizes a more compact data representation than shear-warp rendering. On the other hand, shear-warp rendering allows a more efficient rendition due to the shear-warp factorization of the viewing matrix. This work proposes an extension of the data structure of shell rendering to allow shear-warp factorization of the viewing matrix. The new method, named shell rendering with shear-warp factorization, provides the best $cost \times benefit$ compromise among the methods, considering three aspects: image quality, speed of rendering, and memory requirements. This dissertation presents a review of the main concepts in volume visualization, describes the methods shell rendering, shear-warp rendering and shell rendering with shear-warp factorization pointing out their differences in the computational aspects, and presents a comparative analysis among them. Another important contribution reported in this dissertation is a conceptual mistake of shell rendering, which affects the correctness of the rendering, and as consequence the final image quality.

"Calvin: I think we have got enough information now, don't you?

Hobbes: All we have is one 'fact' that you made up.

Calvin: That's plenty. By the time we add an introduction, a few illustrations and a conclusion, it'll look like a graduate thesis."

Calvin & Hobbes

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A visualização volumétrica tem sido utilizada em várias áreas tais como microscopia, modelagem molecular, astrofísica, geofísica, química e engenharia. Na medicina, a visualização pode auxiliar no diagnóstico por imagens, no planejamento e acompanhamento de cirurgias, em tratamentos por radiação, no estudo da cinemática de articulações, na caracterização de doenças que afetam a anatomia e a fisiologia de determinadas estruturas, e na educação médica [1]. Nessas aplicações, a principal meta é auxiliar os especialistas na compreensão de fenômenos físicos complexos do corpo humano tendo como base, dados relacionados com parâmetros físicos, tais como, medidas de densidade, velocidade, carga eletrostática e entropia. O enfoque desta dissertação é a visualização de volumes aplicada à medicina.

Pesquisadores ao longo das últimas duas décadas têm se empenhado em desenvolver técnicas de processamento, visualização e análise de dados biomédicos 3D. Graças ao aumento da velocidade de processamento e capacidade de memória dos computadores, e ao surgimento de novas modalidades de aquisição de dados - tomografia por Raios-X (CT e Spiral CT), tomografia por emissão de pósitron (PET), tomografia por emissão de fóton (SPECT), ressonância magnética (MRI), ressonância magnética funcional (fMRI), ultrasom, microscopia confocal -um rápido crescimento no desenvolvimento de *hardware* e *software* para implementação dessas técnicas tem sido possível.

Estes equipamentos adquirem dados na forma de imagens de fatias paralelas e ortogonais a um dos eixos principais do paciente (e.g. cortes transversais, sagitais e coronais). O empilhamento destas imagens, mantendo o espaçamento original entre elas, pode ser idealizado de forma que cada pixel representa o volume de um pequeno paralelepípedo no espaço 3D, denominado *v*oxel. Este volume de dados é denominado *c*ena e representa amostras em uma região 3D do corpo do paciente. Cada voxel tem associado um número inteiro proporcional ao tom de cinza do pixel na imagem correspondente. Cada um desses valores representa a integração de uma propriedade física que está sendo mensurada no

interior da cena. Em uma cena os objetos de interesse para visualização são órgãos ou outras estruturas do corpo associadas ao problema em estudo.

1.1 Visualização em Medicina

Existem três importantes etapas em visualização de volumes na medicina - classificação, representação, e *rendering*. *Classificação* é processo de atribuir aos voxels opacidades proporcionais ao grau de interesse nesses voxels para visualização. O maior desafio na classificação é identificar que voxels pertencem a objetos distintos, que é, em essência, um problema de segmentação [2]. *Representação* consiste em definir um modelo geométrico para o objeto classificado e uma estrutura de dados para armazenar o modelo junto com um conjunto de atributos para visualização. *Rendering* é o processo de simulação da propagação dos raios de luz através da cena e da determinação da luz que alcança o observador para criar diferentes vistas dos objetos selecionados.

Abordagens de visualização podem ser classificadas em dois grupos - *rendering* de superfície e *rendering* de volume. Em *rendering* de superfície, os voxels são classificados como opacos ou transparentes [3, 4, 5, 6]. Como conseqüência, um modelo geométrico da superfície dos objetos pode ser criado. Os métodos podem ser posteriormente divididos em duas classes de modelos geométricos - poligonal e digital. Métodos digitais representam a superfície dos objetos como um conjunto de primitivas - voxel e faces de voxel - que estão diretamente associados com o sistema de coordenadas da cena [5, 6]. Em modelos poligonais, um conjunto de polígonos - mais comumente triângulos - são utilizados para representar a superfície [3, 4]. Em *rendering* de volume, o processo de classificação é nebuloso onde voxels têm um grau contínuo de opacidade que varia de transparente a opaco [6, 7, 8, 9]. O modelo geométrico dos objetos é digital (um conjunto de voxels), e no caso de classificação binária, *rendering* de volume é equivalente ao *rendering* de superfície digital, o que torna esta abordagem interessante, simples e flexível para visualização.

1.2 Motivação

A despeito das dificuldades inerentes ao processo de classificação, visualização em velocidade interativa é fundamental. Isto é sempre um desafio porque a quantidade de informação para visualização normalmente requer bastante espaço de memória e cálculos computacionais intensivos. Pesquisadores têm proposto *hardware* especializado [9, 10] e algoritmos que comprometem qualidade de imagem por velocidade [11, 12] na intenção de resolver o problema de velocidade. Enquanto o desenvolvimento de *hardware* dedicado é importante para avançar o conhecimento em computação, abordar o problema da in-

teratividade através de algoritmos eficientes pode ser mais prático do ponto de vista de disponibilidade e portabilidade das implementações. Ainda mais, considerando que mais de 85% dos voxels são normalmente classificados como transparentes, as abordagens mais interessantes para o problema de velocidade usam estruturas de dados espaciais que codificam a presença e/ou ausência de voxels de alta opacidade para eliminar o cálculo em regiões transparentes da cena como também em regiões de alta opacidade [6, 7, 8, 13, 14]. Entre estas abordagens, as técnicas de maior sucesso têm sido *shell rendering* [6] e *shear-warp rendering* [13]. Considerando a importância destas técnicas para visualização, o enfoque desta dissertação está no estudo e aprimoramento dessas técnicas.

1.3 *Shell Rendering e Shear-Warp Rendering*

Shell rendering usa uma estrutura de dados compacta de voxels, que consiste de uma lista de atributos de visualização para voxels não-transparentes e de um arranjo 2D de ponteiros para esta lista. *Shear-warp rendering* usa uma lista de voxels não-transparentes e de atributos para visualização, e para cada eixo principal do sistema de coordenadas 3D, usa uma lista para armazenar o comprimento de corridas transparentes e não-transparentes de voxel e um arranjo de ponteiros 1D para a lista de comprimento de corrida e para a lista de voxels. *Shell rendering* utiliza *voxel splatting* com remoção de superfície *front-to-back* enquanto *shear-warp rendering* combina as vantagens de *voxel splatting* e *ray casting* através da fatoração *shear-warp* da matriz de visualização [15]. Os dois métodos provêm *rendering* de superfície e *rendering* de volume com imagens de alta qualidade. *Shell rendering* de superfície é de 18 a 31 vezes mais rápido em um PC Pentium de 300MHz que métodos poligonais baseados em *software* e em *hardware*, incluindo aqueles baseados em máquinas muito caras como a Silicon Graphics Reality Engine II, como mostrado em [16]. *Shell rendering* também foi estendido para prover perspectiva digital [17] e para o *rendering* de modelos geométricos poligonais dentro do mesmo ambiente de trabalho [18] sem comprometer a velocidade de *rendering*. Ao mesmo tempo, *shear-warp rendering* tem sido desenvolvido em vários aspectos - implementação paralela, seqüencial, baseada em *software*, e baseada em *hardware* [19, 20, 21, 22, 23, 24, 25, 26].

1.4 Objetivos

A partir da descrição sucinta sobre as características principais dos métodos *shell rendering* e *shear-warp rendering*, pode-se perceber que *shell rendering* requer menos espaço de memória do que *shear-warp rendering*. Porém *shear-warp rendering* parece prover maior eficiência que *shell rendering* por explorar a fatoração *shear-warp* da matriz de visua-

lização. Tendo em vista o estudo e aprimoramento desses métodos, os objetivos deste trabalho foram:

1. Investigar um método híbrido que combinasse as vantagens do *shell rendering* com relação baixo uso de memória e as vantagens da fatoração *shear-warp*.

2. Fazer uma análise comparativa dos três métodos, *shell rendering*, *shear-warp rendering*, e *shell rendering* com fatoração *shear-warp*, com relação aos aspectos: qualidade de imagem, velocidade de *rendering* e uso de memória.

1.5 Contribuições

As principais contribuições deste trabalho são:

- O método *shell rendering* com fatoração *shear-warp* [27], que representa o melhor *custo* × *benefício* entre os três métodos.

- Uma análise comparativa dos métodos que envolveu 10 objetos de diferentes tamanhos e derivados de cenas de diversas modalidades de aquisição de dados [28].

- A descoberta de um erro conceitual no *shell rendering* [29] que afeta a corretude de *rendering*, e por conseqüência, a qualidade da imagem final.

1.6 Organização do Trabalho

Esta dissertação está organizada como a seguir. O capítulo 2 faz uma revisão dos principais conceitos sobre visualização de volumes.

O capítulo 3 apresenta dois métodos de *rendering* relacionados com este trabalho, Shell Rendering e Shear-Warp Rendering. Cada método será brevemente descrito em termos sua representação e projeção.

O método proposto neste trabalho é apresentado no capítulo 4. Uma nova representação computacional é sugerida, assim como um algoritmo de *rendering*, adequado à representação.

No capítulo 5, os métodos são comparados e analisados de acordo com os três aspectos: qualidade de imagem, velocidade de *rendering* e uso de memória.

O capítulo 6 conclui o trabalho através da discussão dos objetivos alcançados e lança sugestões de trabalhos futuros.

Capítulo 2

Visualização de volumes

Este capítulo apresenta os conceitos básicos em visualização de volumes necessários à compreensão do restante da dissertação. Esses conceitos provêm de áreas de processamento de imagens [2] e computação gráfica [30]. Eles são organizados em cinco etapas principais de um sistema de visualização: aquisição, pré-processamento, classificação, representação e *rendering*. A aquisição consiste do processo de captura, digitalização e armazenamento de propriedades físicas na forma de uma seqüencia de imagens digitais, referida como cena. O pré-processamento utiliza operações de processamento de imagens para preparar a cena para classificação. A classificação associa um valor de opacidade aos voxels da cena proporcional ao grau de interesse para visualização. A representação define um modelo para os objetos classificados na cena e uma estrutura de dados para armazenar o modelo e as informações necessárias para visualização. O *rendering* simula a propagação de luz na cena para gerar as diferentes vistas dos objetos selecionados.

2.1 Aquisição de Imagens

A *aquisição* dos dados volumétricos é feita por um sensor para imageamento com capacidade de digitalização de sinais. Os dados volumétricos são adquiridos por tomógrafos em forma de imagens de cortes (ou fatias) paralelas ao longo de um dos eixos do paciente. Cada imagem está associada a uma localização $k \in \mathcal{Z}$ no eixo z com uma espessura e que pode variar de 1mm a 15mm, dependendo da resolução necessária à aplicação. Estas imagens têm tamanhos [1] típicos de 256×256 e 512×512 pixels, onde cada pixel tem a profundidade de 12 bits. Os espaçamentos entre fatias adjacentes são normalmente maiores que os espaçamentos entre pixels adjacentes da mesma fatia. As imagens são agrupadas na ordem crescente de sua posição k em uma única estrutura, chamada de cena, cujo

[1] A nova geração de tomógrafos vai produzir imagens de tamanhos 1024×1024 pixels

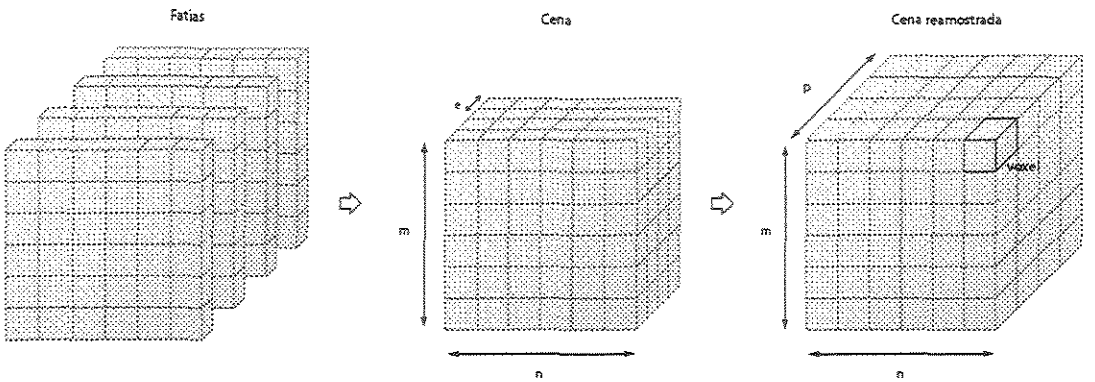


Figura 2.1: Imagens de fatias paralelas formam uma grade retangular finita V de voxels denominada cena.

elemento básico de construção é chamado de voxel (*volume element*). No caso em que a amostragem é variável ao longo do eixo z, o espaçamento entre fatias adjacentes torna-se variável, o que permite que determinadas regiões fiquem mais detalhadas em detrimento do detalhe de outras regiões.

Matematicamente, uma *cena* é um par (V, f) consistindo de uma grade retangular finita V de voxels (pontos em $\mathcal{Z}^3$), e um mapeamento f que designa para cada voxel v em V um valor de intensidade $f(v)$ em Z. Uma cena é comumente armazenada como uma matriz 3D de voxels, onde cada voxel representa um pequeno paralelepípedo em $\Re^3$. Uma cena pode conter um ou mais objetos de interesse para visualização.Um *objeto* em V é um conjunto O de voxels formando um ou mais componentes conexos em $\mathcal{Z}^3$. Cada objeto possui uma *borda* $B \subseteq O$, definida pelo conjunto de voxels formado por todos os voxels $v_b \in O$, onde a vizinhança de v_b intercepta O e o complemento de O.

2.2 Pré-processamento

O pré-processamento pode ser entendido como qualquer operação de processamento de imagens que mapeia um cena C_1 em outra cena C_2 visando aumentar as chances de sucesso na classificação e/ou extrair informação para visualização. Estas operações são: filtragem, segmentação, volume de interesse, transformações radiométricas e reamostragem.

2.2.1 Filtragem

O uso de máscaras espaciais para processamento de imagens é usualmente chamado de *filtragem espacial* (em contrapartida à expressão *filtragem no domínio da freqüência* usando a transformada de Fourier). Nesta seção são considerados filtros lineares e não-lineares para filtragem de imagens [2].

Seja Γ uma filtragem que transforma uma cena C_1 em outra cena C_2. Γ é um filtro linear se:

$$\Gamma[aC_1 + bC_2] = a\Gamma[C_1] + b\Gamma[C_2] \tag{2.1}$$

para cenas C_1 e C_2 e a, b constantes quaisquer.

Os denominados filtros *passa-baixas* atenuam ou eliminam os componentes de alta-freqüência no domínio de Fourier enquanto deixam as freqüências baixas inalteradas. Filtros passa-baixas são normalmente usados para reduzir ruído de quantização e suavizar bordas para visualização através da geração de normais. De modo similar, filtros *passa-altas* atenuam ou eliminam os componentes de baixa-freqüência. O efeito resultante da filtragem passa-altas é um aparente realce das bordas e outros detalhes finos. Este tipo de filtro é útil no cálculo de vetores normais para visualização e na localização das regiões de bordas para classificação. Um terceiro tipo de filtragem, denominado filtragem *passa-banda*, remove regiões selecionadas de freqüências entre altas e baixas freqüências. Esses filtros são usados para restauração de cenas e raramente são interessantes para realce de cenas. Independentemente do tipo filtro linear usado, entretanto, a abordagem clássica consiste em somar os produtos entre os coeficientes da máscara e as intensidades dos voxels sob a máscara numa posição específica da cena.

Filtros espaciais não-lineares baseiam-se diretamente nos valores dos voxels na vizinhança considerada, não utilizando coeficientes como filtros lineares. Exemplos são os filtros mediana para eliminar ruído impulsivo e filtros morfológicos. Filtros morfológicos constituem alternativas à filtragem não-linear e são caracterizados por uma transformação Γ crescente e idempotente [31]. Neste sentido, a erosão e a dilatação, por não serem idempotentes, não constituem filtros morfológicos. Por sua vez, a abertura e o fechamento são exemplos típicos de operadores de filtragem, cuja composição também define filtros. A filtragem morfológica é muito útil também no pós-processamento de imagens, onde filtros de fechamento melhoram a qualidade da imagem gerada por *rendering*, removendo artefatos.

2.2.2 Segmentação

Um dos principais desafios em visualização de volumes é identificar quais voxels pertencem a quais objetos na cena. Esta operação é chamada segmentação. A segmentação subdivide

uma cena em suas partes ou objetos constituintes. O nível até o qual essa subdivisão deve ser realizada depende do problema a ser resolvido [1, 2].

Segmentação tem duas tarefas relacionadas — reconhecimento e delineação. *Reconhecimento* é a tarefa, em alto nível, de determinar a posição aproximada do objeto na cena. *Delineação* é a tarefa, em baixo nível, de determinar a extensão espacial do objeto. Na maioria das tarefas de reconhecimento, operadores humanos treinados superam qualquer algoritmo de computador. Por outro lado, algoritmos de computador delineiam objetos de forma mais precisa, eficiente e exata que o ser humano.

Abordagens para reconhecimento podem ser classificadas em dois grupos: automática e assistida pelo usuário. Em métodos automáticos, uma entre duas abordagens é escolhida: *baseada no conhecimento* [32, 33, 34, 35] ou *baseada em atlas* [36, 37, 38, 39, 40, 41, 42]. A primeira abordagem usa técnicas de inteligência artificial para representar o conhecimento de objetos na região de interesse da cena. Usualmente, a delineação dos objetos é feita primeiro em uma fase de treinamento para depois serem formadas e testadas as hipóteses. Essas operações formam um laço que pode ser executado diversas vezes antes que uma solução aceitável seja encontrada. A segunda abordagem utiliza um atlas já construído que representa as relações geométricas e topológicas dos objetos. Conseqüentemente, algumas entidades geométricas, como pontos, contornos, planos, identificados na cena de entrada (passo de delineação) para uma mesma região do corpo são mapeados para casar com as entidades homólogas no atlas.

Em métodos de reconhecimento automático, a questão levantada é o que fazer em caso de falhas. Métodos completamente automáticos que são a prova de falhas e que têm funcionado corretamente de forma rotineira em um grande número de testes ainda não foram encontrados. A premissa em métodos assistidos pelo usuário é que uma simples ajuda de um operador é suficiente para o reconhecimento.

Abordagens para delineação, por outro lado, foram estudadas mais extensivamente que aquelas para reconhecimento. De fato, a delineação por si só é usualmente considerada um problema de segmentação. Isto é, qualquer que seja a saída do método de delineação, esta é considerada uma representação do objeto de interesse. Duas classes de métodos existem para delineação: *baseada em borda* [43, 44], em que a saída representa a borda do objeto, e *baseada em região* [45, 46], em que a saída representa a região ocupada pelo objeto. Em ambos os grupos, a saída pode ser *binária* ou *nebulosa*. No caso binário, a saída é um conjunto de voxels, onde cada voxel tem um valor de pertinência ao objeto que é 0 ou 1. No caso nebuloso, a saída é um conjunto, onde um grau de pertinência ao objeto, no intervalo $[0, 1]$, é associado a cada voxel.

Considerando as duas estratégias de reconhecimento — automática e assistida pelo usuário — com as quatro abordagens de delineação, oito classes de estratégias de segmentação são possíveis.

2.2.3 Volume de Interesse

Geralmente o objeto de estudo ocupa apenas uma pequena porção do domínio da cena. A operação de *volume de interesse* (VOI) permite criar outra cena cujo domínio é um paralelepípedo que engloba a informação sobre todos os aspectos de interesse do objeto de estudo com o mínimo possível de objetos irrelevantes. O principal propósito da operação de VOI é minimizar o espaço de armazenamento necessário para o processamento futuro da cena. Como exemplo, uma cena $512 \times 512 \times 64$ contém mais de 16,5 milhões de voxels. Se a região ocupada pelo objeto de interesse é $200 \times 200 \times 64$, então o número de voxels — aproximadamente 2,5 milhões, — é cerca de sete vezes menos que a cena inteira. Outra razão para a operação de VOI pode ser a exclusão de um parte do objeto para revelar o seu interior na visualização.

2.2.4 Transformações radiométricas

As transformações radiométricas independem da localização espacial dos voxels na cena e podem, de maneira geral, ser representadas por uma operação do tipo *look-up table* (LUT) que transforma o nível de cinza g_1 em um nível de cinza g_2. Seja G_1 a escala de cinza da cena de entrada C_1: $G_1 = [0, 1, \ldots, N_1]$ e G_2 a escala de cinza da cena resultante C_2: $G_2 = [0, 1, \ldots, N_2]$. Uma LUT é uma operação $r(G_1) \mapsto r(G_2)$ tal que

$$\forall g_1 \in G_2, \exists g_2 \in G_2, g_2 = r(g_1) \tag{2.2}$$

São exemplos de função radiométrica: *stretching*, normalização, limiarização e intensidade de interesse. O objetivo do *stretching* é realçar o contraste entre dois pontos na escala de cinza G_1. A limiarização é utilizada com freqüência no processo de segmentação, onde G_1 é mapeado em intensidades binárias (preto e branco). A operação de intensidade de interesse (IOI) tem o propósito de diminuir o espaço de armazenamento necessário para cada voxel através da redução do domínio de f_1 para f_2, dentro de um intervalo de intensidades cuja representação computacional necessária é menor que em f_1. Por exemplo, valores de intensidade em um intervalo de 256 níveis de cinza podem ser representados por 8 bits ao invés de 12 bits da cena original.

2.2.5 Reamostragem

O objetivo da reamostragem é converter uma cena C_1 em outra cena C_2 com um nível especificado de discretização isotrópica. Isto é, uma cena onde todos os voxels de uma cena têm tamanhos iguais e então cada voxel pode ser identificado por uma tripla (x, y, z) em $\mathcal{Z}^3$ de coordenadas de seu centro. Esta operação é fundamental para evitar distorções na imagem visualizada.

Técnicas de reamostragem podem ser divididas em duas categorias: *baseada na cena* e *baseada no objeto* [1]. Em métodos baseados na cena, os valores de intensidade da cena reamostrada são determinados diretamente dos valores de intensidade da cena de entrada. Em métodos baseados no objeto, alguma informação do objeto é extraída da cena é usada no processo de reamostragem.

Métodos de reamostragem baseados na cena

Supondo que v' é um voxel na cena de saída C_2 e que v_i, $i = 1, 2, \ldots, 8$ são os voxels mais próximos na cena de entrada C_1, conforme ilustra a Figura 2.2. A forma mais simples de reamostragem é contribuir a v', a intensidade do voxel mais próximo entre os voxels v_i, $i = 1, 2, \ldots, 8$. Outra forma, muito comum em visualização de volumes, é a reamostragem linear, que assume que as intensidades dos voxels (g_1) variam linearmente ao longo das direções x, y e z, conforme a Equação 2.3:

$$g_2(v') = g_1(v_n) + [\frac{g_1(v_{n+1}) - g_1(v_n)}{d_{n+1} + d_n}]d_n. \tag{2.3}$$

Para determinar a intensidade em qualquer ponto c' (que representa o centro de um novo voxel v'), são encontrados oito voxels $v_1, v_2, \ldots, v_8$ em C_1 tais que seus centros $c_1, c_2, \ldots, c_8$ estão mais próximos de c'. O cálculo da intensidade em c' consiste de sete cálculos de reamostragem (Equação 2.3): a intensidade em r_1 é calculada através das intensidades de c_1 e c_2 e a intensidade em r_2 é calculada através das intensidades de c_5 e c_6. As intensidades de r_1 e r_2 são utilizadas no cálculo de reamostragem de r_3. A intensidade em r_4 é calculada de forma similar, utilizando três cálculos. Finalmente, $g_2(v')$ é calculado com base nas intensidades de r_3 e r_4.

Refatiamento A transformação de refatiamento pode ser definida como uma operação de reamostragem feita na espaço de coordenadas da cena, para visualizar uma ou mais fatias em qualquer direção arbitrária, que não necessariamente x, y ou z. Esta transformação também permite que medidas quantitativas de distância e área possam ser feitas sobre as novas fatias.

A orientação de uma nova fatia pode ser ortogonal, oblíqua ou curva em relação ao espaço da cena de entrada. Sua localização, resolução e espessura também são parâmetros arbitrários. Portanto, o centro e o volume de um voxel nem sempre coincidem com o centro e o volume de um voxel do espaço original. Neste caso, torna-se necessária a utilização de técnicas de reamostragem para estimar a intensidade dos novos voxels. Os parâmetros de localização e orientação do refatiamento são especificados pelo usuário através de um plano ou uma linha.

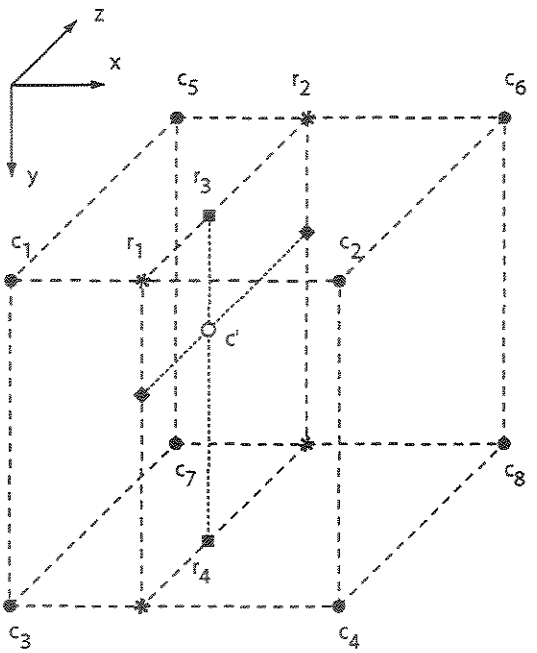


Figura 2.2: Reamostragem trilinear: $c_1, c_2, \ldots, c_8$ são os centros dos oito voxels que estão mais próximos de c', o centro de v'.

Uma grande vantagem do refatiamento é que o paciente não precisa se submeter a um novo exame para que fatias em outras orientações sejam obtidas. No caso das tomografias, evita que o paciente receba uma dose maior de radiação. Em outros casos, a orientação desejada para as fatias é algumas vezes impossível de ser obtida pelo equipamento tomográfico.

Métodos de reamostragem baseados no objeto

O primeiro método baseado no objeto que aparece na literatura é conhecido como *reamostragem baseada na forma*, desenvolvido para cenas de entrada binárias [47]. Estas cenas são resultantes de um processo de segmentação, onde voxels com valor 1 representam o objeto de interesse e voxels com valor 0 representam o fundo. A reamostragem baseada na forma usa uma cena binária como entrada e gera outra cena binária para nível específico de discretização. O método consiste da conversão da cena de entrada binária C_b na cena C_g, ambas com mesmo domínio, pela designação, para cada voxel v em C_g, de um número que representa a menor distância entre v e a borda (mapa de distância Euclidiana) em C_b entre regiões do objeto e regiões na mesma fatia que contém v. O número é positivo se v tem uma intensidade 1 em C_b, caso contrário o número é negativo. A cena C_g é reamostrada usando regras de reamostragem trilinear ou qualquer outra regra de reamostragem

para criar uma cena C'_g. Como os números associados com os voxels em C'_g representam as distâncias da borda, a cena reamostrada C'_b é criada pela designação da intensidade 1 se o voxel correspondente em C'_g tem valor positivo. Todos os outros voxels são designados com intensidade 0. Várias extensões deste método foram desenvolvidas [48, 49, 50].

2.3 Classificação

No modelo conceitual utilizado em *rendering* de volume, uma cena é considerada um bloco colorido e translúcido, com valores de cor e opacidade diferentes para cada região. A simulação da incidência de luz neste bloco permite que, através de sua reflexão, diferentes tipos de tecidos possam ser visualizados pelo observador. O processo de classificação atribui aos tecidos opacidades proporcionais ao grau de interesse nesses tecidos para visualização [51]. Na prática, os voxels de interesse pertencem às superfícies entre os tecidos e a localização desses voxels é um problema de segmentação. Outra forma de resolver o problema é segmentar os objetos e depois atribuir opacidades aos voxels da superfície e/ou vizinhança da superfície desses objetos.

Matematicamente, *classificação* é o processo de transformação de uma cena $C_1 = (V, f)$ em outra cena $C_2 = (V', \alpha)$ pela associação de um valor de opacidade $\alpha(v)$ dentro do intervalo $[0, 1]$ para cada voxel v em V'. Tal classificação é também chamada classificação nebulosa. A classificação binária é um caso particular onde os valores de opacidade $\alpha(v)$ são 0 ou 1. No caso de múltiplos objetos, a classificação pode associar um par ordenado (α, λ) para cada voxel da cena, onde α é o valor de opacidade no intervalo [0,1] e λ é um rótulo no conjunto $\{0, 1, \ldots, n-1\}$ que identifica o objeto. No caso da classificação binária, valores $\alpha \in \{0, 1\}$ e $\lambda \in \{0, 1, \ldots, n-1\}$ são associados.

A função que faz o mapeamento de opacidades para os voxels da cena é associada a uma ou mais características da imagem. Surgem dificuldades no processo de classificação quando esta característica da imagem não é capaz de dissociar dois objetos na cena. Uma classificação baseada apenas na intensidade dos voxels, por exemplo, deve apresentar estruturas em intervalos de intensidade separados. Este tipo de classificação apresenta bom resultado, como mostra a função de classificação de uma cena para pele e osso na figura 2.3. Infelizmente, na maioria das situações isto não ocorre, tornando o problema mais complexo.

Pesquisadores da Pixar, em 1985, foram os primeiros a utilizar técnicas de classificação de forma mais sofisticada em dados médicos 3D. Uma técnica utilizada, descrita em termos gerais por Smith [52] e apresentada em detalhes por Drebin et al. [51], consiste em estimar a fração de cada material no interior dos voxels e usar esta fração para calcular uma cor e uma opacidade parcial para cada voxel. O método de Levoy [53] é semelhante, mas calcula as cores e opacidades diretamente do valor escalar de cada voxel e na magnitude

de gradiente. Quando o valor escalar escolhido é a intensidade dos voxels na cena, a classificação varia de acordo com os valores resultantes da multiplicação ponto a ponto das curvas de classificação de intensidade e magnitude de gradiente da cena, como mostra a Figura 2.3.

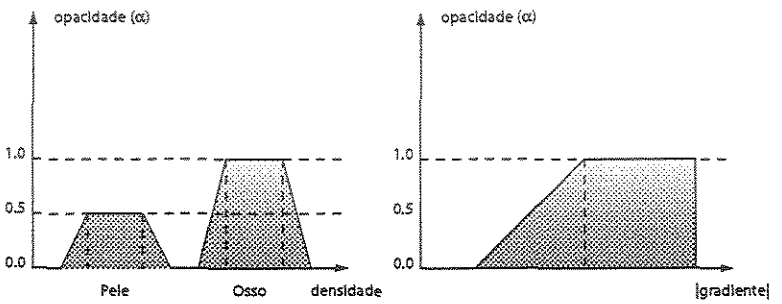


Figura 2.3: Função de classificação para pele e osso baseada na intensidade e na magnitude do gradiente.

2.4 Representação

Os objetos classificados podem ser representados de diversas formas como descrito no Capítulo 1. Assumindo a representação digital, existem várias formas eficientes de armazenar os voxels de uma cena classificada. Um exemplo é a representação *octree*. Esta representação resulta de repetidas subdivisões do cena em subcenas até que todas as subcenas tenham apenas voxels pertencentes ao objeto ou não. A estrutura final é uma árvore, onde cada nó tem oito ramos. Quando a cena tem mais de um objeto de interesse, a representação por *octree* é feita para cada objeto isoladamente. Apesar do acesso aos voxels não ser direto, operações simples de conjuntos, como união, intersecção e diferença, e transformações geométricas podem ser feitas acessando cada nó da árvore no máximo uma vez. Além disto, com exceção das transformações geométricas, estas operações requerem aritmética simples, tais como, adições de inteiros, deslocamentos de bits e comparações [54]. A Figura 2.4a ilustra a idéia da representação *octree* de uma cena. A cena é dividida em oito subcenas iguais, que são identificados por números associados (Figura 2.4b). O processo de subdivisão termina para as subcenas que contêm apenas voxels não pertencentes ao objeto (nós vazios) ou voxels pertencentes ao objeto (nós cheios). No caso de subcenas parciais, cada subcena é subdividida em oito subcenas novamente seguindo a mesma lógica de numeração como mostra a Figura 2.4c. O resultado final é a estrutura de árvore octal, onde os subcenas cheias e vazias representam as folhas e os subespaços parciais representam os nós com oito ramos cada.

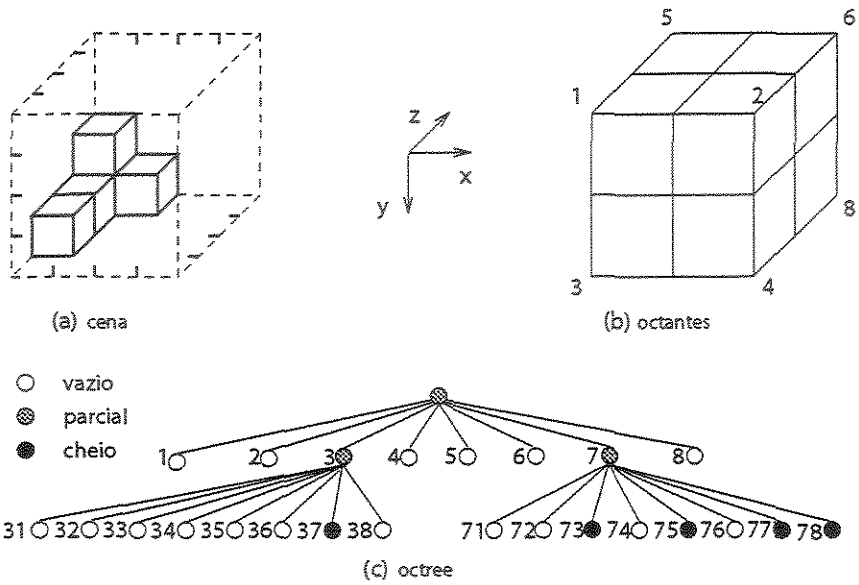


Figura 2.4: A cena em (a) é subdividida em octantes em (b), representados por uma *octree* em (c).

Duas outras representações digitais são a estrutura shell [6] e o codificação da cena em corridas transparentes e não-transparentes [9, 13]. Estas abordagens serão deixadas para o próximo capítulo, onde estas estruturas serão apresentadas no contexto dos métodos *shell rendering* e *shear-warp rendering.*

2.5 *Rendering*

O *rendering* simula a propagação de luz em um ambiente de visualização para formar diferentes vistas dos objetos da cena. No caso de *rendering* de superfície, onde os voxels são opacos, a luz incidente é sempre refletida e nunca transmitida pelo voxel. No caso de *rendering* de volume, onde as opacidades estão no intervalo [0,1], o *rendering* simula o efeito da reflexão e transmissão da luz através dos voxels. Em ambos os casos a imagem resultante codifica no brilho dos pixels a parcela de luz refletida que chega aos olhos do observador. As transformações geométricas são usadas para simular os movimentos da cena com relação ao observador. Estas transformações são rotação e translação. O *rendering* propriamente dito consiste de projeção da cena no plano de visualização para uma dada posição relativa entre observador e cena, e uma tonalização dos voxels projetados.

2.5.1 Ambiente de visualização

O ambiente de visualização contém o observador, a cena e fontes de luz. O modelo de ambiente utilizado neste trabalho é apresentado na Figura 2.5. Neste ambiente o observador e uma fonte de luz branca estão em uma posição $(0, 0, -\infty)$, sobre o semi-eixo $z-$ e o plano de visualização é paralelo ao plano xy e posicionado em $(0, 0, d/2)$, onde d é a diagonal da cena. Assume-se que essa cena pode ser rotacionada em torno do seu centro na frente do observador, que permanece em uma posição fixa para qualquer direção de visualização.

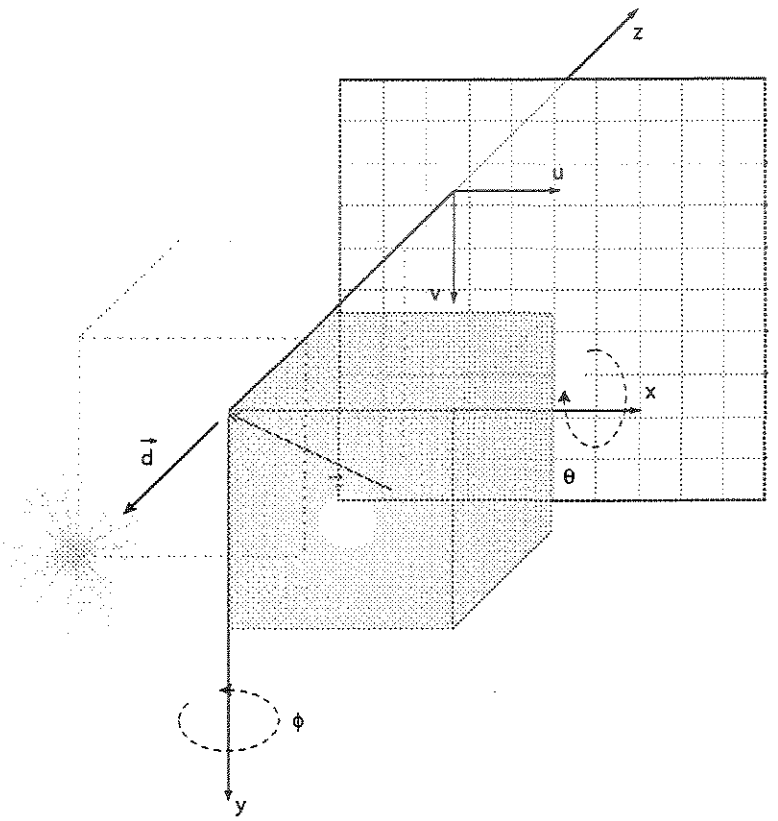


Figura 2.5: O ambiente de visualização utilizado neste trabalho é constituído pelo observador e uma única fonte de luz posicionados em $(0, 0, -\infty)$ e uma cena, centralizada na origem do sistema de coordenadas.

Translação

A rotação da cena nesse ambiente de visualização requer uma translação para que seu centro coincida com a origem do sistema de coordenadas. Esta operação de translação consiste de um deslocamento de todos voxels da cena em uma direção fixa $\vec{t}$. Para tanto,

a cada coordenada x, y, z de voxel são somados os respectivos valores t_x, t_y e t_z através da matriz M_T. Quando a translação $\vec{t} = (-n/2, -m/2, -p/2)$ é aplicada na cena da Figura 2.1, esta fica centralizada na origem do sistema de coordenadas, como mostra a Figura 2.5. Após a formação da imagem no plano de visualização, uma translação de $\vec{t} = (d/2, d/2, 0)$ na imagem torna as coordenadas dos pixels positivas.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_T \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.4}$$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Rotação

Esta transformação é aplicada a cada coordenada (x, y, z) de cada voxel, definindo novo posicionamento (x'_i, y'_i, z'_i). Inicialmente os voxels têm o posicionamento absoluto definido em relação ao sistema de coordenadas, conforme o modelo apresentado na Figura 2.5. Neste modelo, a cena pode ser transladada e rotacionada. A translação centraliza a cena na origem do sistema de coordenadas e a rotação define dois ângulos θ e ϕ em torno dos eixos x e y, respectivamente. Desta forma, a rotação é sempre aplicada em torno do centro da cena e permite a visualização de qualquer face da cena através da variação dos ângulos θ e ϕ. A seguinte equação representa a transformação geométrica da rotação dos voxels da cena:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_R \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.6}$$

A matriz M_R é a composição das conhecidas matrizes M_θ e M_ϕ de rotação em torno dos eixos x e y, respectivamente (Equações 2.7 e 2.8). A ordem das matrizes na Equação 2.9 indica que a rotação é feita inicialmente em torno do eixo y e em seguida em torno do eixo x.

$$M_\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & sen(\theta) & 0 \\ 0 & -sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$M_\phi = \begin{bmatrix} cos(\phi) & 0 & -sen(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ sen(\phi) & 0 & cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$M_R = M_\theta \cdot M_\phi \tag{2.9}$$

Portanto,

$$M_R = \begin{bmatrix} cos(\phi) & 0 & -sen(\phi) & 0 \\ sen(\theta)sen(\phi) & cos(\theta) & sen(\theta)cos(\phi) & 0 \\ cos(\theta)sen(\phi) & -sen(\theta) & cos(\theta)cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

2.5.2 Projeção

A projeção dos voxels no plano de visualização é uma transformação geométrica aplicada em $\mathcal{R}^3$ gerando coordenadas em $\mathcal{R}^2$. Os modelos de projeção podem ser divididos em projeção perspectiva e ortogonal. A projeção em perspectiva é usada em aplicações de endoscopia virtual, onde o observador passeia no interior da estrutura em estudo. A maioria das aplicações usa a projeção ortogonal por ser mais simples, mais eficiente e atender os objetivos da visualização. O modelo de projeção utilizado neste trabalho é o modelo de projeção ortogonal.

Em projeção ortogonal as linhas formadas pelo percurso do raio de luz são paralelas entre si e ortogonais ao plano de visualização. Isto é simulado com o observador posicionado no infinito conforme o ambiente de visualização da Figura 2.5. Como resultado, objetos mais afastados do observador aparecem na imagem final com mesmo tamanho de objetos mais próximos. As coordenadas (x', y') em R^2 (Equação 2.11) são obtidas nas duas primeiras linhas da matriz de visualização M_{view} (Equação 2.12):

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_{view} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.11}$$

Onde:

$$M_{view} = M_R \cdot M_T = \begin{bmatrix} cos(\phi) & 0 & -sen(\phi) & -t_x \\ sen(\theta)sen(\phi) & cos(\theta) & sen(\theta)cos(\phi) & -t_y \\ cos(\theta)sen(\phi) & -sen(\theta) & cos(\theta)cos(\phi) & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.12)$$

Os tipos de projeção ortogonal podem ser divididos em duas classes principais: *ray casting* e *voxel splatting* [13]. As duas características que distinguem cada classe são: a ordem em que cada algoritmo atravessa a cena e o método que o algoritmo usa para projetar voxels na imagem. Cada classe de algoritmo tem vantagens e desvantagens em desempenho, e além disto, uma classe têm técnicas de aceleração mais simples que outra.

A taxonomia utilizada é baseada em laços de varredura. Grande parte dos algoritmos de *rendering* consistem de seis laços encadeados: três que fazem a iteração na cena e três que fazem a iteração no filtro de reamostragem.

Ray casting

Em *ray casting*, o ambiente de visualização é definido com o plano de visualização entre a cena e o observador, como mostra a Figura 2.6. Algoritmos *ray casting* produzem uma imagem através do lançamento de um raio através da cena para cada pixel e integrando o brilho e opacidade ao longo do raio [53, 55, 56]. Algoritmos *ray casting* são chamados *image order* pois a iteração dos laços mais externos é feita na imagem. Os laços mais internos utilizam uma função que extrai as coordenadas dos voxels e sua contribuição parcial na reamostragem linear nos eixos x,y e z.

Para $y_i \leftarrow 1$ **até** Imagem.altura
  Para $x_i \leftarrow 1$ **até** Imagem.comprimento
    Para $z_i \leftarrow 1$ **até** Imagem.profundidade
      Para cada x na Reamostragem(x_i, y_i, z_i)
        Para cada y na Reamostragem(x_i, y_i, z_i)
          Para cada z na Reamostragem(x_i, y_i, z_i)
            Adicione a contribuição do Voxel(x, y, z) no Pixel(x_i, y_i)

Os dois laços mais externos fazem a iteração nos pixels da imagem. O próximo laço faz a amostragem de pontos ao longo do raio no espaço imagem. Finalmente, os três laços internos fazem a iteração sobre os voxels necessários na reamostragem para reconstruir uma amostra no espaço imagem. O corpo do laço multiplica o valor de cada voxel por um peso de reamostragem e adiciona o resultado no pixel correspondente na imagem.

Algoritmos *ray casting* são também chamados de algoritmos de projeção inversa, pois eles calculam a mapeamento de voxels para pixels da imagem, partindo dos pixels.

A maior desvantagem deste tipo de algoritmo é que ele não acessa a cena na ordem de armazenamento, uma vez que os raios de visualização atravessam a cena em uma direção arbitrária. Como resultado, este algoritmo desperdiça mais tempo calculando a posição dos pontos de amostragem e fazendo o endereçamento aritmético do que a outra classe de algoritmos.

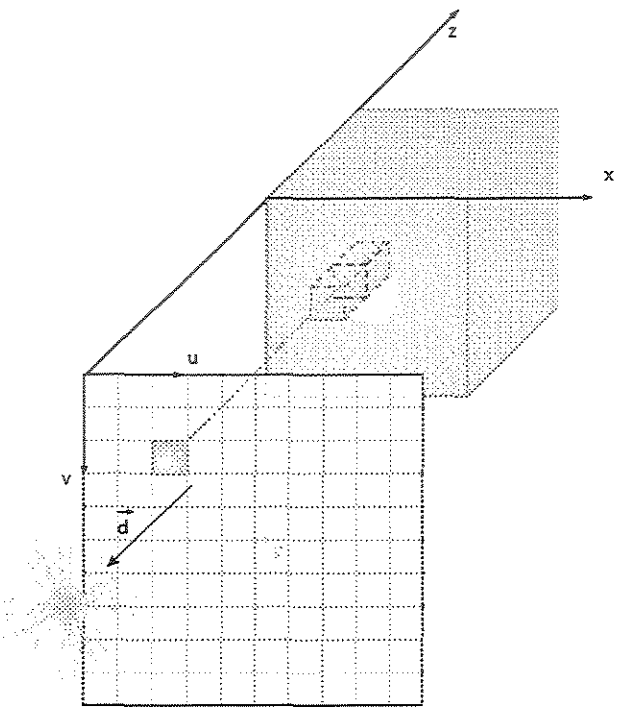


Figura 2.6: Em *ray casting*, o plano fica entre a cena e o observador. Neste algoritmo, uma imagem é formada pelo lançamento de um raio através cena para cada pixel.

Remoção de superfícies escondidas Um aspecto importante na projeção é a técnica de remoção de superfícies escondidas. Esta técnica visa tonalizar apenas os voxels visíveis pelo observador, fazendo composição da tonalização dos voxels que pertencem a um mesmo raio e ao mesmo tempo calculando a opacidade acumulada no raio, conforme será descrito na seção 2.5.3.

Em *ray casting*, as superfícies escondidas são removidas durante o cálculo da opacidade acumulada do raio que atravessa a cena. Desta forma, o algortimo utiliza força bruta, ou seja, não possui nenhuma otimização do caminho pecorrido pelo raio. O objetivo dessa otimização é reduzir o cálculo em regiões ocultas da cena. Entre as técnicas utilizadas destacam-se *heirarchical enumeration* e *early ray termination*.

Heirarchical enumeration utiliza uma estrutura de dados *octree* que subdivide sucessivamente cena, formando uma árvore com diversos níveis. Um nível é percorrido apenas se este possui voxels não-transparentes. Caso contrário, cada nível da árvore que possui voxels transparentes pode ser omitido do processo de busca do raio. Em *early ray termination*, a otimização do *rendering* é terminar os cálculos de tonalização de um pixel quando a opacidade acumulada alcança um valor limiar que corresponde a uma parcela ínfima de luz refletida para observador. Levoy reporta que um algoritmo de *ray casting* em cena de dados médicos, com limiar de 95%, consegue acelerações de 1,6 a 2,2 vezes. Quando combinado com uma *octree*, esta aceleração é de 5 a 11 vezes [7].

Voxel Splatting

Ao contrário dos algoritmos de *ray casting*, algoritmos de *voxel splatting* operam através da iteração nos voxels da cena [57]. Esta classe de algoritmos calcula a contribuição de cada voxel para a imagem através de sua projeção em uma vizinhança de pixels que distribui o valor de tonalização do voxel de acordo com o filtro utilizado. Esta vizinhança envolve mais de um pixel quando o voxel ou a máscara do filtro possui tamanho maior que um pixel. Algoritmos deste tipo são chamados de *object order*, pois o laço mais externo faz a iteração com os voxels do objeto que está sendo construído na imagem.

Para $z \leftarrow 1$ **até** Cena.profundidade
  Para $y \leftarrow 1$ **até** Cena.altura
    Para $x \leftarrow 1$ **até** Cena.comprimento
      Para cada y_i na Projeção(x, y, z)
        Para cada x_i na Projeção(x, y, z)
          Adicione contribuição do Voxel(x, y, z) no Pixel(x_i, y_i)

Os três laços externos fazem a iteração sobre os voxels. Finalmente, os laços mais internos fazem a iteração sobre os pixels afetados por um voxel.

Este tipo de algoritmo é também chamado de algoritmo de projeção direta, pois os voxels são projetados diretamente na imagem, na mesma direção dos raios de visualização. Uma vantagem sobre o algoritmo *ray casting* é que a varredura em *object order* é feita na ordem de armazenamento. A vizinhança deve ser escolhida de modo a evitar "buracos" (Figura 5.3) ou sobreposição excessiva entre voxels adjacentes após a projeção na imagem.

Remoção de superfícies escondidas Durante a projeção da cena é necessário evitar a projeção de voxels que não são visíveis pelo observador. A remoção de superfícies

escondidas consiste em identificar sobre cada raio de projeção, o ponto a partir do qual as contribuições dos voxels que pertencem ao objeto, projetados sobre um pixel p_i, deixam de influenciar na composição do brilho desse pixel. Isto ocorre quando a opacidade acumulada do pixel p_i está saturada, isto é, quando a luz refletida por um voxel v_i não consegue alcançar o pixel sobre o qual é projetada porque a soma das opacidades dos voxels entre v_i e o observador é maior ou igual a 1 (assumindo o ambiente da Figura 2.5). Dentre as técnicas de remoção de superfícies escondidas usadas com *voxel splatting* de voxels [58], as mais comuns são: *depth-sort*, *z-buffer*, *back-to-front* e *front-to-back*:

- **z-buffer**: algoritmo bastante difundido em computação gráfica. Para cada pixel do plano de visualização são associados dois valores: um valor de tonalização e o valor da distância (medida sobre o raio de visualização) entre o voxel que está sendo projetado e o pixel. Um novo valor de tonalização e um novo valor de distância são associados ao pixel apenas se o novo valor de distância for maior que o anterior. Isto garante que, entre os voxels não-nulos interceptados pelo raio de visualização, o voxel visível é o que está mais próximo do observador. Fica claro também que este método, da forma como está descrito acima, pode ser utilizado apenas em *rendering* de superfícies, uma vez que a tonalização do pixel não é calculada pela combinação da contribuição de vários voxels, mas é considerada apenas a contribuição do voxel mais próximo ao observador.

- **depth-sort**: trata-se de um outro algoritmo bastante comum em computação gráfica. Ele classifica todos os voxels da cena de acordo com a distância ao plano de visualização, e depois projeta-os na ordem em que a distância decresce; do mais afastado para o mais próximo do plano. Esse processo de classificação torna-o mais lento e ligeiramente mais complexo que o algoritmo *z-buffer*. Porém, este método permite que o brilho de um pixel seja calculado pela combinação das contribuições de vários voxels. Logo, é adequado também para *rendering* de volume.

- **back-to-front**: o algoritmo *back-to-front* (BTF) [53, 54, 59] explora a informação de proximidade com o observador, implícita na cena (coluna-por-coluna, linha-por-linha, fatia-por-fatia), evitando a utilização de qualquer tipo de classificação de distâncias. Este algoritmo acessa os voxels coluna-por-coluna, linha-por-linha, fatia-por-fatia, na ordem em que eles se aproximam do observador, fazendo com que os mais próximos sejam projetados, no plano de visualização, sobre os mais afastados. Uma desvantagem deste algorítmo é que ele somente remove superfícies de cenas totalmente opacas, ou seja, não pode ser utilizado para *rendering* de volume.

- **front-to-back**: o algoritmo *front-to-back* (FTB) baseia-se na mesma idéia do algoritmo anterior, mas acessa os voxels na ordem em que eles se afastam do observador.

A vantagem do acesso FTB [58, 59] em relação ao acesso BTF é que ele pode acelerar o processo através de técnicas de otimização, como evitar o cálculo de tonalização para pixels da imagem final que já tem suas opacidades saturadas. Os métodos descritos nos próximos capítulos usam a técnica FTB. Portanto, pata tornar mais clara sua definição apresenta-se um exemplo 2D na Figura 2.7.

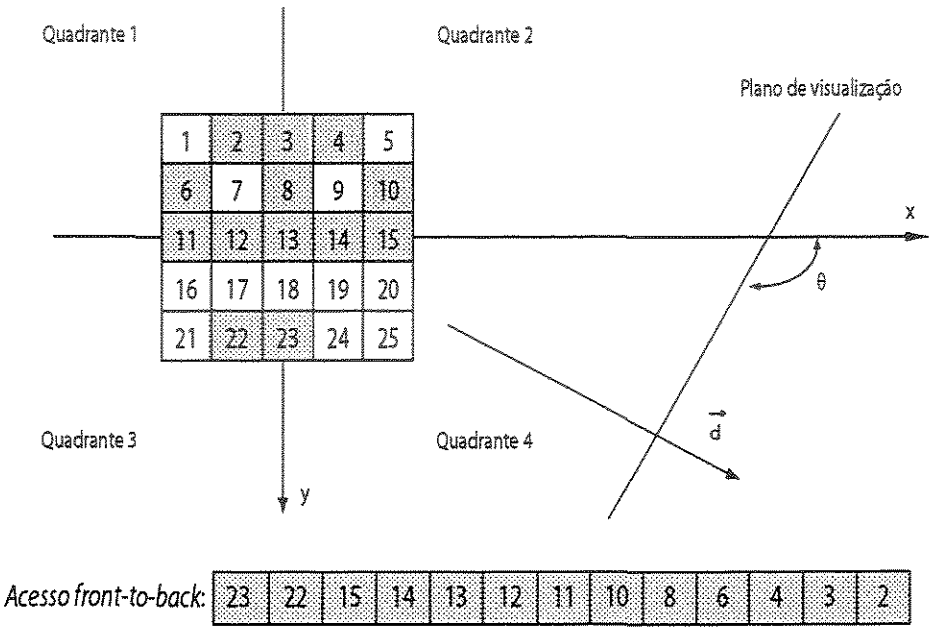


Figura 2.7: Um exemplo de uma cena 2D 5×5 com voxels numerados de 1 a 25. Os voxels opacos estão sombreados. A varredura é feita na ordem dos eixos: x e y, do voxel mais próximo do observador para o mais afastado.

A implementação eficiente de *early ray termination* é trivial para um algoritmo de *ray casting*, mas o mesmo não é verdadeiro em *voxel splatting*. Uma aproximação de *early ray termination* é checar a opacidade de cada pixel na imagem antes de fazer a sua composição com o voxel. No entanto, o algoritmo ainda teria que visitar cada voxel da cena. Esta otimização reduz o número de operações de tonalização, reamostragem e composição, mas não reduz o tempo gasto com regiões ocultas da cena com transformações geométricas e projeção. Estas operações em sua maioria envolvem multiplicações ponto-flutuante que têm grande efeito no tempo de processamento do *rendering*, sendo necessário evitá-las quando possível.

Estruturas de dados baseadas em *octree* e *quadtree* na imagem podem reduzir significativamente a complexidade do algoritmo da cena [60]. Ao invés de examinar cada voxel, o algoritmo pode evitar um nó oculto na *octree* da cena checando um pequeno número de nós na *quadtree* da imagem que já tem opacidade saturada.

Reynolds et al. [61] propõem uma técnica mais eficiente, chamada de *dynamic screen*, para implementação de *early ray termination* em um algoritmo de *rendering* de volume com projeção *object-order*. Ao invés de usar uma *octree* e uma *quadtree*, esta técnica utiliza um código de corrida para codificar a coerência espacial da cena e da imagem. Os autores fazem a seguinte observação: se uma transformação é uma projeção paralela então ela transforma linhas paralelas no espaço da cena em linhas paralelas no espaço da imagem. Esta observação leva a um algoritmo de *rendering* eficiente: para cada linha de varredura de voxels (em projeção *front-to-back*) o algoritmo atravessa simultaneamente a linha de varredura na cena e a linha projetada na imagem.

2.5.3 Tonalização e Cor

A tonalização tem por objetivo fornecer a impressão 3D associando a cada pixel da imagem um brilho e, alternativamente, uma cor utilizada para identificar objetos distintos na imagem ou criar uma impressão de realismo [62, 58].

A tonalização, associada a um pixel p_i da imagem, é resultado da combinação da luz e da opacidade provenientes de todos os voxels $v_1, v_2, ..., v_n$, que são projetados em p_i. A combinação dos voxels deve levar em conta a opacidade acumulada a cada passo do processo, pois se essa opacidade acumulada atinge um valor muito próximo do valor de opacidade máxima, pode-se considerar que a luz refletida por qualquer outro voxel mais afastado não chegará ao pixel, pois irá perdendo intensidade, conforme atravessa os voxels mais próximos do observador, até que nada reste para contribuir na tonalização do pixel. Nessa situação, o pixel p_i pode ser considerado saturado.

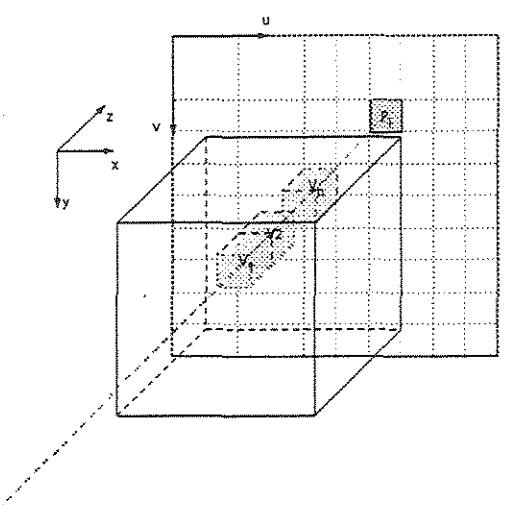


Figura 2.8: A tonalização é resultado da combinação da luz e da opacidade provenientes de todos os voxels $v_1, v_2, ..., v_n$, que são projetados em p_i.

A equação abaixo mostra uma forma como a tonalização associada ao pixel p_i pode ser estimada:

$$S_{p_i} = \sum_{i=1}^{n} I_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{2.13}$$

O componente $I_i \alpha_i$, da equação 2.13, representa o total de luz que atravessa um voxel v_i, onde I_i é o valor da luz incidente neste voxel, calculado através de algum modelo de iluminação conhecido, como por exemplo o modelo de Gouraud [63] ou Phong [64]; e α_i é o valor da opacidade de v_i (esse valor está no intervalo [0, 1], de forma que se $\alpha_i = 1$, toda a luz incidente é refletida).

A luz refletida por um voxel v_i, antes de atingir o pixel sobre o qual está sendo projetada, deve atravessar o meio semi-transparente de todos os outros voxels que estão na mesma direção de projeção. Esse obstáculo faz com que a luz total, refletida por v_i, perca intensidade conforme viaja através desses voxels. A equação 2.13 expressa esse efeito através do componente $\prod_{j=1}^{i-1}(1 - \alpha_j)$, que representa a parcela de luz refletida por v_i que atravessa os voxels v_j, $j = 1, 2, \ldots, i-1$, que estão entre v_i e o observador.

O modelo de iluminação mais difundido na área de visualização de imagens médicas [62] é o modelo de iluminação de Phong. Em Phong, algumas propriedades do comportamento da luz são levadas em conta, para definir como a luz ambiente e a fonte de luz influenciam no cálculo do valor de tonalização:

- **Influência da distância da fonte de luz:** A intensidade de luz, refletida por cada voxel em direção ao observador, cai com o aumento da distância entre esse voxel e o observador. Este efeito é representado pela variação do valor de tonalização, em função da distância normalizada entre o voxel e o plano de visualização (valor do *z-buffer*). Quanto mais afastado do plano de visualização, mais claro será esse ponto, e quanto mais próximo do plano, mais escura será sua tonalização. Esse efeito é modelado de forma linear pela equação 2.14.

$$I_{dist}(u, v) = \frac{I_{max} - I_{min}}{d_{max} - d_{min}} [d - d_{min}] + I_{min} \tag{2.14}$$

onde:
I_{max} é a intensidade máxima desejada na imagem;
I_{min} é a intensidade mínima desejada na imagem;
d_{max} é a maior distância possível entre um voxel e o plano de visualização (maior valor que z' pode assumir);
d_{min} é a menor distância possível entre um voxel e o plano de visualização (menor valor que z' pode assumir);

d é a distância do voxel v até o observador.

- **Influência da Fonte de Luz:** a luz proveniente de uma fonte pontual, incidindo sobre um voxel, resulta em duas formas de reflexão que são consideradas no modelo de iluminação de Phong:

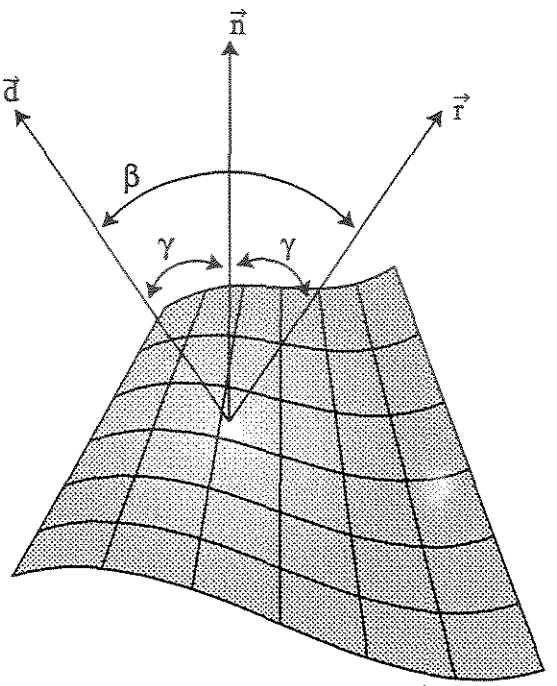


Figura 2.9: Modelo de Phong de iluminação.

  - **Reflexão Difusa:** Parcela da intensidade de luz refletida em um voxel é a mesma em todas as direções, portanto atingindo o observador independente de sua posição, mas varia com o cosseno do ângulo γ entre a direção do raio de luz incidente e a direção do vetor normal à superfície, estimado neste voxel.

  - **Reflexão Especular:** Parcela da intensidade de luz, refletida em um voxel, que é máxima na direção $\vec{r}$ de reflexão, que forma um ângulo γ com o vetor normal à superfície, no lado oposto ao da luz incidente, e vai diminuindo com o afastamento em relação à direção de visualização $\vec{d}$, de acordo com alguma regra (Figura 2.9). Esta intensidade varia com o cosseno do ângulo β, entre a direção $\vec{d}$ e a direção do observador, elevado a um expoente n, relativo ao tipo de tecido. Como as direções do observador e da fonte de luz são iguais, $\beta = 2\gamma$.

- **Influência à luz ambiente:** Parcela da intensidade de luz, refletida pelos voxels, que é constante devido à luminosidade natural do ambiente, e é modelada por um fator constante K_a.

Uma vez definidas tais propriedades, pode-se representar o modelo de iluminação de Phong através da seguinte equação:

$$I(u,v) = I_{max}K_a + I_{dist}(u,v)(K_d cos(\gamma) + K_s cos^n(\beta)) \tag{2.15}$$

onde:
I_{max} é a intensidade máxima desejada na imagem;
$I(u,v)$ é o nível de cinza associado ao pixel (u, v);
K_a é o coeficiente de reflexão para a luz ambiente;
K_d é o coeficiente de reflexão difusa;
K_s é o coeficiente de reflexão especular;
n é um coeficiente do material;
$I_{dist}(u,v)$ é o valor da tonalização baseada em distância, calculado através da (equação 2.14);
γ é o ângulo entre o vetor normal à superfície e o raio de incidência. Devem ser considerados os valores de γ entre 0 e 90 graus para o componente difusa, e valores entre 0 e 45 graus para o componente especular.
β é o ângulo entre o raio de incidência e a direção de reflexão D, e como a posição do observador e da fonte de luz são iguais, o valor de β é igual a 2γ.

Os valores dos parâmetros da equação 2.15 são escolhidos pelo usuário de forma empírica, até a obtenção de um resultado satisfatório. Os coeficientes de reflexão (K_d, K_s, K_a) devem apresentar valores no intervalo [0, 1], de forma que a soma deles seja igual a 1. O expoente n, relativo ao tipo de tecido, deve ter um valor no mínimo igual a 0. Os cossenos dos ângulos β e γ são obtidos pelo produto interno entre os vetores normalizados nas respectivas direções.

Estimação das normais à superfície dos tecidos

As técnicas de visualização, em sua maioria, requerem também que um vetor normal seja associado a cada voxel. O vetor normal é utilizado para o cálculo de tonalização durante a projeção.

A direção da normal em cada voxel pode ser estimada de diversas formas: do gradiente na cena original, o *z-buffer*, gradiente da cena de opacidades, ou o gradiente da cena binária filtrada do mapa de distância Euclidiana. Devido à sua influência na qualidade da imagem, deve-se escolher com muito cuidado o método utilizado para estimar as normais.

A Figura 2.10 ilustra uma vizinhança do tipo 6 de voxels, para o cálculo dos vetores normais à superfície do objeto a partir do gradiente da cena. Esse cálculo pode ser feito da seguinte forma:

$$g_x = \frac{f(x+1,y,z) - f(x-1,y,z)}{2}$$

$$g_y = \frac{f(x,y+1,z) - f(x,y-1,z)}{2}$$

$$g_z = \frac{f(x,y,z+1) - f(x,y,z-1)}{2}$$

$$\vec{n} = \frac{\vec{g}}{||\vec{g}||} \tag{2.16}$$

onde:

$\vec{g}$ é o vetor gradiente da cena de entrada.
g_x, g_y, g_z são os componentes do vetor gradiente nos eixos x, y e z.
$f(x,y,z)$ é o nível de cinza na cena de entrada;
$\vec{n}$ é o vetor normal estimado na superfície do objeto, normalizado.

A normal proveniente do cálculo do gradiente da cena depende da constituição da estrutura que se deseja visualizar. Por exemplo, o gradiente na cena original é recomendado para tecidos ósseos em CT, mas não para estes mesmos tecidos em MRI. Como depende muito da modalidade de aquisição, esta solução não é indicada para todos os casos.

Em cenas binárias, a normal estimada não reflete com fidelidade a normal da superfície do objeto pois a cena apresenta variações abruptas de intensidade em uma vizinhança local. A Figura 2.10 mostra um exemplo de uma vizinhança-6, onde a normal é estimada através do gradiente (Equação 2.16) da intensidade dos voxels. A suavização das bordas do objeto através de uma filtragem Gaussiana da cena binária diminui a diferença entre a normal calculada e normal da superfície do objeto, mas não é uma solução eficaz para todos os casos.

O cálculo da transformada de distância Euclidiana 3D na cena binária que contém o objeto de interesse é a solução que apresenta melhor resultados entre as alternativas anteriores.

Um aspecto importante é o armazenamento do vetor normal na lista de atributos para visualização do voxel. Armazenar os valores das normais em ponto-flutuante é bastante dispendioso em memória. Uma técnica de quantização é normalmente utilizada para diminuir o custo de memória. Desta forma, para cada voxel é associado o índice de

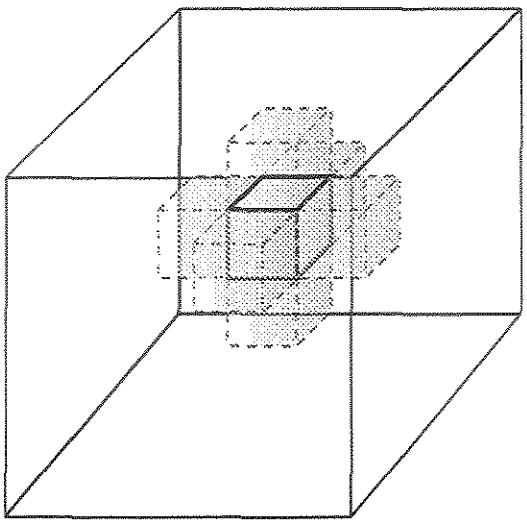


Figura 2.10: Vizinhança-6 de um voxel na cena.

vetor em uma tabela de normais. Esta tabela é construída através da amostragem de um grande número de normais na superfície de um sólido como, por exemplo, uma esfera ou um cubo. O voxel guarda o índice do elemento desta tabela que contém a normal que mais se aproxima à normal estimada em sua superfície. A quantização pode afetar a qualidade da imagem final quando o número de normais amostradas não é suficiente para gerar tons de cinza diferentes necessários para representar de forma satisfatória a variação de tonalidade na superfície dos tecidos.

Capítulo 3

Métodos relacionados

O capítulo anterior reviu os conceitos comuns aos métodos *shell rendering* e *shear-warp rendering*. Neste capítulo serão abordados os aspectos que diferenciam estes métodos. Estes aspectos dizem respeito à representação e *rendering*.

Shear-warp rendering pertence a uma família de algoritmos de *rendering*, proposta na tese de doutorado de Philippe Lacroute [13], que combina vantagens de métodos *voxel splatting* e *ray casting* através da fatoração *shear-warp* da matriz de visualização. Além da fatoração são utilizadas em conjunto várias estratégias para otimizar o *rendering*, como a coerência espacial de estruturas da cena, *early ray termination* e a relação entre pixels do plano de visualização e voxels do volume.

Shell rendering usa uma estrutura compacta chamada *shell*, que consiste de uma lista de atributos de visualização para voxels da cena e um arranjo 2D de ponteiros para esta lista. Este método também utiliza *voxel splatting* com acesso *front-to-back* aos voxels.

3.1 *Shear-Warp Rendering*

A principal característica do *shear-warp rendering* é a fatoração *shear-warp* [15] da matriz de visualização em um sistema de coordenadas padrão onde o terceiro eixo é o mais paralelo à direção de visualização — denominado eixo principal. A fatoração *shear-warp*, consiste na separação da matriz de visualização em duas matrizes: *shear* e *warp*. A transformação *shear* corresponde ao deslocamento 3D que distorce os objetos na cena, como mostra a Figura 3.1. A transformação *warp* corresponde a uma correção da distorção desses objetos que resulta na imagem final. Em *shear-warp rendering* a projeção ortogonal é realizada após a transformação *shear* gerando uma imagem intermediária e a transformação *warp* (2D) é aplicada na imagem intermediária para gerar a imagem final. O detalhamento matemático da fatoração é apresentado no Apêndice A.

Os aspectos relacionados às transformações que cada uma destas matrizes aplicam

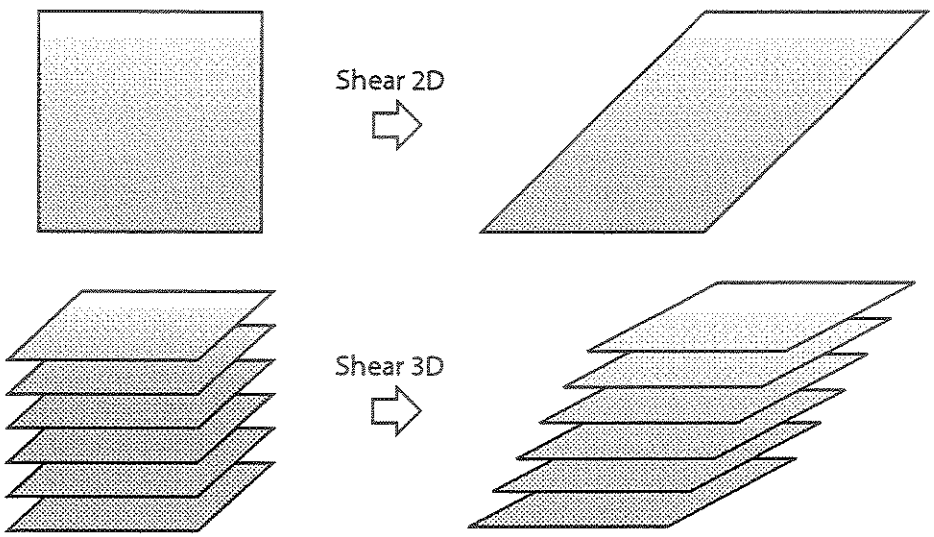


Figura 3.1: Exemplos da transformação *shear* em 2D e 3D.

na cena tornam possíveis algumas otimizações que facilitam o processamento paralelo durante o *rendering*. A fatoração *shear-warp* também já foi explorada em trabalhos anteriores com os seguintes objetivos: reduzir o custo da projeção de uma cena em algoritmos *object-order*, reduzir o custo de traçar raios em algoritmos de *ray-casting* e regularizar os padrões de comunicações de algoritmos paralelos utilizados em processadores de arquitetura SIMD [19].

3.1.1 Representação

O algoritmo de *shear-warp rendering* utiliza estruturas de dados para codificar o volume e imagem intermediária. Uma propriedade da fatoração *shear-warp* afirma que linhas de varredura na cena deslocada são paralelas às linhas de varredura na imagem intermediária. Esta propriedade permite que as estruturas de dados da cena e da imagem possam ser percorridas simultaneamente na ordem de varredura, para implementação de *early ray termination*. Duas estruturas de dados baseadas em coerência de linhas, como representações de código de corrida, são utilizadas para na codificação da cena e da imagem intermediária.

A primeira estrutura de dados utilizada é um código de corrida pré-calculado nas linhas da cena. Esta estrutura permite que voxels transparentes sejam omitidos durante o *rendering*. A codificação da corrida em uma linha da cena consiste de uma série de corridas, cada uma representada por um comprimento de corrida e os atributos dos voxels de cada corrida. O algoritmo de *shear-warp* rendering utiliza dois tipos de corrida para a cena: uma corrida de voxels transparentes e outra corrida de voxels não-transparentes.

As estruturas dados necessárias para estas corridas são construídas durante o processo de classificação da cena. Durante o *rendering*, os voxels são transladados e compostos na imagem intermediária sem a representação explícita da cena decodificada.

Um código de corrida também é utilizado na imagem intermediária, aproveitando a coerência entre a imagem e a cena. Neste caso existem dois tipode de corrida: pixels saturados e não saturados (opacidade acumulada menor que 1). A saturação dos pixels ocorre durante a projeção *shear*, então a estrutura de dados para a imagem intermediária deve ser dinâmica, ao contrário da estrutura utilizada na cena. Este problema é equivalente ao de união de conjuntos disjuntos [65, 66]. Uma estrutura de dados comum para representar estes conjuntos é uma floresta de árvores. As árvores são representadas pelas corridas na imagem intermediária. O algoritmo constrói sua estrutura de dados enquanto calcula o *rendering* da imagem intermediária. Este código de corrida consiste em um valor de corrida na imagem guardado em cada pixel da imagem intermediária (Figura 3.2). O valor de corrida em cada pixel aponta para o próximo pixel não saturado da mesma linha. Através disso é possível omitir o cálculo de projeção e composição de voxels correspondentes a pixels saturados na imagem. Uma otimização de compressão de caminho pode ser utilizada, modificando o ponteiro de cada pixel em uma corrida para o final desta.

A codificação da cena em corridas pelo método *shear-warp rendering* apresenta algumas desvantagens. Em relação manipulação, um voxel com coordenadas arbitrárias (x_0, y_0, z_0) só pode ser acessado após uma série de corridas em cada linha desde o primeiro voxel do volume. Em relação ao uso de memória, são necessárias três codificações, uma em cada direção de varredura dos eixos x, y e z, como mostra a Figura 3.3. Isto fica claro quando a ordem de varredura da Tabela 3.1 é respeitada de acordo com eixo principal associado à direção de visualização.

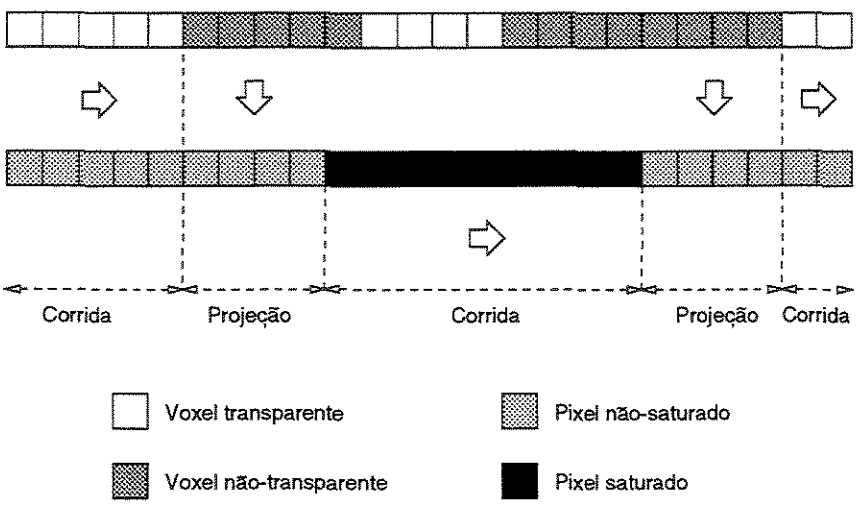


Figura 3.2: Código de corrida na cena e na imagem intermediária.

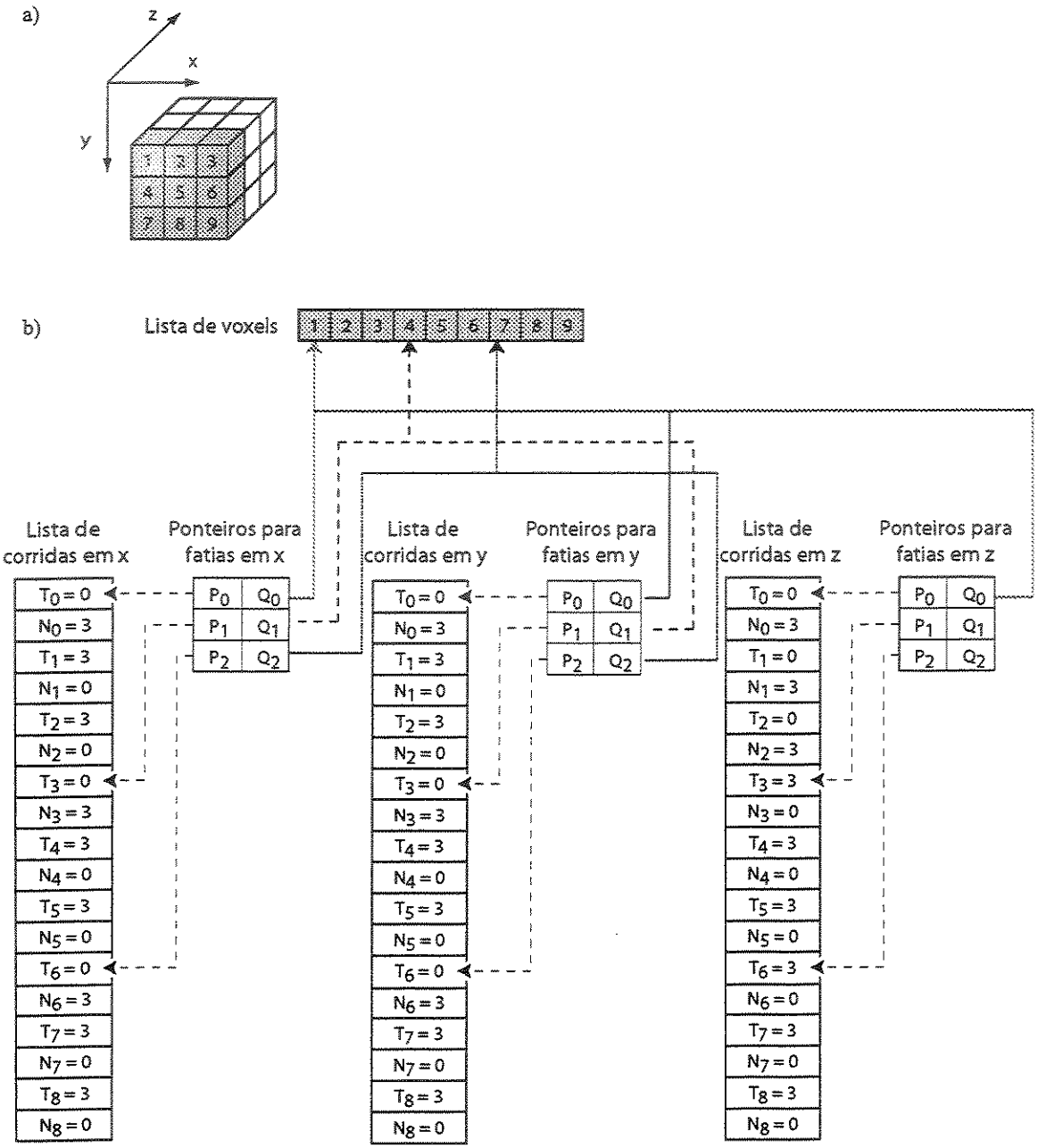


Figura 3.3: (a) Exemplo de uma cena $9 \times 9 \times 9$ com voxels numerados de 1 a 27. (b) A estrutura de dados do método *shear-warp rendering* codifica a cena em corridas para o exemplo ilustrado em (a) com voxels numerados de 1 a 9. As corridas transparentes (T_i) e não transparentes (N_i) são codificadas em listas ao longo de cada eixo. Os ponteiros P_i apontam para as fatias de cada eixo e os ponteiros Q_i apontam para o primeiro voxel da corrida na lista de voxels.

O eixo principal é x	
Octante	Ordem de acesso
1	$y^- \rightarrow y^+, z^- \rightarrow z^+, x^- \rightarrow x^+$
2	$y^- \rightarrow y^+, z^- \rightarrow z^+, x^+ \rightarrow x^-$
3	$y^+ \rightarrow y^-, z^- \rightarrow z^+, x^- \rightarrow x^+$
4	$y^+ \rightarrow y^-, z^- \rightarrow z^+, x^+ \rightarrow x^-$
5	$y^- \rightarrow y^+, z^+ \rightarrow z^-, x^- \rightarrow x^+$
6	$y^- \rightarrow y^+, z^+ \rightarrow z^-, x^+ \rightarrow x^-$
7	$y^+ \rightarrow y^-, z^+ \rightarrow z^-, x^- \rightarrow x^+$
8	$y^+ \rightarrow y^-, z^+ \rightarrow z^-, x^+ \rightarrow x^-$

O eixo principal é y	
Octante	Ordem de acesso
1	$x^- \rightarrow x^+, z^- \rightarrow z^+, y^- \rightarrow y^+$
2	$x^+ \rightarrow x^-, z^- \rightarrow z^+, y^- \rightarrow y^+$
3	$x^- \rightarrow x^+, z^- \rightarrow z^+, y^+ \rightarrow y^-$
4	$x^+ \rightarrow x^-, z^- \rightarrow z^+, y^+ \rightarrow y^-$
5	$x^- \rightarrow x^+, z^+ \rightarrow z^-, y^- \rightarrow y^+$
6	$x^+ \rightarrow x^-, z^+ \rightarrow z^-, y^- \rightarrow y^+$
7	$x^- \rightarrow x^+, z^+ \rightarrow z^-, y^+ \rightarrow y^-$
8	$x^+ \rightarrow x^-, z^+ \rightarrow z^-, y^+ \rightarrow y^-$

O eixo principal é z	
Octante	Ordem de acesso
1	$x^- \rightarrow x^+, y^- \rightarrow y^+, z^- \rightarrow z^+$
2	$x^+ \rightarrow x^-, y^- \rightarrow y^+, z^- \rightarrow z^+$
3	$x^- \rightarrow x^+, y^+ \rightarrow y^-, z^- \rightarrow z^+$
4	$x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^- \rightarrow z^+$
5	$x^- \rightarrow x^+, y^- \rightarrow y^+, z^+ \rightarrow z^-$
6	$x^+ \rightarrow x^-, y^- \rightarrow y^+, z^+ \rightarrow z^-$
7	$x^- \rightarrow x^+, y^+ \rightarrow y^-, z^+ \rightarrow z^-$
8	$x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^+ \rightarrow z^-$

Tabela 3.1: Esta tabela mostra a ordem de acesso *front-to-back* do método *shear-warp rendering* para cada octante da Figura 3.4, onde o eixo principal determina uma ordem de precedência entre x, y e z, respectivamente. A notação $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indica que os voxels devem ser acessados da menor para a maior coordenada ao longo de cada eixo, x, y e z, respectivamente, para projeção *front-to-back*.

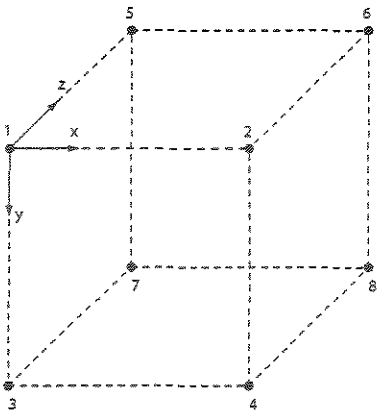


Figura 3.4: Uma cena em que cada octante tem um número de identificação de 1 a 8.

3.1.2 *Rendering*

A projeção ortogonal da fatoração *shear-warp* utiliza os seguintes passos, detalhados no Apêndice A:

1. Identificar o eixo principal de visualização (Equações A.14 e A.6) e escolher a correspondente matriz M_P que realiza a transformação do sistema de coordenadas da cena no sistema de coordenadas padrão.

2. Calcular a matriz de visualização correspondente ao eixo principal de visualização (M'_{view}) através de M_{view} e M_P (Equação A.12).

3. Calcular os fatores de *shear* (Equações A.15 e A.16) e os deslocamentos de cada fatia no sistema de coordenadas padrão.

4. Calcular a translação entre as origens das coordenadas do sistema de coordenadas padrão a da imagem intermediária (Figura A.3).

5. Calcular a matriz de *shear* (Equação A.17) e a matriz de *warp* (Equação A.26).

6. Aplicar a transformação de *shear* em cada fatia da cena no sistema de coordenadas padrão percorrendo a estrutura de dados de acordo com a tabela 3.1. A transformação *shear* 3D pode ser realizada por uma tabela de *look-up*, onde são pré-calculados os deslocamentos de cada fatia no plano ij;

7. Projetar cada fatia na imagem intermediária, percorrendo corridas opacas e saltando corridas transparentes na cena. A imagem intermediária é simultaneamente percorrida pelas linhas formadas pelas projeções dos voxels, saltando as corridas de pixels com opacidade acumulada saturada, como mostra o exemplo da Figura 3.2.;

8. Aplicar a transformação *warp* na projeção intermediária para gerar a imagem final.

A projeção da cena na imagem intermediária tem propriedades geométricas que simplificam o algoritmo de *rendering*:

Propriedade 1: As linhas de varredura da imagem intermediária são paralelas às linhas de varredura dos voxels na cena (Figura 3.5).

Propriedade 2: Todos os voxels em uma dada fatia são escalonados de um mesmo fator.

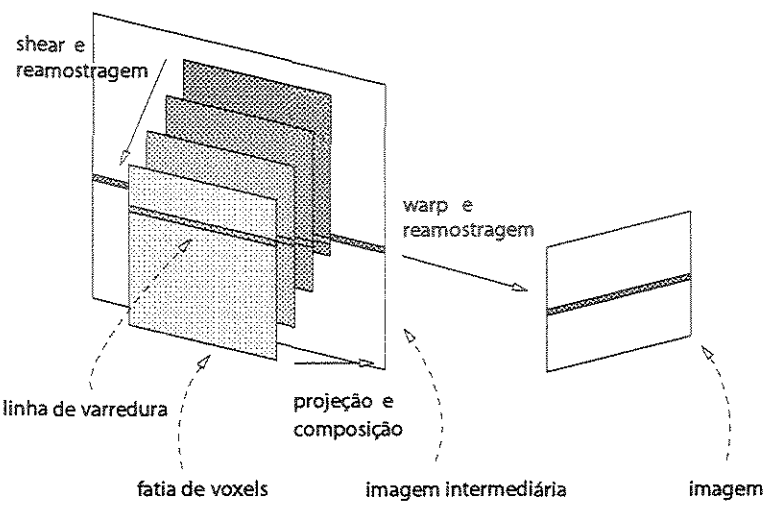


Figura 3.5: As linhas de varredura da imagem intermediária são paralelas às linhas de varredura dos voxels na cena.

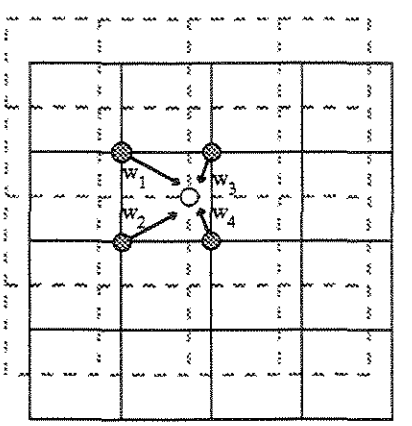


Figura 3.6: Em projeção ortogonal, cada voxel em uma dada fatia da cena tem o mesmos pesos de reamostragem w_1, w_2, w_3, w_4.

A Propriedade 1 provém do fato que a matriz de *shear* não contém somente deslocamentos no plano ij. A Propriedade 2 provém do fato que a matriz de *shear* faz o escalonamento uniforme de cada fatia.

Uma implicação útil dessas propriedades para projeções paralelas é que cada voxel em uma dada fatia da cena tem o mesmo peso de reamostragem (Figura 3.6). Como cada fatia é simplesmente transladada, um conjunto de pesos de reamostragem podem ser pré-calculados e então reutilizados para cada voxel da fatia. Este fato elimina os problemas associados com reamostragem eficiente em algoritmos de *rendering* que utilizam a técnica *voxel splatting*.

3.2 *Shell Rendering*

O método *shell rendering* [6], proposto por Udupa e Odhner, é apresentado nesta seção nos aspectos de representação e *rendering*. *Shell* é uma estrutura de dados especial que armazena somente os voxels que potencialmente serão visualizados. Uma característica desta estrutura que torna o *rendering* bastante rápido em relação a outros métodos, é a rapidez de acesso ao voxels e seus atributos.

3.2.1 Representação

Para uma dada cena, *shell rendering* usa uma estrutura compacta, chamada *shell*, que armazena apenas os voxels da borda do objeto. Esses voxels são selecionados entre aqueles que têm opacidade α no intervalo de opacidades (α_l, α_h) da seguinte forma: um voxel v é armazenado no shell se $\alpha(v) > \alpha_l$ e se v não é completamente cercado de voxels v' tais que $\alpha(v') \geq \alpha_h$. Além do valor de opacidade a estrutura *shell* armazena os seguintes atributos de visualização para cada voxel: sua coordenada x na cena, o vetor normal à superfície que passa pelo voxel, um bit de visibilidade que indica quando o voxel é visível em uma dada operação de manipulação, e o índice do objeto que o voxel pertence. Esta última informação é escolhida para saber se o objeto é visível dada uma operação de manipulação, qual sua cor, entre outras informações associadas ao objeto que contém o voxel em questão. Os atributos de visualização são armazenados em uma lista V_x, começando de $(x, y, z) = (0, 0, 0)$, em uma ordem x-por-x, y-por-y, z-por-z das coordenadas dos voxels da cena. Além da lista v_x, a estrutura *shell* utiliza um arranjo 2D de ponteiros P_{zy} para indicar o primeiro voxel na lista associada com uma coordenada particular (y, z) na cena. Figura 3.7b ilustra um simples exemplo de uma estrutura *shell* para uma cena de tamanho $3 \times 3 \times 3$ voxels. Figura 3.7a mostra a cena onde cada voxel é numerado de 1 a 27, seguindo a ordem x-por-x, y-por-y, z-por-z. Os voxels do *shell* são aqueles numeradas de 1 até 9. Figura 3.7b mostra a lista V_x e o arranjo 2D de ponteiros para este exemplo.

Um exemplo 2D da forma de armazenamento do *shell* pode ser visto na Figura 3.8. Na parte superior, a figura mostra uma cena 2D onde os voxels pertencentes ao *shell* aparecem escurecidos. Na parte inferior, a figura mostra uma ilustração da estrutura *shell*, com o vetor de ponteiros P_y e a lista de voxels V_x para esse exemplo.

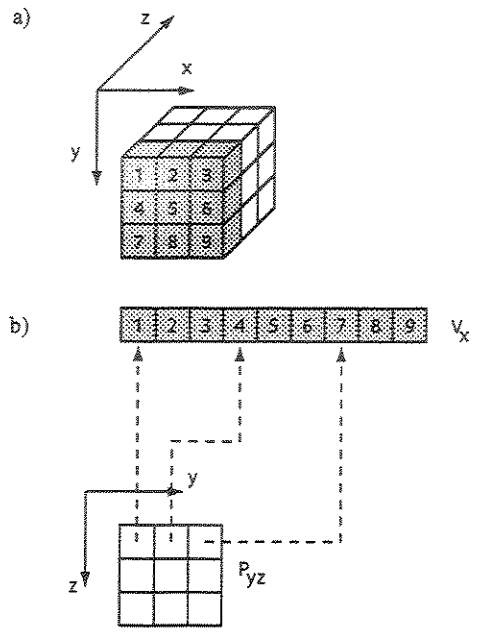


Figura 3.7: (a) Exemplo de uma cena $9 \times 9 \times 9$ com voxels numerados de 1 a 27. (b) Estrutura *shell* para o exemplo ilustrado em (a) com voxels numerados de 1 a 9.

Dentre os atributos armazenados em V_x, a normal estimada pode ser substituída por um índice para uma tabela de normais pré-calculadas com o objetivo de reduzir sua representação computacional em cada voxel, conforme visto na seção 2.5.3. O bit de visibilidade indica se o voxel é visível ou não após operação de manipulação tais como cortes por superfícies. Neste trabalho, a estrutura *shell* consiste apenas da opacidade, coordenada x e índice da normal.

3.2.2 *Rendering*

Em *shell rendering*, a técnica adotada para projeção ortogonal deve ser *voxel splatting*. Apesar da estrutura *shell* manter o registro das coordenadas originais dos voxels armazenados, a informação sobre o posicionamento relativo entre esses voxels é perdida. Isto se deve à codificação realizada pelo *shell*, onde apenas uma parte dos voxels da cena classificada são armazenados. Esta característica torna o emprego da técnica de *ray-casting* computacionalmente inviável. Nessa situação, a tarefa de localizar no *shell* os voxels atra-

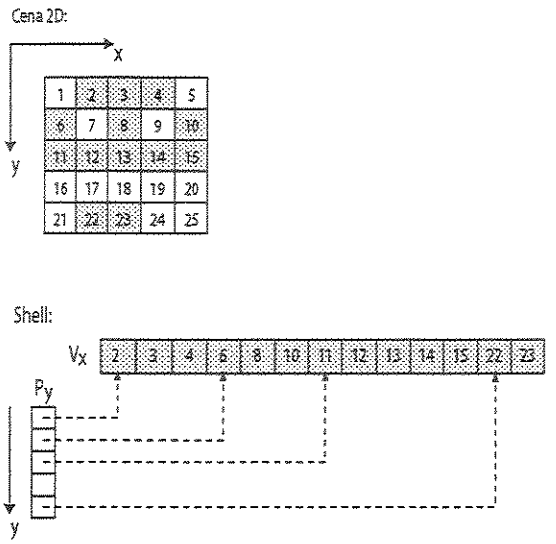


Figura 3.8: Exemplo de uma cena 2D 5×5 com voxels numerados de 1 a 25. Estrutura *shell* é formada pelo vetor de ponteiros P_y e pela lista de voxels V_x

vessados por um determinado raio de busca, é extremamente computacionalmente custosa pois requer uma busca binária em V_x para cada coordenada x a ser avaliada sobre o raio.

Dentre as técnicas de remoção de superfície para *voxel splatting*, este trabalho emprega a técnica *front-to-back* (FTB), pois apresenta uma projeção ortogonal rápida. *Shell rendering* explora o fato que para projeção ortogonal, "não há ordem de preferência entre os eixos x, y e z"[1]. Assim, como os voxels na lista V_x são armazenados ao longo do eixo x, é mais rápido projetá-los no plano de visualização percorrendo o eixo$x$ antes de percorrer os eixos y e z. Figura 3.4 ilustra uma cena com seu sistema de coordenadas, onde cada octante tem um número de identificação de 1 a 8. A ordem de acesso dependerá de qual octante o observador está situado. A Tabela 3.2 mostra a ordem de acesso *front-to-back* em cada octante na Figura 3.4. A notação $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indica que os voxels devem ser acessados da menor para a maior coordenada ao longo de cada eixo, x, y e z, respectivamente, para projeção ortogonal *front-to-back*.

Um método simples utilizado para identificar o octante mais próximo é calcular o octante enumerado com i, $i = 1, 2, \ldots 8$, cujo respectivo produto interno entre os vetores $\vec{d}$ e $\vec{c_i}$ é mínimo, onde $\vec{d}$ é o vetor da direção de visualização e $\vec{c_i}$ é o vetor unitário na direção que via do centro da cena até o vértice correspondente ao octante i. Durante este cálculo, o vetor $\vec{d}$ é rotacionado de $-\phi$ e $-\theta$ nos eixos y e x, respectivamente, para simular a rotação da cena em relação ao observador. Outra forma de identificar os octantes utilizados na varredura *front-to-back* é aplicar a matriz de visualização nas coordenadas

[1]No capítulo 5, mostra-se que esta premissa não é sempre verdadeira para *voxel splatting*. Este erro conceitual é comum em muitos outros métodos de visualização.

Octante	Ordem de acesso
1	$x^- \rightarrow x^+, y^- \rightarrow y^+, z^- \rightarrow z^+$
2	$x^+ \rightarrow x^-, y^- \rightarrow y^+, z^- \rightarrow z^+$
3	$x^- \rightarrow x^+, y^+ \rightarrow y^-, z^- \rightarrow z^+$
4	$x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^- \rightarrow z^+$
5	$x^- \rightarrow x^+, y^- \rightarrow y^+, z^+ \rightarrow z^-$
6	$x^+ \rightarrow x^-, y^- \rightarrow y^+, z^+ \rightarrow z^-$
7	$x^- \rightarrow x^+, y^+ \rightarrow y^-, z^+ \rightarrow z^-$
8	$x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^+ \rightarrow z^-$

Tabela 3.2: Esta tabela mostra a ordem de acesso *front-to-back* do método *shell rendering* para cada octante da Figura 3.4, onde o eixo principal determina uma ordem de precedência entre x, y e z, respectivamente. A notação $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indica que os voxels devem ser acessados da menor para a maior coordenada ao longo de cada eixo, x, y e z, respectivamente, para projeção *front-to-back*.

(x, y, z) de cada octante para calcular um conjunto de coordenadas transformadas z' . O octante mais próximo é aquele cuja respectiva coordenada transformada z' é a menor.

Capítulo 4

Shell Rendering com fatoração *Shear-Warp*

Foi observado no capítulo 3 que a transformação *shear* 3D pode ser realizada por uma tabela de *look-up*, e que a transformação *warp* 2D é feita apenas nos pixels projetados na imagem intermediária. No contexto de *shell rendering*, isto evita as multiplicações da matriz de visualização para todos os voxels do *shell* aumentando a eficiência do método. Como a representação do *shell rendering* é evidetemente mais compacta que a representação do *shear-warp rendering*, o desafio principal deste trabalho foi esteder o método *shell rendering* para se beneficiar da fatoração *shear-warp*. Este capítulo, portanto, apresenta o novo método híbrido, denominado *shell rendering* com fatoração *shear-warp*.

4.1 Representação

A principal dificuldade de incorporar a fatoração *shear-warp* no método *shell rendering* é a ordem de precedência fixa entre os eixos x, y e z, pois conforme descrito na Seção 3.2.2. Isto causa uma dificuldade no *shell rendering* sempre que o eixo principal for o eixo x. Como os voxels estão armazenados em V_x na ordem crescente de x de acordo com a tabela 3.1.

A incorporação da fatoração *shear-warp* no método *shell rendering* é possível através da adição de um segundo arranjo 2D de ponteiros P_{xz} e uma segunda lista de voxels V_y, como ilustrado na Figura 4.1. Acessando os voxels da cena, começando da coordenada $(x, y, z) = (0, 0, 0)$, em uma ordem y-por-y, z-por-z, e x-por-x, os voxels da segunda lista (V_y) armazenam a coordenada y de cada voxel do shell e um ponteiro para a sua posição correspondente na primeira lista (V_x). Similarmente, cada ponteiro no arranjo 2D P_{xz} indica o primeiro voxel na segunda lista associado com a coordenada (x, z) da cena. Note que, os outros atributos de visualização e manipulação são armazenados apenas na lista

V_x. Então, o arranjo 2D de ponteiros P_{xz} e a lista V_y devem ser usadas quando o eixo principal é o eixo x. Esta estrutura de dados é denominada *E-shell* [29].

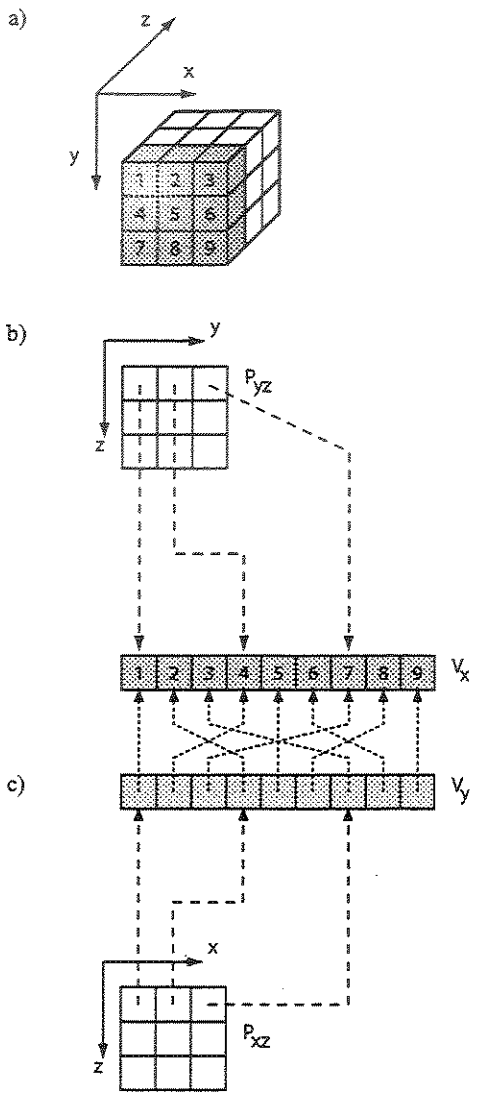


Figura 4.1: (a) Exemplo de uma cena $9 \times 9 \times 9$ com voxels numerados de 1 a 27. (b) Estrutura *E-shell* para o exemplo ilustrado em (a) com voxel numerados de 1 a 9.

4.2 *Rendering*

Em *shell rendering* com fatoração *shear-warp*, a operação de *shear* 3D é implementada por uma tabela de *look-up*, evitando as multiplicações ponto-flutuante da matriz de visualização para cada voxel no *shell* O número de multiplicações é reduzido para o número de

operações de uma transformação de *warp* 2D da imagem intermediária. Infelizmente, outras otimizações, como *early ray termination*, são ainda ineficientes neste método devido às razões já citadas na seção 3.2.2.

Um aspecto de otimização importante da fatoração *shear-warp* é a possibilidade de implementar em *voxel splatting*, a técnica *early ray termination*, comum dos métodos que utilizam *ray casting*. Esta propriedade é facilmente explorada em *shear-warp rendering* como descrito na capítulo anterior, mas infelizmente não pode ser explorada no caso da estrutura *E-shell*.

Como descrito na seção 3.1.1, a técnica de *early ray termination* é implementada através de corridas de pixels com opacidade acumulada saturada na imagem intermediária. Estas corridas possibilitam saltos na imagem intermediária e saltos ainda maiores de voxels da cena que seriam projetados sobre esses pixels. No caso do *shear-warp rendering* os saltos na cena são facilmente implementados devido a representação da cena ser também em código de corrida. Já no caso do *shear-warp shell rendering*, cada salto implica em uma busca binária em V_x (ou V_y), do próximo voxel da estrutura E-shell a ser projetado. Na prática, essas corridas na imagem intermediária geralmente são curtas, o que demanda muitas buscas em V_x (ou V_y), tornando a implementação de *early ray termination* inviável.

O *rendering* usando a estrutura *E-shell* pode ser resumida nos seguintes passos:

1. Identificar o eixo principal de visualização (Equações A.14 e A.6) e escolher a correspondente matriz M_P de permutação.
2. Calcular a matriz de visualização permutada através de M_{view} e M_P (Equação A.12).
3. Calcular os coeficientes de *shear* (Equações A.15 e A.16) através de matriz de visualização permutada.
4. Calcular a translação entre a origem do sistema de coordenadas padrão a e a origem da imagem intermediária (Figura A.3).
5. Calcular a matriz de *shear* (Equação A.17) e a matriz de *warp* (Equação A.26).
6. Determinar a ordem de varredura entre os eixos x, y e z;
7. Determinar o sentido de varredura nos eixos x, y e z;
8. Fazer a varredura na estrutura E-shell;
9. Aplicar a transformação de *shear* em cada fatia no sistema de coordenadas padrão;
10. Projetar cada voxel na imagem intermediária e tonalizar voxels visíveis;
11. Aplicar a transformação de *warp* dos voxels tonalizados para gerar a imagem final.

A formação das imagens intermediária e final são detalhadas a seguir.

Formação da imagem intermediária A formação da imagem intermediária é feita durante a varredura da estrutura E-shell. Como os deslocamentos de *shear* são iguais para todos os voxels de uma mesma fatia, estes são guardados em um tabela de *look-up* para serem utilizados durante a projeção. Apesar de ser necessário recalcular esta tabela toda vez que a direção de visualização é alterada, as operações de *shear* requerem aritmética simples como adição de inteiros.

Formação da imagem A imagem é formada pela aplicação da matriz de *warp* 2D através (Equação A.26) nos pixels da imagem intermediária. As coordenadas (i_p, j_p) de cada pixel p tonalizado na imagem intermediária são transformadas em coordenadas (u_p, v_p) na imagem. Para evitar o aparecimento de artefatos, a composição de pixels da imagem é feita com a relação de um pixel para 3×3 pixels. Os voxels que não têm a opacidade acumulada saturada na imagem são tonalizados com o modelo de iluminação de *Phong* e compostos na imagem (Equação 2.13).

Algoritmo de *rendering* O algoritmo para *shell rendering* com fatoração *shear-warp* é descrito a seguir. Cada passo possui uma descrição em alto nível e um detalhamento do passo entre parênteses.

Entrada:

θ e ϕ: Ângulos de *tilt* e *spin* da cena, respectivamente;

$\vec{t}$: Vetor de translação que centraliza a cena na origem do sistema de coordenadas;

oct_0: Coordenadas do ponto referente ao octante mais próximo do observador;

oct_n: Coordenadas do ponto referente ao octante mais afastado do observador;

V_x: Lista de atributos de voxels ao longo do eixo x;

V_y: Lista de ponteiros para V_x ao longo do eixo y;

e_p: Identificador do eixo principal de visualização;

$M_P(c)$: Matriz de permutação correspondente ao eixo principal;

P_{zy}: Arranjo 2D de ponteiros para a lista V_x no plano zy;

P_{xz}: Arranjo 2D de ponteiros para a lista V_y no plano xz;

Variáveis auxiliares:

M_R: Matriz de rotação (Equação 2.10);

M_T: Matriz de translação (Equação 2.5);

M_{view}: Matriz de visualização (Equação 2.12);

M_{shear}: Matriz de *shear* (Equação A.17);

M_{warp}: Matriz de *warp* (Equação A.26);

V_i: Identificador da lista em uso (V_x ou V_y);

P_{kj}: Identificador do arranjo 2D em uso: (P_{zy} ou P_{xz});

i_0, j_0, k_0: Coordenadas iniciais de varredura no sistema de coordenadas padrão;

i_n, j_n, k_n: Coordenadas finais de varredura no sistema de coordenadas padrão;

v_0, v_n: Índices que determinam um intervalo de corrida em V_i;

v, ℓ: Índices que determinam o voxel corrente em V_i e V_x, respectivamente;

γ: Ângulo entre o vetor de reflexão $\vec{r}$ e o vetor de direção de visualização $\vec{d}$;

β: Ângulo igual a 2γ;

I_{dist}: Influência da distância da fonte de luz (Equação 2.14);

α: Valor de opacidade;

α_{shear}: *Buffer* de opacidade na imagem intermediária;

α_{warp}: *Buffer* de opacidade na imagem.

Funções:

f_{phong}: Função que retorna o cálculo da tonalização de Phong (Equação 2.15);

f_{dist}: Função que retorna o cálculo de I_{dist} em uma tabela de *look-up* (Equação 2.14).

Saída:

I: Imagem final.

Início

1. Calcule matriz M_{view};
   $(M_{view} \leftarrow M_R(\theta, \phi) \cdot M_T(\vec{t}))$

2. Calcule matrizes M_{shear} e M_{warp};
   $(M_{view} \rightarrow M_{warp} \cdot M_{shear})$

3. Calcule os octantes mais próximo e mais afastado do observador;
   $(oct_0 \leftarrow \max_z(oct_i), i \in \{1, 2, \ldots 8\})$
   $(oct_n \leftarrow \min_z(oct_i), i \in \{1, 2, \ldots 8\})$

4. Calcule o eixo principal de visualização;
   $(e_p \in \{x, y, z\} \mid \max(d_x, d_y, d_z) \mapsto e_p)$

5. Transforme o sistema de coordenadas dos octantes mais próximo e mais afastado do observador para o sistema de coordenadas padrão;
   $([i_0, j_0, k_0]^T \leftarrow M_P(e_p) \cdot [oct_0.x, oct_0.y, oct_0.z]^T)$
   $([i_n, j_n, k_n]^T \leftarrow M_P(e_p) \cdot [oct_n.x, oct_n.y, oct_n.z]^T)$

6. Determine P_{kj};
   $(P_{kj} \mid P_{kj} \in \{P_{zy}, P_{xz}\})$

7. Determine V_i;
   $(V_i \mid V_i \in \{V_x, V_y\})$

8. **Para** $k \leftarrow k_0$ **até** k_n **faça**

9. **Para** $j \leftarrow j_0$ **até** j_n **faça**

10. Encontre o intervalo $[v_0, v_n]$ em V_i com coordenadas (k, j);
    $(P_{kj}(k, j) \mapsto [v_0, v_n])$;

11. **Para** $v \leftarrow v_0$ **até** v_n **faça**

12. Ache indice da lista V_x;
    $(\ell \leftarrow V_i[v].indice)$

13. Ache coordenada i;
$(i \leftarrow V_i[\ell].coordenada)$

14. Ache a opacidade do voxel;
$(\alpha \leftarrow V_x[\ell].opacidade)$

15. Ache o pixel de projeção de *shear* na imagem intermediária;
$((u_i, v_i) \leftarrow M_{shear} \cdot (i, j))$

16. **Se** a opacidade no pixel da imagem intermediária não está saturada **então**
$((\alpha_{shear}(u_i, v_i) \approx 0)$ é *falso*$)$

17. Calcule pixel na imagem final;
$((u, v) \leftarrow M_{warp} \cdot (u_i, v_i))$

18. **Se** a opacidade no pixel da imagem final não está saturada **então**
$((\alpha_{warp}(u, v) \approx 0)$ é *falso*$)$

19. Calcule I_{dist};
$(I_{dist} \leftarrow f_{dist}(k))$

20. Calcule γ;
$(\gamma \leftarrow \arccos(V_x[\ell].\vec{n} \cdot \vec{d}))$

21. Calcule β;
$(\beta \leftarrow 2\gamma)$

22. Calcule tonalização do pixel;
$(I(u_i, v_i) \leftarrow f_{phong}(\beta, I_{dist}, k_a, k_s, k_d, n))$

23. Faça a composição do(s) pixel(s) na imagem final que não tem opacidade saturada;
$(I(u, v) \leftarrow I(u, v) + (I(u_i, v_i) \cdot \alpha_{warp}(u, v))$

24. Atualize opacidade da imagem final;
$(\alpha_{warp}(u, v) \leftarrow \alpha_{warp}(u, v) + (1 - \alpha))$

25. **fim Se**

26. Atualize opacidade da imagem intermediária.
$(\alpha_{shear}(u_i, v_i) \leftarrow \alpha_{warp}(u, v))$

27. **fim Se**

28. **fim Para**

29. **fim Para**

30. **fim Para**

Fim

A seguir segue uma descrição das principais etapas do algoritmo proposto:

Linhas 1-5: Calculam as variáveis relacionadas com a direção de visualização;

Linhas 6-7: Determinam as estruturas utilizadas para varredura da cena;

Linha 8: Laço de varredura na profundidade do sistema de coordenadas padrão;

Linhas 9-11: Laços de varredura nas fatias do sistema de coordenadas padrão;

Linhas 12-14: Determinação do voxel através dos arranjos 2D de ponteiros P_{zy} ou P_{xz} e a respectivas listas V_x e V_y. Os atributos do voxel é encontrado através do índice ℓ em V_x;

Linha 15: Projeção do voxel na imagem intermediária. Cada fatia é deslocada de um valor associado com a profundidade em uma tabela de *look-up*;

Linhas 16: Para evitar cálculos desnecessários de *warp*, é utilizado um *buffer* de opacidade acumulada α_{shear} na imagem intermediária;

Linha 17: As coordenadas do pixel na imagem intermediária são transformadas em coordenadas na imagem final através da matriz de transformação 2D M_{warp}. A transformação *warp* é feita com a origem no centro da imagem intermediária.

Linha 18: É utilizado um *buffer* de opacidade acumulada α_{warp} na imagem para evitar cálculos desnecessários de tonalização;

Linhas 19-22: Cálculos para a tonalização do pixel;

Linhas 24: É feita a composição de um ou mais pixels da imagem final, de acordo com a relação de escala entre pixel e voxel;

Linhas 25-26: Os *buffers* de opacidade são atualizados.

Capítulo 5

Análise Comparativa

Este capítulo apresenta uma análise comparativa dos métodos relacionados *shell rendering* (SR), *shear-warp rendering* (SWR) e *shell rendering* com fatoração *shear-warp* (SWSR), com relação a três aspectos: requerimento de memória, velocidade de *rendering* e qualidade de imagem.

Os experimentos foram realizados com 10 objetos O_i, $i = 1, 2, \ldots, 10$, de diferentes tamanhos e derivados de cenas constituindo diferentes modalidades, conforme a descrição na Tabela 5.1. Para cada objeto foram selecionadas duas representações: uma de superfície com 1 voxel de espessura e outra de volume com 3 voxels de espessura em torno da superfície. Os voxels são classificados como opacos na representações de superfície e com opacidade 1/3 nas representações de volume. Isto garante que todos os métodos operam com o mesmo número de voxels por representação. O número de voxels nessas representações varia de 22.390 a 2.833.821 voxels, cobrindo todos os casos de pequenas a grandes estruturas. A Tabela 5.1 também mostra em parênteses a porcentagem de redução no número de voxels das representadas comparadas com as suas respectivas cenas originais. Isto ilustra a vantagem de armazenar e processar somente os voxels significativos para a visualização.

5.1 Espaço de memória

A Tabela 5.2 mostra o requerimento de memória em Kbytes necessários para o armazenamento das superfícies e volumes apresentados na Tabela 5.1 nos métodos SR, SWR e SWSR. É importante notar que SWR requer de 2, 21 (superfície de O_5) até 8, 14 (superfície de O_7) mais espaço de memória que SR. SWSR representa uma solução intermediária de armazenamento. Sua economia em espaço de memória varia de 27% (volume de O_8) até 75% (superfície de O_4, O_7, O_8 e O_9) quando comparado com SWR, enquanto este requer de $1,38$ (superfície de O_5) até $2,0$ (superfícies de $O_2, O_4 - O_7$ e O_9) vezes mais espaço de

Objeto	Cena	Superifície	Volume
O_1 (joelho - ossos - CT)	3.300.300	134.370 (95,93%)	241.953 (92,67%)
O_2 (talus - MR)	8.584.216	22.390 (99,74%)	72.946 (99,15%)
O_3 (crânio - CT)	11.769.912	261.676 (97,78%)	836.889 (92,89%)
O_4 (cabeça - pele - MR)	11.927.552	419.186 (96,49%)	1.349.048 (88,69%)
O_5 (veias - MRA)	16.777.216	29.753 (99,82%)	65.092 (99,61%)
O_6 (órbitas - pele - CT)	25.165.824	250.974 (99,00%)	791.090 (96,86%)
O_7 (órbitas - ossos - CT)	25.165.824	263.961 (98,95%)	763.071 (96,97%)
O_8 (crânio de criança - MR)	57.451.680	933.061 (98,38%)	2.833.821 (95,07%)
O_9 (cabeça e espinha - ossos - CT)	77.332.480	745.824 (99,04%)	2.119.109 (97,26%)
O_{10} (cabeça e espinha - pele - CT)	77.332.480	780.135 (98,99%)	2.180.696 (97,18%)
Total	314.808.484	3.841.330 (98,78%)	11.253.715 (96,43%)

Tabela 5.1: Os objetos selecionados para os experimentos, o número de voxels após reamostragem da superfície e do volume, respectivamente. O percentual de redução do número de voxels, entre parênteses, com relação às respectivas cenas originais em parênteses.

memória que SR.

A característica de armazenamento do SWR é baseada na codificação de corridas transparentes e opacas da cena. Se o objeto possui muitas transições entre objeto e fundo, o gasto decorrente da codificação é grande. Para tentar minimizar este problema, cada corrida é codificada em até 255 comprimentos, o necessário para armazenar em apenas 1 byte. Como conseqüência, corridas com comprimento maior que 255 voxels são divididas em duas ou mais corridas. A estrutura de ponteiros que aponta para a primeira corrida em cada fatia é também modificada para apontar para a primeira corrida em cada linha, com o propósito de facilitar o acesso aos voxels, uma vez que é necessário percorrer várias corridas em uma fatia para ter acesso a um determinado voxel.

SR utiliza um arranjo 2D de ponteiros onde cada célula necessita de uma representação de 4 bytes para acessar os elementos da lista de voxels, onde cada voxel armazena os atributos de visualização. Portanto é uma estrutura bem mais compacta que SWR, conforme mostra a Tabela 5.2.

Em SWSR, são adicionados à estrutura *shell* original, um segundo arranjo 2D de ponteiros e uma segunda lista de ponteiros para a lista de voxels, que quase duplicam a quantidade de memória do método original. Em aplicações em que lista de voxels possui um maior número de atributos para visualização, a estrutura *E-shell* tem um custo menor que o apresentado na Tabela 5.2.

Objeto	Supefície			Volume		
	SR	SWR	SWSR	SR	SWR	SWSR
O_1	600	4.404	1.192	1.324	4.328	2.505
O_2	218	1.164	437	618	1.504	974
O_3	1.260	9.640	2.473	4.515	12.720	8.601
O_4	1.819	14.732	3.639	6.951	19.204	13.538
O_5	372	1.776	744	830	1.832	1.148
O_6	1.172	9.132	2.345	4.247	13.256	8.109
O_7	1.223	9.952	2.446	4.110	10.972	7.836
O_8	4.284	34.104	8.412	14.959	39.436	28.796
O_9	3.503	27.772	7.007	11.527	32.576	21.874
O_{10}	3.637	25.468	7.275	11.828	36.148	22.476
Média	1.809	13.814	3.597	6.091	17.198	11.586

Tabela 5.2: Requerimentos de memória em Kbytes para armazenar a superfícies e o volumes nas três abordagens.

5.2 Velocidade de *rendering*

Os experimentos foram realizados em um PC Pentium 1GHz com 512 Mbytes de memória RAM. As implementações dos métodos SR e SWR foram comparadas com a implementação de SWR feita por Lacroute, disponível em [67]. Em todos os experimentos é utilizada projeção ortogonal com uma única fonte de luz na mesma posição do observador. Os métodos foram implementados em linguagem C com opções iguais de otimização para a compilação dos programas. As superfícies e volumes de cada objeto são os mesmos para todas as abordagens, garantindo coerência nos experimentos. Em todos os programas é utilizada a função gettimeofday() para mensurar o tempo médio do cálculo de *rendering* de 100 imagens de mesmo tamanho em diferentes direções de visualização. É importante observar que o tempo necessário para classificação e representação não são considerados nestes experimentos. Os tempos gastos em milisegundos para cada *rendering* nas três abordagens são mostrados na Tabela 5.3. Desta tabela várias observações podem ser feitas.

Todos os métodos conseguem fazer *rendering* de superfície em menos de $0,5$ segundo, e *rendering* de volume em menos de $1,0$ segundo, com exceção do SR que requer $1,42$ segundos para o *rendering* de volume do objeto O_8. Note que, apesar de explorar mais propriedades da fatoração *shear-warp*, SWR não é sempre mais rápido que SR e SWSR. Ao contrário, as superfícies dos objetos O_2, O_5, O_6 e O_7, SR pode ser até $2,05$ vezes mais rápido (objeto O_5) que SWR, e SWSR é sempre mais rápido que SWR para todas as

Objeto	Superfície			Volume		
	SR	SWR	SWSR	SR	SWR	SWSR
O_1	76	49	45	135	75	86
O_2	32	54	32	60	62	49
O_3	144	119	99	426	198	248
O_4	208	127	115	639	229	333
O_5	41	83	44	66	88	64
O_6	168	209	152	478	292	351
O_7	173	189	136	480	257	290
O_8	475	380	317	1421	647	873
O_9	430	379	286	1226	597	729
O_{10}	421	408	330	1175	646	857
Média	216,8	199,7	155,6	610,6	309,1	388

Tabela 5.3: Os tempos gastos em milisegundos (ms) para o *rendering* de superfície e volume nas três abordagens.

superfícies. Isto significa que *early ray termination* parece não ser tão útil no caso de superfícies. Um erro comum em muitos trabalhos publicados é pensar que a velocidade do método de visualização é sempre inversamente proporcional ao número de voxels. Um contra-exemplo é ilustrado nas Figuras 5.1 e 5.2, onde as superfícies dos objetos são apresentadas em ordem crescente de número de voxels, respectivamente. Em SWR, por exemplo, a velocidade de *rendering* depende fortemente de características topológicas e morfológicas do objeto. Representações que têm grandes corridas de voxels opacos têm a tendência de aumentar a eficiência do SWR, por causa do *early ray termination.*

Por outro lado, a velocidade em SWR e SWSR é certamente menos afetada pelo número de voxels quando comparadas ao SR, pois os dois primeiros métodos usam *voxel splatting* através de operações *look-up* enquanto SR calcula as multiplicações da matriz de visualização. Note que, SWR e SWSR podem ser $2,79$ e $1,92$ vezes mais rápidos que SR, respectivamente (objeto O_4).

Em média, SWSR e SWR são mais rápidos que SR, mas somente ao custo de mais requerimento de memória, e SWSR oferece a melhor relação de compromisso *velocidade* $\times$ *memória* entre eles.

A Tabela 5.4 apresenta ao ganho em velocidade (entrada esquerda) e custo de memória (entrada direita) de SWSR sobre SR para superfícies e volumes. Podemos observar que em média SWSR é $1,35$ e $1,52$ mais rápido que SR para superfície e volume, respectivamente. O ganho de velocidade tem a penalidade de cerca de 2 vezes mais no requisito de memória. De acordo com a Tabela 5.2, isto é certamente não proibitivo em computadores pessoais

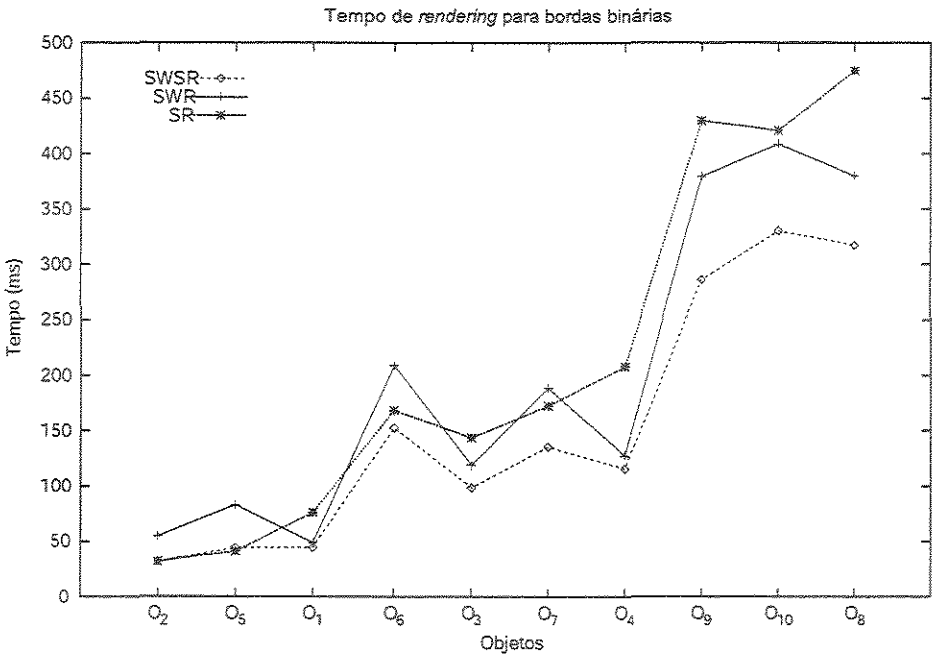


Figura 5.1: Tempo de rendering dos três métodos para superfícies em função de seu tamanho. Objetos são apresentados em ordem crescente de número de voxels ao longo do eixo horizontal.

modernos, onde a memória RAM disponível pode alcançar 1 Gbyte. Tabela 5.5 apresenta o ganho de velocidade (entrada esquerda) e o requisito de memória (entrada direita) de SWSR sobre SWR para superfície e volume, respectivamente. Note que em média, SWSR é 1,35 mais rápido que SWR para superfícies é quase tão rápido quanto SWR para volumes. No entanto, SWSR é 71% e 34% mais compacto que SWR para superfícies e volumes, respectivamente.

5.3 Qualidade de imagem

Métodos de visualização baseados em *voxel splatting* têm um problema bastante conhecido: o aparecimento de buracos na imagem gerada para algumas direções de visualização (veja Figuras 5.3a e 5.3b). Algoritmos de pós-processamento podem ser utilizados para "fechar" estes buracos, mais o sucesso desse algoritmos não é garantido e a imagem resultante pode apresentar baixa qualidade. Um solução mais eficiente é tornar o tamanho dos voxels maior que o tamanho de pixels durante a projeção *voxel splatting*. Neste caso, no entanto, existe um erro conceitual no método SR, que afeta a qualidade da imagem e corretude do *rendering*. Este erro foi detectado no decorrer deste trabalho e é divulgado em [29].

SR explora a propriedade da projeção ortogonal de não impor ordem de precedência

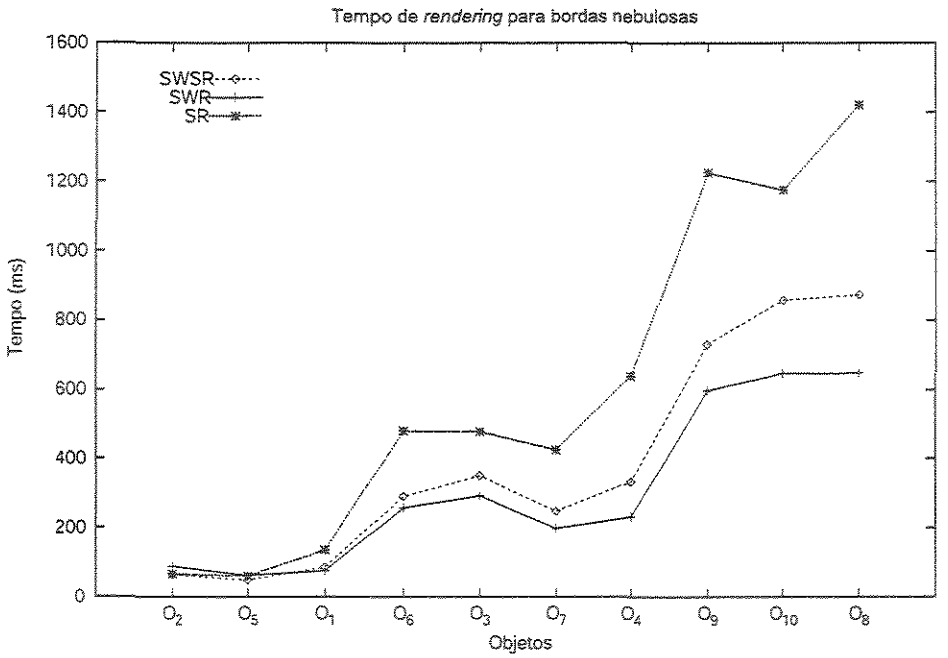


Figura 5.2: O tempo de *rendering* dos três métodos para volumes como função do seu número de voxels. Objetos são apresentados em ordem crescente de número de voxels ao longo do eixo horizontal.

entre os eixos x, y e z no acesso aos voxels. Infelizmente essa propriedade não é válida para o caso de *voxel splatting* para voxels maiores que pixels na imagem gerada, um erro comum inclusive em outros trabalhos. A Figura 5.4a ilustra um exemplo onde o observador está no octante 6, a ordem de acesso é como indicada no Tabela 3.2, o eixo principal de visualização x é perpendicular ao plano de visualização, onde cada voxel pinta 3×3 pixels na ordem correta de acesso *front-to-back*. A Figura 5.4b ilustra o resultado do acesso *front-to back* feito pelo SR para o mesmo exemplo da Figura 5.4a. Observe que quanto mais longe os voxels escondidos são pintados no lugar de voxels mais próximos e visíveis. Em *rendering* de superfície, o problema ainda tem uma solução se for utilizado o método *z-buffer* para a remoção de superfícies escondidas, mas infelizmente o problema persiste para *rendering* de volume. Esse erro conceitual é muito difícil de detectar, porque depende do tipo de representação e dos valores de opacidade dos voxels. Isto fica mais evidente quando os valores de opacidades são mais baixos, ou mesmo quando eles são mais altos, em casos quando há variações abruptas da superfície do objeto. Figura 5.5 ilustra os efeitos deste erro conceitual para SR (Figura 5.5a) comparando com a qualidade de imagem do SWSR (Figura 5.5b), para mesma direção de visualização e distribuição de opacidades. Neste caso, valores de opacidade baixos mostram que o problema aparece como manchas escuras na superfície do objeto (Figura 5.5a). Figura 5.5c ilustra outra

Objeto	Superfície	Volume
O_1	1,69 — 1,99	1,57 — 1,89
O_2	1,00 — 2,00	1,22 — 1,58
O_3	1,45 — 1,96	1,72 — 1,91
O_4	1,81 — 2,00	1,92 — 1,95
O_5	0,93 — 2,00	1,03 — 1,38
O_6	1,11 — 2,00	1,36 — 1,91
O_7	1,27 — 2,00	1,66 — 1,91
O_8	1,50 — 1,96	1,63 — 1,92
O_9	1,50 — 2,00	1,68 — 1,90
O_{10}	1,28 — 2,00	1,37 — 1,90
Média	1,35 — 1,99	1,52 — 1,82

Tabela 5.4: Ganho de velocidade (entrada esquerda) e requisito de memória (entrada direita) do SWSR sobre SR para superfícies e volumes.

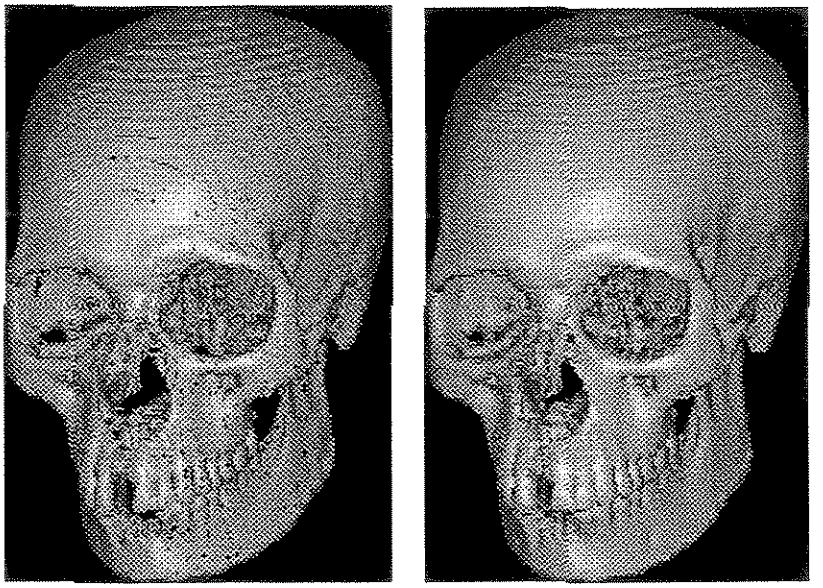
situação quando os voxels tem altos valores de opacidade. Neste caso, percebe-se um padrão claro-escuro na região mandibular onde ocorrem variações abruptas na superfície do objeto. Isto parece com artefato de *aliasing*, mas não é. De fato, o problema pode aparecer sempre que o eixo principal não é acessado por último. Neste sentido, SWSR e SWR oferecem melhor qualidade de imagem que SR, pois usam uma ordem acesso ao voxel como indicado na Tabela 3.1.

SWR apresenta imagens de qualidade semelhantes à imagens produzidas por algoritmos de *ray-casting*. Apesar de ser um método *object-order*, uma reamostragem é feita durante a composição da imagem intermediária (simulando uma reamostragem bilinear de voxels). Esta reamostragem é feita com facilidade na estrutura de dados de código de corrida, uma vez que, durante a varredura, são percorridas duas linhas adjacentes na imagem intermediária correspondentes a duas linhas na cena. Como resultado da Propriedade 2 da fatoração (seção 3.1.2), os pesos associados à participação de cada voxel no processo de reamostragem são os mesmos para cada voxel da mesma fatia (Figura 3.6). Isto torna o processo de composição dos voxels menos dispendioso do ponto de vista computacional. Outra reamostragem é feita durante a formação da imagem final, durante a transformação *warp*. Como resultado, não aparecem artefatos na imagem final.

As estruturas *shell* e *E-shell* não permitem, durante a projeção, uma reamostragem eficiente dos voxels. A solução adotada para este problema no *rendering* é a replicação da tonalização em 3×3 pixels na imagem final. Com isso podemos dizer que a qualidade do SWR é um pouco melhor que a do SWSR, que por sua vez é melhor que a do SR (Figuras 5.7-5.11).

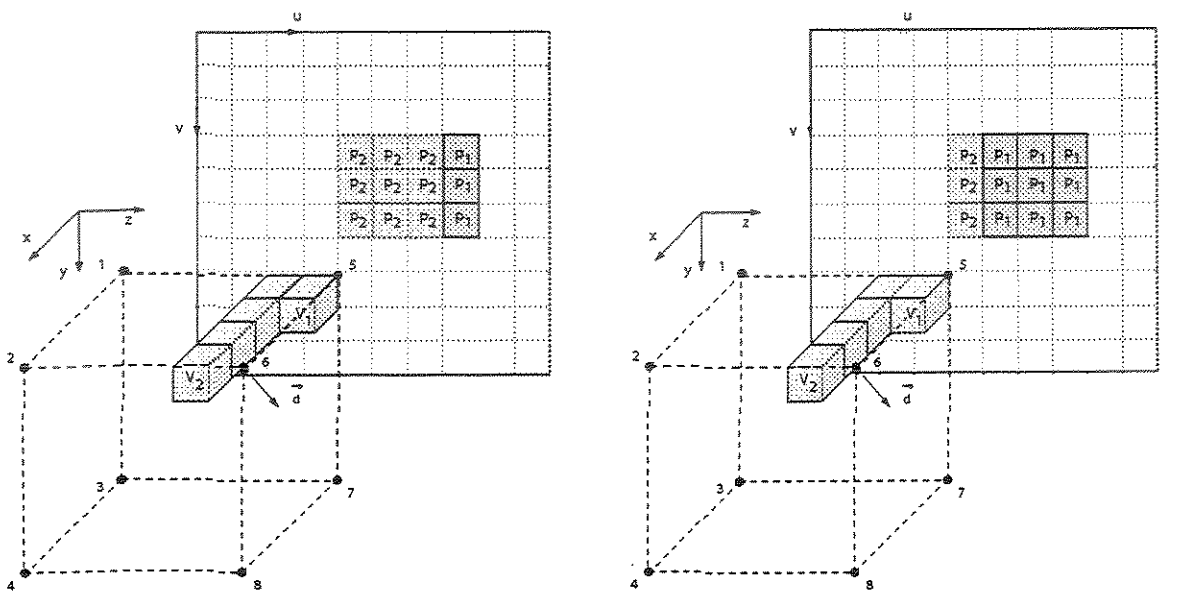
Objeto	Superfície	Volume
O_1	1,09 — 0,27	0,87 — 0,58
O_2	1,69 — 0,38	1,27 — 0,65
O_3	1,20 — 0,26	0,80 — 0,66
O_4	1,10 — 0,25	0,69 — 0,70
O_5	1,89 — 0,42	1,38 — 0,63
O_6	1,38 — 0,26	0,83 — 0,61
O_7	1,39 — 0,25	0,89 — 0,71
O_8	1,20 — 0,25	0,74 — 0,73
O_9	1,33 — 0,25	0,82 — 0,67
O_{10}	1,24 — 0,29	0,75 — 0,62
Média	1,35 — 0,29	0,90 — 0,66

Tabela 5.5: Ganho de velocidade (entrada esquerda) e requisito de memória (entrada direita) do SWSR sobre SWR para superfícies e volumes.



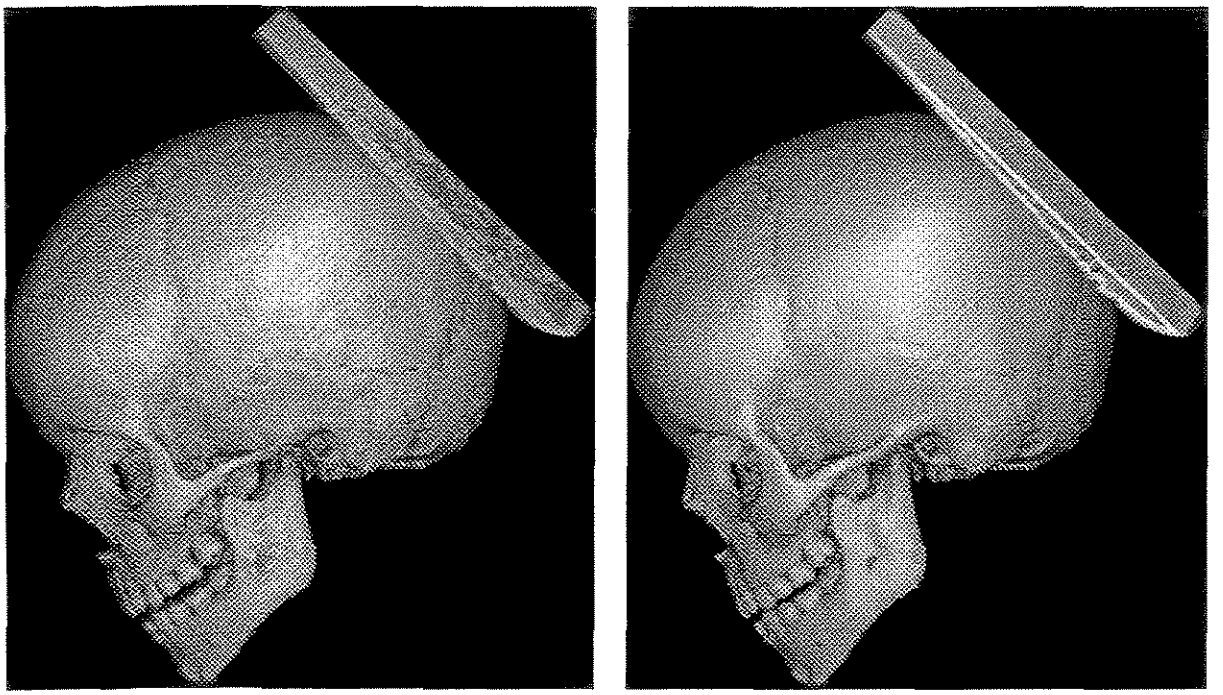
(a) (b)

Figura 5.3: *Shell rendering* de superfície usando *splatting* (a) de um voxel para um pixel e (b) de um voxel para 3×3 pixels.



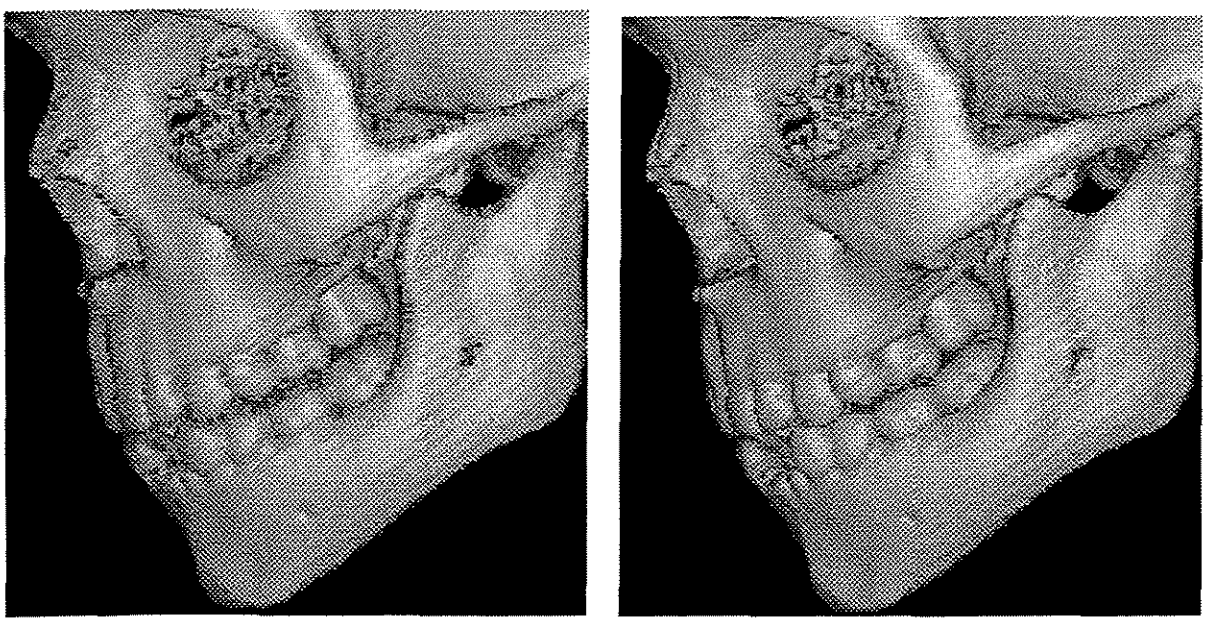
(a) (b)

Figura 5.4: Exemplo onde o observador está no octante 6 e o eixo x é o eixo principal. (a) O acesso *front-to-back* com *splatting* de um para 3×3 pixels na cena faz com que o voxel v_2 pinte os pixels p_2 e o voxel v_1 pinte os pixels p_1. (b) Esta figura mostra que os voxels não são pintados em *front-to-back* se for seguida a ordem de acesso indicada na Tabela 3.2 para *shell rendering*.



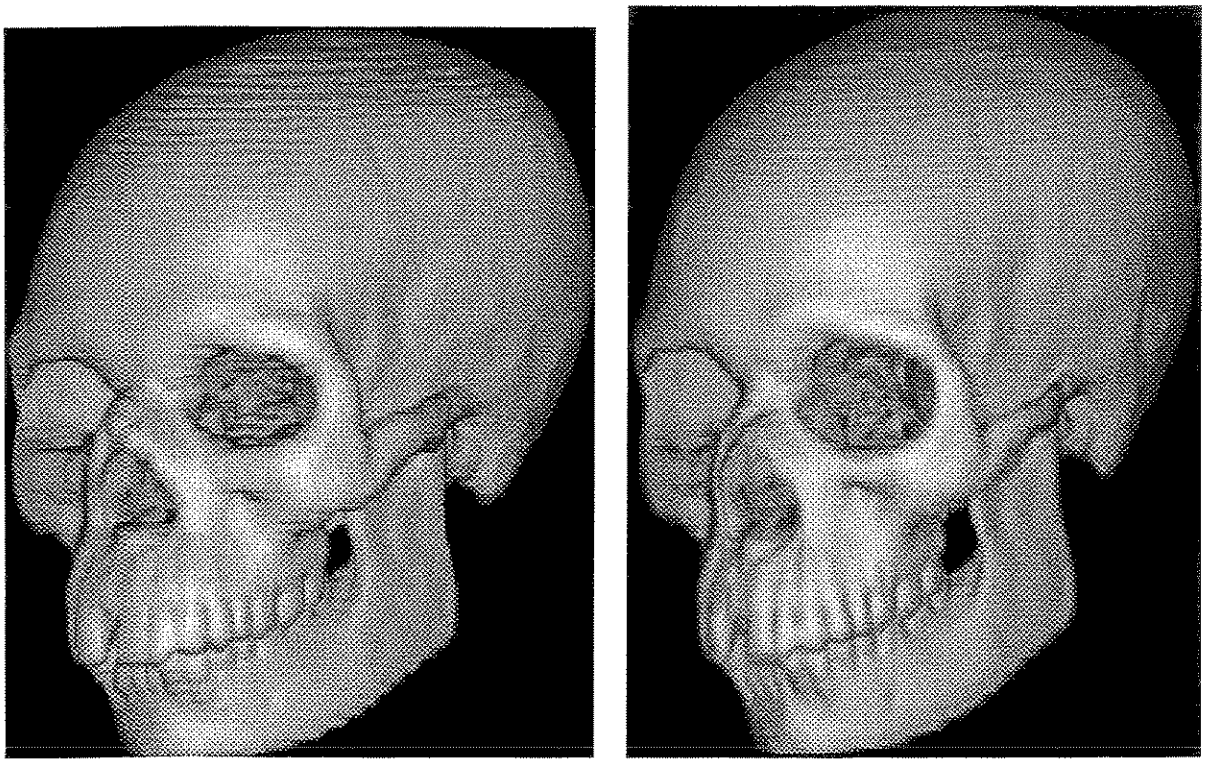
(a) (b)

Figura 5.5: Comparação entre imagens geradas pelo *rendering* de volume com *splatting* de voxels para 3×3 pixels utilizando (a) SR e (b) SWSR.



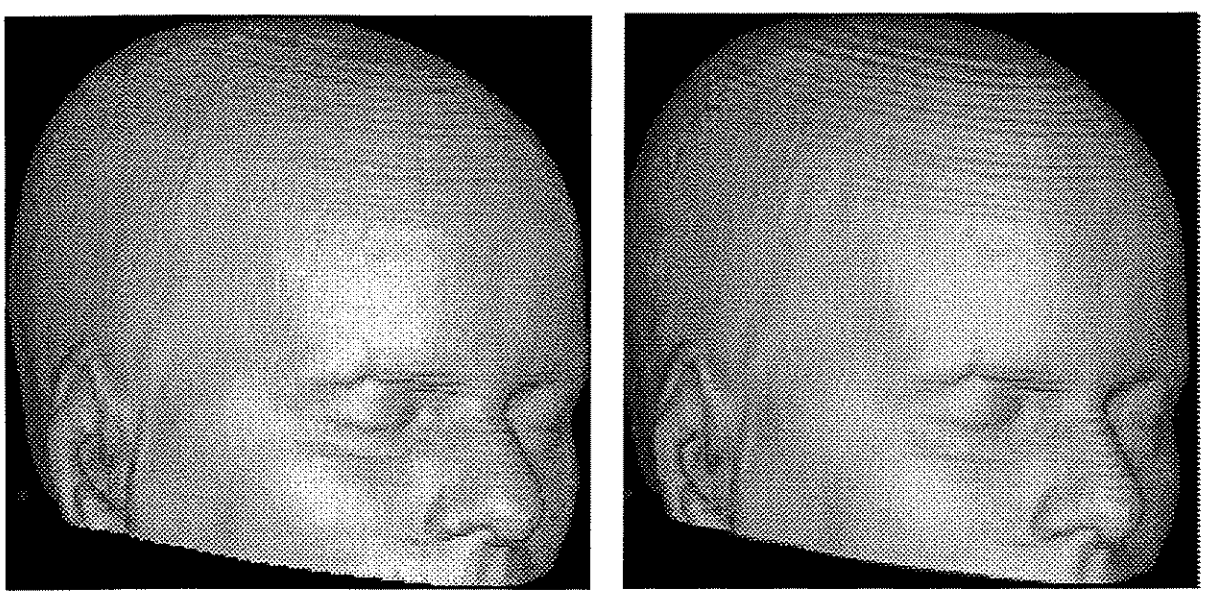
(a) (b)

Figura 5.6: Comparação entre imagens geradas pelo *rendering* de superfície com *splatting* de voxels para 3×3 pixels utilizando (a) SR e (b) SWSR.



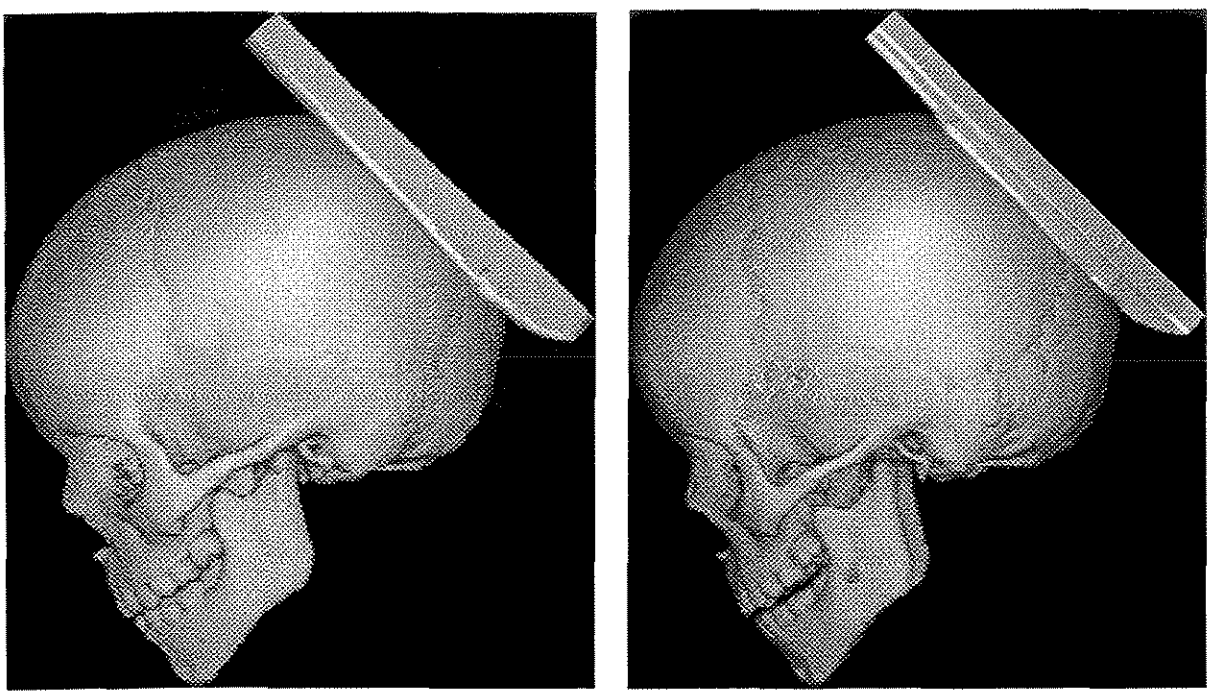
(a) (b)

Figura 5.7: Imagens do objeto O_3 (crânio - CT) geradas por (a) SWSR e (b) SWR.



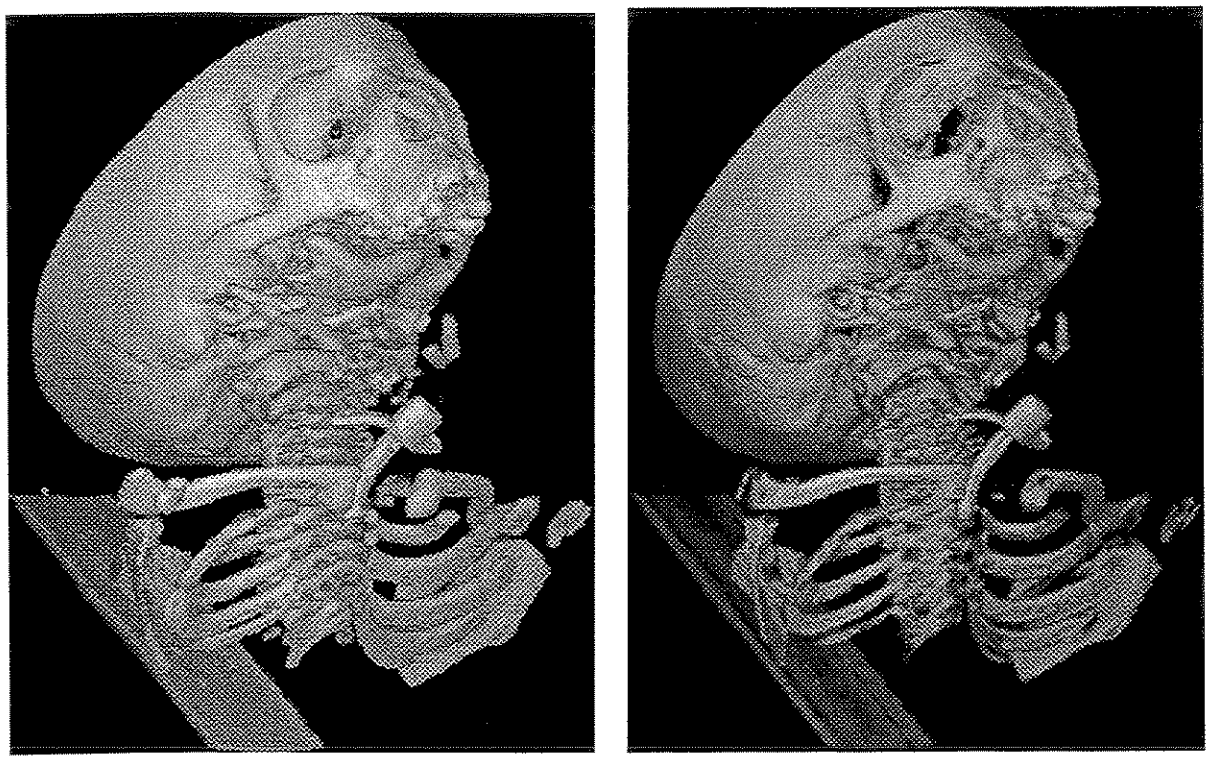
(a) (b)

Figura 5.8: Imagens do objeto O_4 (cabeça - MR) geradas por (a) SWSR e (b) SWR.



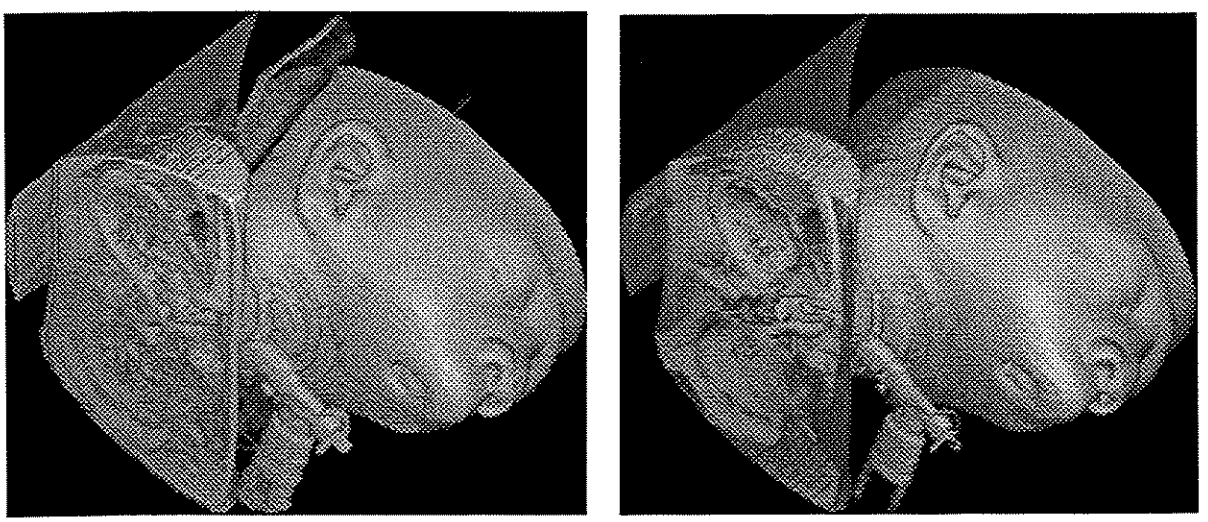
(a) (b)

Figura 5.9: Imagens do objeto O_8 (crânio de criança - CT) geradas por (a) SWSR e (b) SWR.



(a) (b)

Figura 5.10: Imagens do objeto O_9 (cabeça e espinha (ossos) - CT) geradas por (a) SWSR e (b) SWR.



(a) (b)

Figura 5.11: Imagens do objeto O_{10} (cabeça e espinha (pele) - CT) geradas por (a) SWSR e (b) SWR.

Capítulo 6

Conclusão

Este capítulo conclui o trabalho apresentado nesta dissertação, discute as dificuldades encontradas durante a busca dos objetivos lançados no primeiro capítulo e apresenta algumas sugestões para trabalhos futuros.

6.1 Discussão

Nesta dissertação foi proposto um novo método para visualização de volumes, chamado *shell rendering* com fatoração *shear warp* (SWSR). A validação desse método foi feita através de uma análise comparativa entre SWSR e os outros dois métodos relacionados - *shell rendering* (SR) e *shear-warp rendering* (SWR). SWSR é uma variação de SR que permite aceleração pela fatoração da matriz de visualização, uma propriedade matemática também explorada em SWR, sem os altos custos de armazenamento de SWR.

Foram escolhidos 10 objetos diferentes de vários tamanhos, formas e topologias e um computador 1Ghz Pentium-III PC com 512 Mbytes de RAM para os experimentos. Superfícies e volumes de mais de 2.833.000 voxels em tamanho foram criados para testar os métodos em *rendering* de superfície e *rendering* de volume, respectivamente. Os resultados mostram que SWSR é 1.35 vezes mais rápido que SR e SWR para *rendering* de superfície, e para *rendering* de volume, SWSR é 1,52 vezes mais rápido que SR e quase tão rápido quanto SWR. Em *rendering* de superfície, SWSR requer 2 vezes mais memória que SR, mais economiza 71% de espaço de memória quando comparado ao SWR. Em *rendering* de volume, SWSR requer 1,8 vezes mais espaço de memória que SR e economiza 34% do espaço de memória em relação ao SWR. SWSR também gera imagens com qualidade melhor que SR e comparáveis às imagens de alta qualidade geradas pelo SWR. Adicionalmente ao fato que SR tem provado ser um método uma ordem de grandeza mais rápido que métodos poligonais [16], é possível concluir que SWSR é uma ótima opção atualmente para *rendering* de superfície e volume.

6.2 Dificuldades encontradas

Uma das dificuldades encontradas no uso da estrutura *shell* durante o trabalho foi a incorporação de técnicas que fazem a procura de voxels arbitrários do objeto, como *early ray termination* e reamostragem de voxels. O uso destas técnicas foi descartado pois não atendiam aos objetivos deste trabalho.

Outra dificuldade foi a falta de detalhamento dado por Lacroute sobre certos aspectos referentes a SWR, como varredura na cena e a transformação de *warp* através da fatoração da matriz de visualização. Adicionando isto à complexidade da implementação de SWR, optou-se utilizar a biblioteca disponibilizada em [67].

6.3 Sugestões

A continuação do trabalho na área de visualização de volumes pode ser feita através da incorporação de projeção em perspectiva e imageamento estéreo. A estrutura *E-shell* é semelhante à estrutura utilizada em [17], o que facilita a incorporação de projeção em perspectiva ao SWSR.

Como SWSR mostrou ser uma abordagem bastante eficaz, uma aplicação para o trabalho desenvolvido nesta dissertação é o desenvolvimento de um ambiente onde as operações de visualização e manipulação de dados volumétricos possam ser acompanhas com um *rendering* iterativo. Alguns aspectos, como o uso de cores para imageamento estéreo ou rótulos para identificação de objetos podem ser adicionados ao *E-shell* na forma de atributos de visualização.

Apêndice A

Fatoração *shear-warp*

Este apêndice define os sistemas de coordenadas utilizados e faz o detalhamento matemático da fatoração *shear-warp*.

A.1 Sistema de coordenadas padrão

Os sistemas de coordenadas utilizados na fatoração *shear-warp* são ilustrados na Figura A.1: coordenadas da cena, coordenadas padrão, coordenadas intermediárias, e coordenadas da imagem.

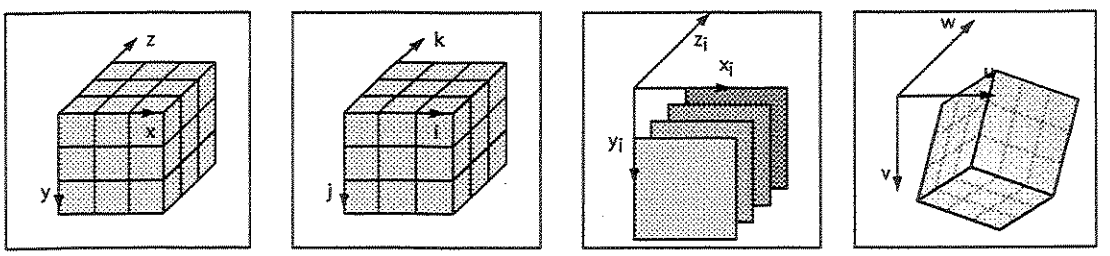


Figura A.1: Sistemas de coordenadas: Cena, Padrão, Intermediário, Imagem (da esquerda para direita).

No sistema de coordenadas da cena a origem corresponde ao voxel (0,0,0) e os eixos são rotulados de x, y e z.

O sistema de coordenadas padrão da cena é formado pela permutação dos eixos do sistema de coordenadas da cena de forma que o principal eixo de visualização torna-se o terceiro eixo de coordenadas. O eixo principal é o mais paralelo à direção de visualização $\vec{d}$. No sistema de coordenadas padrão os eixos são rotulados com i, j e k, onde k é sempre o eixo principal de visualização.

O sistema de coordenadas intermediárias é formado pela transformação de *shear* do sistema de coordenadas padrão. O resultado é formado por diferentes translações no plano ij de cada fatia com coordenada k. O eixos são rotulados x_i, y_i e z_i.

A matriz de *warp* transforma as coordenadas intermediárias em coordenadas da imagem. A origem do sistema de coordenadas da imagem é localizada na esquina esquerda superior da imagem final. Os eixos são rotulados de u, v e w.

A matriz de visualização M_{view} é uma matriz 4×4 que transforma pontos homogêneos do sistema de coordenadas da cena no sistema de coordenadas da imagem.

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = M_{view} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{A.1}$$

O objetivo da fatoração *shear-warp* é representar uma transformação afim de visualização M'_{view} através de deslocamentos M_{shear} do sistema de coordenadas padrão e uma transformação M_{warp} no sistema de coordenadas intermediárias, como a seguir:

$$M'_{view} = M_{warp} \cdot M_{shear} \tag{A.2}$$

A.2 Eixo Principal de Visualização

O eixo principal de visualização pode ser definido como o eixo do espaço objeto que forma o menor ângulo com o vetor da direção de visualização. No espaço da imagem intermediária este vetor é:

$$\vec{d_i} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{A.3}$$

Seja $\vec{d}$ o vetor de visualização transformado para o sistema de coordenadas da cena. Então:

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \tag{A.4}$$

onde m_{ij} são os elementos da matriz de visualização M_{view}. Somente a matriz 3×3 superior esquerda de M_{view} é necessária, sendo denotada por $M_{view,3\times3}$.

Utilizando a Regra de Cramer, a solução para o sistema linear é:

$$d_x = \frac{\begin{vmatrix} 0 & m_{12} & m_{13} \\ 0 & m_{22} & m_{23} \\ -1 & m_{32} & m_{33} \end{vmatrix}}{|M_{view,3\times3}|} = \frac{m_{22}m_{13} - m_{12}m_{23}}{|M_{view,3\times3}|}$$

$$d_y = \frac{\begin{vmatrix} m_{11} & 0 & m_{13} \\ m_{21} & 0 & m_{23} \\ m_{31} & -1 & m_{33} \end{vmatrix}}{|M_{view,3\times3}|} = \frac{m_{11}m_{23} - m_{21}m_{13}}{|M_{view,3\times3}|}$$

$$d_z = \frac{\begin{vmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ m_{31} & m_{32} & -1 \end{vmatrix}}{|M_{view,3\times3}|} = \frac{m_{21}m_{12} - m_{11}m_{22}}{|M_{view,3\times3}|}$$

Como o denominador é o mesmo para os três componentes de $\vec{d}$, ele pode ser eliminado, resultando:

$$\vec{d} = \begin{bmatrix} m_{22}m_{13} - m_{12}m_{23} \\ m_{11}m_{23} - m_{21}m_{13} \\ m_{21}m_{12} - m_{11}m_{22} \end{bmatrix} \tag{A.5}$$

O cosseno do ângulo entre a direção de visualização e cada eixo do sistema de coordenadas da cena é proporcional ao produto interno de $\vec{d}$ com cada um dos vetores unitários dos eixos principais deste sistema. O maior produto interno corresponde ao menor ângulo. O eixo principal pode ser calculado pela Equação A.6:

$$e_p = \max(|d_x|, |d_y|, |d_z|) \tag{A.6}$$

Se $e_p = |d_x|$ então o eixo principal de visualização é o eixo x. Se $e_p = |d_y|$ então o eixo principal de visualização é o eixo y. Caso contrário, o eixo principal de visualização é o eixo z.

A.3 Coordenadas padrão

A fatoração *shear-warp* é realizada nas fatias de voxels que são perpendiculares ao eixo principal de visualização. A transformação do sistema de coordenadas da cena para o

sistema de coordenadas padrão é feita através da permutação dos eixos x, y e z. A fatoração da matriz visualização no sistema de coordenadas da cena (M_{view}) é então representada pelas matrizes de *shear* (M_{shear}), *warp* (M_{warp}) e também por uma matriz de permutação (M_P):

$$M_{view} = M_{warp} M_{shear} M_P \tag{A.7}$$

Se o eixo principal de visualização for o eixo x então a matriz de permutação é:

$$M_P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.8}$$

Se o eixo principal de visualização for o eixo y então a matriz de permutação é:

$$M_P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.9}$$

Se o eixo principal de visualização for o eixo z então a matriz de permutação é a matriz identidade:

$$M_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.10}$$

A transformação de coordenadas da cena para coordenadas padrão é:

$$\begin{bmatrix} i \\ j \\ k \\ 0 \end{bmatrix} = M_P \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} \tag{A.11}$$

Seja M'_{view} a matriz de visualização permutada:

$$M'_{view} = M_{view} (M_P)^{-1} \tag{A.12}$$

Esta matriz transforma pontos do sistema de coordenadas da cena no sistema de coordenadas padrão.

A.4 Transformação *Shear*

O próximo passo é fatorar a matriz de visualização permutada M'_{view} nas direções de i e j. A fatoração deve satisfazer a seguinte condição: depois da transformação *shear* a direção de visualização deve ser perpendicular ao plano ij. A matriz M'_{view} pode ser escrita através de seus elementos m'_{ij}:

$$M'_{shear} = \begin{bmatrix} m'_{11} & m'_{12} & m'_{13} & m'_{14} \\ m'_{21} & m'_{22} & m'_{23} & m'_{24} \\ m'_{31} & m'_{32} & m'_{33} & m'_{34} \\ m'_{41} & m'_{42} & m'_{43} & m'_{44} \end{bmatrix} \tag{A.13}$$

No sistema de coordenadas padrão o vetor da direção de visualização $\vec{d}'$ é:

$$\vec{d}' = M_P \cdot \vec{d} = \begin{bmatrix} m'_{22}m'_{13} - m'_{12}m'_{23} \\ m'_{11}m'_{23} - m'_{21}m'_{13} \\ m'_{21}m'_{12} - m'_{11}m'_{22} \end{bmatrix} \tag{A.14}$$

A projeção do vetor de direção no plano ij tem uma inclinação d'_i/d'_k (Figura A.2). O deslocamento necessário na direção de i para tornar a direção de visualização perpendicular ao plano ij é o negativo desta inclinação. O mesmo argumento serve para o deslocamento na direção de j. Os coeficientes de *shear* são:

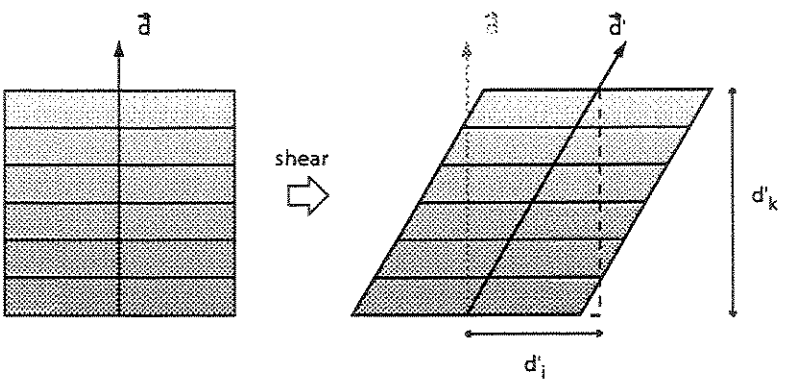


Figura A.2: Cálculo dos coeficientes de *shear* s_i e s_j.

$$s_i = -\frac{d'_i}{d'_k} = \frac{m'_{12}m'_{23} - m'_{22}m'_{13}}{m'_{21}m'_{12} - m'_{11}m'_{22}} \tag{A.15}$$

$$s_j = -\frac{d'_j}{d'_k} = \frac{m'_{21}m'_{13} - m'_{11}m'_{23}}{m'_{21}m'_{12} - m'_{11}m'_{22}} \tag{A.16}$$

A.5 Imagem intermediária

A transformação *shear*, seguida da projeção e composição das fatias, forma uma imagem intermediária (Figura A.4a). No entanto, nos casos Figura A.3) em que os voxels assumem coordenadas negativas após o deslocamento das fatias, é necessário adicionar uma translação para tornar suas coordenadas positivas. A transformação *shear*, que produz um sistema de coordenadas intermediárias, é então definida por:

$$M_{shear} = \begin{bmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_i & t_i \\ 0 & 1 & s_j & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.17}$$

A Figura A.3 ilustra os quatro possíveis casos e as formulas correspondentes para o cálculo de cada translação necessária. Em cada caso é identificado pelos sinais dos coeficientes de *shear*.

Uma informação extra é necessária para calcular a imagem intermediária: a ordem de empilhamento das fatias na imagem intermediária. As fatias de voxels são ordenadas pela sua coordenada k no sistema de coordenadas padrão. Uma varredura *front-to-back* das fatias corresponde percorrer as fatias em ordem crescente ou decrescente, dependendo da direção de visualização. A ordem de empilhamento é encontrada através da examinação do componente d'_k do vetor de direção de visualização. Se d'_k é positivo então a fatia em no plano $k = 0$ é a fatia da frente, caso contrário, a fatia no plano $k = k_{\max}$ é a fatia de frente.

A.6 Transformação *Warp*

A fatoração *shear-warp* pode ser obtida pela matrizes M'_{view} e M_{shear}:

$$M_{warp} M_{shear} = M'_{view} \tag{A.18}$$

$$M_{warp} = M'_{view} M^{-1}_{shear} \tag{A.19}$$

onde M^{-1}_{shear} é:

$$M^{-1}_{shear} = \begin{bmatrix} 1 & 0 & -s_i & -t_i \\ 0 & 1 & -s_j & -t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.20}$$

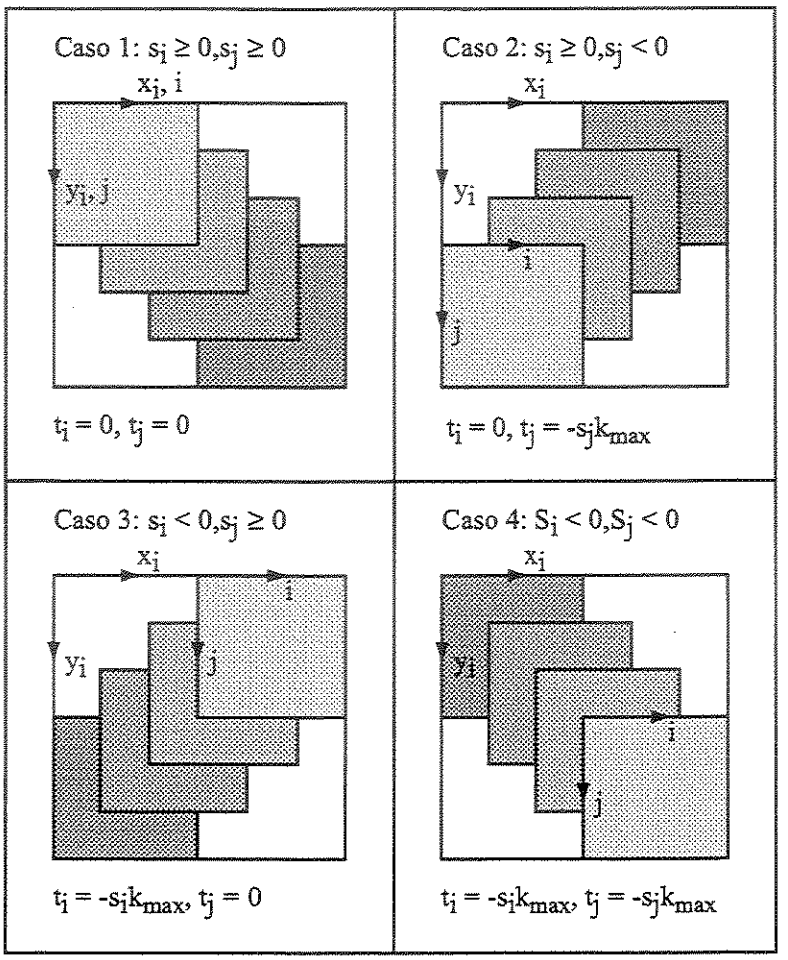


Figura A.3: Os quatro possíveis casos e as fórmulas correspondentes para o cálculo de cada translação t_i e t_j em relação ao fatores de *shear* e a posição $k_{\max}$ da última fatia.

As três possibilidades de M'_{view} no sistema de coordenadas padrão dependem do eixo principal de visualização. Se o eixo principal de visualização for o eixo x:

$$M'_{view} = \begin{bmatrix} 0 & -sen(\phi) & cos(\phi) & -t_z \\ cos(\theta) & sen(\theta)cos(\phi) & sen(\theta)sen(\phi) & -t_x \\ -sen(\theta) & cos(\theta)cos(\phi) & cos(\theta)sen(\phi) & -t_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.21)$$

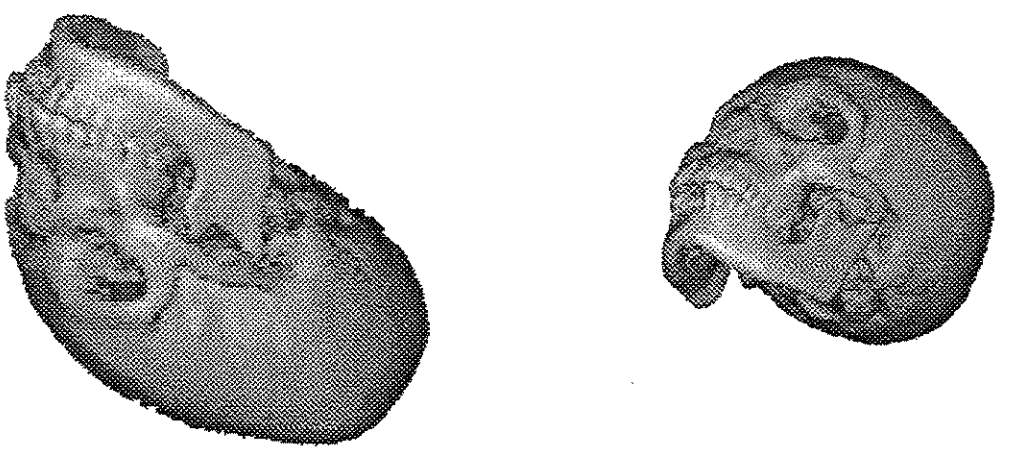
Se o eixo principal de visualização for o eixo y:

$$M'_{view} = \begin{bmatrix} -sen(\phi) & cos(\phi) & 0 & -t_y \\ sen(\theta)cos(\phi) & sen(\theta)sen(\phi) & cos(\theta) & -t_z \\ cos(\theta)cos(\phi) & cos(\theta)sen(\phi) & -sen(\theta) & -t_x \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.22)$$

Se o eixo principal de visualização for o eixo z, M_{view} e M'_{view} são iguais:

$$M'_{view} = \begin{bmatrix} cos(\phi) & 0 & -sen(\phi) & -t_x \\ sen(\theta)sen(\phi) & cos(\theta) & sen(\theta)cos(\phi) & -t_y \\ cos(\theta)sen(\phi) & -sen(\theta) & cos(\theta)cos(\phi) & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.23)$$

Na equação A.19, a matriz de *warp* foi construída de forma que, depois da transformação *shear* na cena, a projeção e composição das fatias (Figura A.4a) forma uma imagem intermediária. Assim é possível aplicar a transformação *warp* em 2D (Figura A.4b):



(a) (b)

Figura A.4: A transformação de *shear* projeta a cena na imagem intermediária em (a) e a transformação de *warp* 2D transforma a imagem intermediária na imagem final em (b).

A matriz de *warp*, que transforma a imagem intermediária (Figura A.4a) nas coordenadas da imagem (Figura A.4b):

$$M_{warp} = \begin{bmatrix} m'_{11} & m'_{12} & m'_{13} - s_i m'_{11} - s_j m'_{12} & m'_{14} - t_i m'_{11} - t_j m'_{12} \\ m'_{21} & m'_{22} & m'_{23} - s_i m'_{21} - s_j m'_{22} & m'_{24} - t_i m'_{21} - t_j m'_{22} \\ m'_{31} & m'_{32} & m'_{33} - s_i m'_{31} - s_j m'_{32} & m'_{34} - t_i m'_{31} - t_j m'_{32} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.24}$$

Como é possível aplicar a matriz de *warp* em 2D, a terceira linha e a terceira coluna podem ser removidas de M_{warp}.

$$M_{warp2D} = \begin{bmatrix} m'_{11} & m'_{12} & m'_{14} - t_i m'_{11} - t_j m'_{12} \\ m'_{21} & m'_{22} & m'_{24} - t_i m'_{21} - t_j m'_{22} \\ 0 & 0 & 1 \end{bmatrix} \tag{A.25}$$

Esta é a matriz de *warp* 3×3 que transforma a imagem intermediária 2D na imagem final 2D:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{warp2D} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{A.26}$$

Referências Bibliográficas

[1] J.K. Udupa. Three-dimensional imaging: Principles and approaches. Technical Report MIPG254, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Feb 1999.

[2] R. Gonzalez e R.R. Woods. *Digital Image Processing.* Addison-Wesley, New York, 1993.

[3] W.E. Lorensen e Cline H.E. Marching cubes: A high resolution 3D surface construction algorithm. *Computer & Graphics*, páginas 163–169, Jul 1987.

[4] M.J. Durst. Letters: Additional reference to marching cubes. *Computer Graphics (In Proc. of ACM SIGGRAPH)*, 22(2):72–73, Apr 1988.

[5] I. Gargantini e H. Atkinson. Multiple-seed 3D connectivity filling for innacurate borders. *CVGIP: Graphical Models and Image Processing*, 53(6):563–573, 1991.

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

[7] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, páginas 2–7, 1990.

[8] K.R. Subramanian e D.S. Fussell. Applying space subdivision techniques to volume rendering. Em *Proceedings of Visualization'90*, páginas 150–159, San Francisco, CA, 1990.

[9] P. Lacroute e M. Levoy. Fast volume rendering using a shear-warp factorization of viewing transformation. *Computer Graphics*, páginas 451–458, 1994.

[10] M. Bentum. *Interactive visualization of volume data.* PhD thesis, Dept. of Electrical Engeneering, University of Twente, Enschede, The Netherlands, Jun 1996.

[11] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, páginas 245–261, 1990.

[12] J. Danskin e P. Hanrahan. Fast algorithms for volume ray tracing. Em *Proceedings of the 1992 Workshop on Volume Rendering*, volume 19, páginas 91–98, 1992.

[13] P. Lacroute. *Fast volume rendering using a shear-warp factorization of the viewing transformation.* PhD thesis, Dept. of Electrical Engineering and Computer Science, Stanford University, Stanford, CA, Sep 1995.

[14] J.K. Zuiderveld, A.H.J. Koning, e M.A. Viergever. Acceleration of ray-casting using 3d distance transforms. Em *Proceedings of Visualization in Biomedical Computing*, páginas 324–335, Chapel Hill, North Caroline, Oct 1992.

[15] G.G. Cameron e P.E. Undrill. Rendering volumetric medical image data on a SIMS-architecture computer. Em *Proceedings of the Third Eurographics Workshop on Rendering*, páginas 135–145, Bristol, UK, May 1992.

[16] G.J. Grevera, J.K. Udupa, e D. Odhner. An order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware. *IEEE Transactions on Visualization and Computer Graphics*, 6(4):335–345, Oct-Dec 2000.

[17] G.P. Carnielli, A.X. Falcao, e J.K. Udupa. Fast digital perspective shell rendering. Em *XII Brazilian Symposium on Computer Graphics and Image Processing*, páginas 105–111, Campinas, SP, Brazil, Oct 1999.

[18] G.J. Grevera, J.K. Udupa, e D. Odhner. T-shell rendering. Em *Proceedings of SPIE on Medical Imaging: Visualization, Display, and Image-Guided Procedures*, volume 4319, páginas 413–425, Feb 2001.

[19] P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, Sep 1996.

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

[21] B.H. Kim, J. Seo, e Y.G. Shin. Binary volume rendering using slice-based binary shell. *The Visual Computer*, 17(4):243–257, 2001.

[22] T.Y. Kim e Y.G. Shin. Fast volume rendering with interactive classification. *Computer & Graphics*, 25(5):819–831, Oct 2001.

[23] D.M. Jiang e J.P. Singh. Improving parallel shear-warp volume rendering on shared address space multiprocessors. *ACM SIGPLAN NOTICES*, 32(7):252–263, Jul 1997.

[24] W.S. Edwards, C. Deforge, e Y. Kim. Interactive three-dimensional ultrasound using a programmable multimedia processor. *International Journal of Imaging Systems and Technology*, 9(6):442–454, 1998.

[25] W.L. Cai e G.B. Sakas. Maximum intensity projection using splatting in sheared object space. *Computer Graphics Forum*, 17(3):113–124, 1998.

[26] H.S. Kang, B.H. Kim, J.W. Ryu, S.H. Hong, H.W. Chung, S.Y. Cho, Y.H. Kim, S.I. Hwang, D.K. Jeong, e Y.G. Shin. The visible man: Three-dimensional interactive musculoskeletal anatomic atlas of the lower extremity. *Radiographics*, 20(1):279–286, Jan-Feb 2000.

[27] A.X. Falcao L. M. Rocha e J.K. Udupa. Shear-warp shell rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2002. Submetido.

[28] A.X. Falcao, L.M. Rocha, e J.K. Udupa. A comparative analysis of shell rendering and shear-warp rendering. Em *Proceedings of SPIE on Medical Imaging 2002*, volume 4681, San Diego, CA, Feb 2002.

[29] L. M. Rocha e A. X. Falcao. E-shell rendering. Em *Proceedings of the XV Brazilian Symposium on Computer Graphics and Image Processing*, oct 2002.

[30] J.D. Foley, S.K. Feiner A. van Dam, e J.F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, second ed. edition, 1990.

[31] J. Serra. Morphological filtering: An overview. *Signal Processing*, páginas 3–11, 1994.

[32] S. Raya. Low-level segmentation of 3-d magnetic resonance brain images. *IEEE Transactions on Medical Imaging*, páginas 327–337, 1990.

[33] L. Gong e C. Kulikowski. Composition of image analysis processes through object-centered hierarchical planning. *IEEE Transactions on Medical Imaging*, 53(6):563–573, 1991.

[34] D. Collins e T. Peters. Model-based segmentation of individual brain structures from mri data. Em *SPIE on Medical Imaging*, páginas 71–83, 1992.

[35] M. Sonka, S. Tadikonda, e S. Collins. Knowledge-based interpretation of mr brain images. Em *IEEE Transactions on Medical Imaging*, páginas 443–452, 1996.

[36] R. Bajecsy e S. Kovacic. Multiresolution elastic matching. *Computer Vision, Graphics and Image Processing*, páginas 1–21, 1989.

[37] J. Gee, C. Barillot, L. Le Briquer, D. Haynor, e R. Bajecsy. Matching structural images of the human brain using statistical and geometrical image features. Em *SPIE on Medical Imaging*, páginas 191–204, 1994.

[38] J. Talairach, G. Szikla, P. Tournoux, A. Prossalentis, M. Bordas-Ferre, L. Covello, M. Jacob, A. Mempel, P. Buser, e J. Bancaud. *Co-Planar Stereotaxis Atlas of the Human Brain: 3-Dimensional Proportional System - An Approach to Cerebral Imaging.* New York: Thieme Med., New York, 1998.

[39] J. Gee, M. Reivich, e R. Bajesy. Elastically deforming 3d atlas to match anatomical brain. *Journal of Computer Assited Tomography*, páginas 225–236, 1993.

[40] G. Christensen, R. Rabbitt, e M. Miller. 3-d brain mapping using a deformable neuroanatomy. *Physics in Medicine Biology*, 39:609–518, 1994.

[41] M. Kamber, R. Shinghal, D. Collins, G. Francis, e A. Evans. Model-based 3d segmentation of multiple sclerosis lesions in magnetic resonance brain images. Em *IEEE Transactions on Medical Imaging*, volume 14, páginas 442–453, 1995.

[42] D. Collins e T. Peters. A dual probabilistic classifier for three-dimensional neuroimaging from mri data. Em *SPIE on Medical Imaging*, páginas 373–384, 1994.

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

[44] J. Udupa, S. Srihari, e G. Herman. Boundary detection in multidimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:41–50, 1982.

[45] J. Canny. A computacional approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

[46] H. Liu. Two- and three-dimensional boundary detection. *Computer Graphics and Image Processing*, 8:123–134, 1977.

[47] S.P. Raya e J.K. Udupa. Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging*, páginas 32–42, 1990.

[48] G.T. Herman, J. Zheng, e C.A. Bucholtz. Shape-based interpolation. *IEEE Computer Graphics and Applications*, páginas 69–79, 1992.

[49] W.E. Higgins, C. Morice, e E.L. Ritman. Shape-based interpolation technique for three-dimensional images. *Proceedings of IEEE 1990 International Conference on Acoustics, Speech and Signal Processing*, páginas 1841–1844, April 1990.

[50] R.A. Lotufo, G.T. Herman, e J.K. Udupa. Combining shape-based interpolation and gray-level interpolations. Em *SPIE Proceedings in Visualization in Biomedical Computing*, volume 1808, páginas 289–298, 1992.

[51] R.A. Drebin, L. Carpenter, e P. Hanrahan. Volume rendering. *Computer Graphics*, páginas 65–74, Aug 1988.

[52] A.R. Smith. Volume graphics and volume visualization: A tutorial. Technical Memo 176, Pixar Inc., California, 1987.

[53] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, páginas 29–37, May 1988.

[54] G. Frieder, D. Gordon, e R. A. Reynolds. Back-to-front display of voxels based objects. *IEEE Computer Graphics and Applications*, páginas 52–60, Jan 1985.

[55] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics*, páginas 51–58, Aug 1988.

[56] C. Upson e M. Keeler. V-BUFFER: Visible volume rendering. *Computer Graphics*, 22:59–64, 1988.

[57] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24:367–376, 1990.

[58] J.K. Udupa. 3D visualization of images. Technical Report MIPG196, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Jun 1993.

[59] R. A. Reynolds. *Fast methods for 3D display of medical objects.* PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, May 1985.

[60] D.J. Meagher. Efficient synthetic image generation of arbitrary 3-d objects. Em *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing*, páginas 473–478, 1982.

[61] R. Reynolds, D. Gordon, e L. Chen. A dynamic screen technique for shaded graphics display of slice-represented objects. *Computer Vision, Graphics and Image Processing*, páginas 275–298, 1987.

[62] A.X. Falcao. Visualização de volumes aplicada à área médica. Tese de mestrado, FEEC-UNICAMP, Feb 1993.

[63] H. Gouraud. Continous shading of curved surfaces. *IEEE Transaction of Computers*, páginas 623–629, 1971.

[64] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, páginas 311–317, Jun 1975.

[65] A.V. Aho, J.E. Hopcroft, e J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, MA, 1974.

[66] T.H. Leiseson, C.E., e R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[67] P. Lacroute. The volpack volume rendering library. Disponível em: <http://graphics.stanford.edu/software/volpack>. Acesso em: 30 de junho de 2002.