# Um Método Rápido para Rendering de Volumes com Perspectiva Digital

*Giulliano Paes Carnielli*

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

# Um Método Rápido para Rendering de Volumes com Perspectiva Digital

Giulliano Paes Carnielli

20 de dezembro de 1999

**Banca Examinadora:**

- Prof. Dr. Alexandre Xavier Falcão
  IC - Universidade Estadual de Campinas (Orientador)
- Prof. Dr. Léo Pini Magalhães
  FEEC - Universidade Estadual de Campinas
- Prof. Dr. Jorge Stolfi
  IC - Universidade Estadual de Campinas
- Prof. Dr. Neucimar Jerônimo Leite (Suplente)
  IC - Universidade Estadual de Campinas

## FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Carnielli, Giulliano Paes

C217m Um método rápido para rendering de volumes com perspectiva digital / Giulliano Paes Carnielli -- Campinas, [S.P. :s.n.], 1999.

Orientador : Alexandre Xavier Falcão

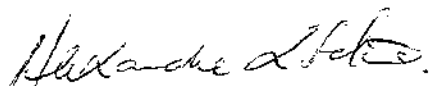
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computação gráfica. 2. Visualização. I. Falcão, Alexandre Xavier. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

# Um Método Rápido para Rendering de Volumes com Perspectiva Digital

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Giulliano Paes Carnielli e aprovada pela Banca Examinadora.

Campinas, 20 de dezembro de 1999.



Prof. Dr. Alexandre Xavier Falcão
IC - 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.

# TERMO DE APROVAÇÃO

Tese defendida e aprovada em 20 de dezembro de 1999, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Léo Pini Magalhães
FEEC-UNICAMP

Prof. Dr. Jorge Stolfi
IC-UNICAMP

Prof. Dr. Alexandre Xavier Falcão
IC-UNICAMP

# Agradecimentos

Gostaria de expressar meus agradecimentos a todos que, de uma forma ou de outra, contribuiram para a conclusão deste trabalho, que significou uma grande conquista em minha vida:

A Deus, pois *"todas as coisas foram feitas por Ele, e sem Ele nada do que foi feito se fez."* (Jo1:3)

Meu Pai, por ter se revelado o meu grande amigo, e por ter me proporcionado chegar onde estou.

Minha Mãe, pela sua dedicação e seu amor.

Minha namorada Juliana "Tuca" Ferreira, por ter me oferecido a sua compreensão.

Meus amigos de Tatuí, porque posso chamá-los *irmãos*.

Os amigos da Comunidade Arroba e de São Carlos, pelo tempo que passamos juntos.

Professores e Funcionários do IC.

A Acassil José Vieira de Camargo Jr., em nome do Colégio Objetivo de Tatuí, pois foi onde tudo começou.

Meu orientador Alexandre Xavier Falcão, por me mostrar o caminho.

FAPESP, CAPES e CNPq, pelo apoio financeiro.

# Resumo

Métodos de visualização de volumes requerem altos custos de armazenamento e tempo computacional. Para torná-los viáveis, pesquisadores têm proposto soluções baseadas em *hardware* e em *software* para minimizar tais custos. Na maioria dessas soluções, o *rendering* é realizado com projeção ortogonal. Entretanto, em muitas aplicações, tais como endoscopia virtual, a projeção perspectiva torna-se necessária.

Este trabalho descreve um estudo sobre técnicas de processamento de imagens e computação gráfica para visualização de volumes aplicada à medicina. E estende um destes métodos de visualização, chamado *shell rendering* , para prover projeção perspectiva de forma eficiente. O método *shell rendering* é descrito e os problemas encontrados na sua extensão para perspectiva são discutidos. A solução adotada propõe uma modificação da estrudura de dados original e um algoritmo rápido de projeção, introduzindo o conceito de perspectiva digital. O método desenvolvido foi denominado *shell rendering* com perspectiva digital.

A idéia principal em perspectiva digital é que as estruturas devem ser projetadas com tamanho inversamente proporcional à distância ao observador e que esta distância pode ser discretizada. Neste trabalho, assume-se o observador no exterior das estruturas. Os métodos *shell rendering* e *shell rendering* com perspectiva digital foram implementados, testados e comparados sob dois aspectos: qualidade de imagem e tempo computacional. O *shell rendering* com perspectiva digital mostrou produzir a mesma qualidade de imagem que o método original, com tempo de *rendering* apenas cerca de 1.6 vezes maior que o *rendering* com projeção ortogonal.

# *Abstract*

Volume visualization methods require extensive storage and time-intensive computations. To make them viable, researches have proposed hardware-based and software-based solutions. In most of these solutions, the rendering is based on orthographic parallel projection. However, in many applications such as virtual endoscopy, perspective projection becomes necessary.

This work reports a study on image processing and computer graphics techniques for volume visualization applied to medicine. And extends one of these methods, called shell rendering, to provide perspective projection in an efficient way. The method shell rendering is described and the problems related to its extension to perspective projection are discussed. The adopted solution proposes a modification in the original data structure and a fast projection algorithm, introducing the concept of digital perspective. The new method was denominated digital perspective shell rendering.

The main idea in digital perspective is that each object should be rendered with size inversely proportional to its distance to the observer and this distance can be discretized. In this work, the observer is assumed to be always outside the object. The shell rendering and digital perspective shell rendering methods were implemented, tested and compared under two aspects: image quality and computational time. The digital perspective shell rendering has shown to produce the same image quality as the original method and computational time only about 1.6 times greater than with orthographic parallel projection.

# Sumário

# Lista de Tabelas

# Lista de Figuras

# Capítulo 1

# Introdução

Métodos de visualização de volumes são bastante usados em diversas áreas da ciência e engenharia (e.g. medicina, engenharia do petróleo, meteorologia, geologia, biologia, astronomia) com o objetivo de analisar e compreender certos fenômenos físicos [1, 2, 3]. Em medicina, por exemplo, estes fenômenos estão relacionados com a anatomia e a função de estruturas internas ao corpo humano. A visualização em tempo real destas informações é imprescindível em muitas aplicações. Este fato tem levado muitos pesquisadores a investigar métodos rápidos de visualização de volumes. Este trabalho descreve um estudo realizado sobre uma classe destes métodos que resultou em um novo método, denominado *Shell Rendering* em Perspectiva Digital [4]. Muito embora o método desenvolvido possa ser utilizado para visualização de outros tipos de dados volumétricos, o enfoque deste trabalho foi o problema da visualização de volumes aplicada à medicina.

## 1.1 Visualização de Volumes na Medicina

Em medicina, a visualização tem como principal objetivo facilitar o entendimento de estruturas complexas internas ao corpo humano. Ela é utilizada em conjunto com duas outras operações: manipulação e análise. A manipulação envolve rotação, escalamento (*scaling*), cortes, e outras técnicas usadas para alterar a forma e a função das estruturas. A análise envolve a avaliação qualitativa e quantitativa das estruturas, levando em conta medidas de distância, área, volume, e localização espacial. Quando combinadas, a visualização, a manipulação e a análise constituem a área de `Computação de Imagens Médicas` [5]. Suas aplicações compreendem o diagnóstico por imagens, o planejamento de cirurgias [6] e radioterapias, a educação médica, o estudo de diversos fenômenos que alteram a anatomia e/ou função de órgãos do corpo humano, e a análise de movimento de articulações [7, 8].

Dados volumétricos são obtidos por amostragem e quantização de certas proprieda-

des físicas em uma região do corpo do paciente e representados como uma seqüência de imagens digitais. Estes dados podem ser adquiridos via tomografias de raios-X (CT), de emissão de pósitrons (PET), de emissão de fótons (SPECT), ressonância magnética (MRI), ultrasom, ou, no caso de estruturas biológicas, através de uma câmera especial (e.g. câmera confocal) acoplada a um microscópio. Cada uma destas técnicas oferece um tipo diferente de informação sobre as estruturas contidas nos dados volumétricos. Na tomografia de raios-X, por exemplo, o brilho das imagens resultantes é proporcional à densidade do material na região em que foi feito o exame. Esta modalidade é preferida para análise da anatomia de tecidos ósseos, pois estes possuem alta densidade gerando um bom contraste em relação aos tecidos vizinhos.

Os problemas básicos em visualização de volumes são a classificação e o *rendering* de estruturas tridimensionais (3D). A classificação é o processo de especificar quais estruturas devem ser visualizadas. O *rendering* consiste da geração de imagens destas estruturas a partir de modelos geométricos. Além das dificuldades inerentes ao processo de classificação, a grande quantidade de informação que pode ser gerada para *rendering* requer bastante espaço em memória e tempo de processamento, comprometendo a interatividade com o usuário durante a visualização. Em computação de imagens médicas, a interatividade é requisito fundamental para a viabilidade de muitas aplicações clínicas, tais como endoscopia virtual e simulações cirúrgicas. Para conseguir visualização em tempo real, alguns pesquisadores têm proposto *hardwares* dedicados [9] e algoritmos que procuram um meio-termo entre qualidade de imagem e tempo de processamento [10]. Enquanto o desenvolvimento de *hardwares* dedicados é importante para avançar o conhecimento em computação, abordar o problema da interatividade através de algoritmos eficientes pode ser preferível do ponto de vista prático de disponibilidade e portabilidade das implementações. Neste sentido, todo esforço feito neste trabalho foi visando um método rápido de visualização que fosse independente da plataforma de implementação.

## 1.2 Visão Geral

Métodos de visualização de volumes podem ser classificados em dois grupos: *rendering* de `volumes` e *rendering* de `superfícies`. Em *rendering* de volumes [11, 12, 13, 14, 15, 16], o conjunto de dados volumétricos é considerado semi-transparente, no qual estruturas têm associado valores de opacidade proporcionais ao grau de interesse na visualização. Estas opacidades são escolhidas durante o processo de classificação. Em *rendering* de superfícies [17, 18, 19, 20, 21], a classificação é feita no sentido de "extrair"as estruturas de interesse. Subsequentemente, um modelo geométrico da superfície destas estruturas é criado para visualização. Se considerarmos a classificação como qualquer operação, *fuzzy* ou binária, de associar opacidades proporcionais ao grau de interesse na visualização, técnicas

de visualização podem ser vistas como especializações de um modelo de teoria de transporte sobre a propagação da luz em materiais. Em *rendering* de superfícies, os materiais são considerados ou completamente opacos ou completamente transparentes, enquanto em *rendering* de volumes eles podem ter diferentes graus de opacidade/transparência. Considerando a extensa literatura em métodos de visualização de volumes, e que o *rendering* de superfícies pode ser obtido como um caso particular de *rendering* de volumes, o interesse principal neste trabalho foi o estudo de métodos rápidos de visualização que usam a técnica de *rendering* de volumes.

Existem diversas correntes de pesquisa em torno de *rendering* de volumes. Como foi mencionado anteriormente, alguns pesquisadores procuram tratar o problema de redução do tempo de processamento através de *hardwares* dedicados [9, 22, 23]. Outros trabalhos paralelizam algoritmos de *rendering* usando, por exemplo, arquiteturas SIMD ou MIMD [24]. A desvantagem destes métodos, porém, é a mesma das técnicas baseadas exclusivamente em *hardwares* dedicados, a limitada disponibilidade e portabilidade das implementações. Outras vertentes procuram tratar o problema exclusivamente através de algoritmos rápidos para *rendering*, que sejam independentes de *hardware*. Uma estratégia mencionada anteriormente é sacrificar a qualidade da imagem em troca do tempo de processamento [10, 25]. Outro tipo de abordagem explora a coerência espacial presente nos dados volumétricos, procurando preservar a qualidade da imagem. Alguns métodos utilizam esta coerência na elaboração de modelos matemáticos que agilizam os cálculos de *rendering* [26]. Entretanto, muitos modelos matemáticos possuem critérios de erro que também acabam comprometendo a qualidade da imagem. Para evitar este compromisso entre qualidade e tempo de processamento, uma estratégia tem sido o uso de estruturas de dados que codificam a presença ou ausência de elementos de alta opacidade, de forma a evitar o cálculo e armazenamento de partes dos dados que pertencem a regiões transparentes. Considerando que 70% a 95% dos dados são normalmente classificados como transparentes, esta abordagem tem apresentado excelentes resultados para o problema da interatividade. Tais estruturas de dados incluem *octrees* e pirâmides [13, 25, 26, 27], k-d *trees* [14], *shell* [29] e transformadas de distância [28].

Em síntese, a estruturação da maioria dos métodos de *rendering* de volumes pode ser esquematizada de acordo com a Figura 1.1. Os dados volumétricos originais (Figura 1.1a) passam inicialmente por um pré-processamento (Figura 1.1b), onde são aplicadas operações como classificação e interpolação. Após esta etapa, define-se um modelo para a representação destes dados pré-processados (Figura 1.1c). Finalmente, o *rendering* é realizado sobre este modelo (Figura 1.1d), resultando na imagem para a visualização (Figura 1.1e). O *rendering* consiste de uma operação de projeção, que pode ser ortogonal (Figura 1.1$d_1$) ou perspectiva (Figura 1.1$d_2$), seguida de uma operação de tonalização (Figura 1.1$d_3$). Os métodos de *rendering* de volumes, apresentados neta seção, diferem

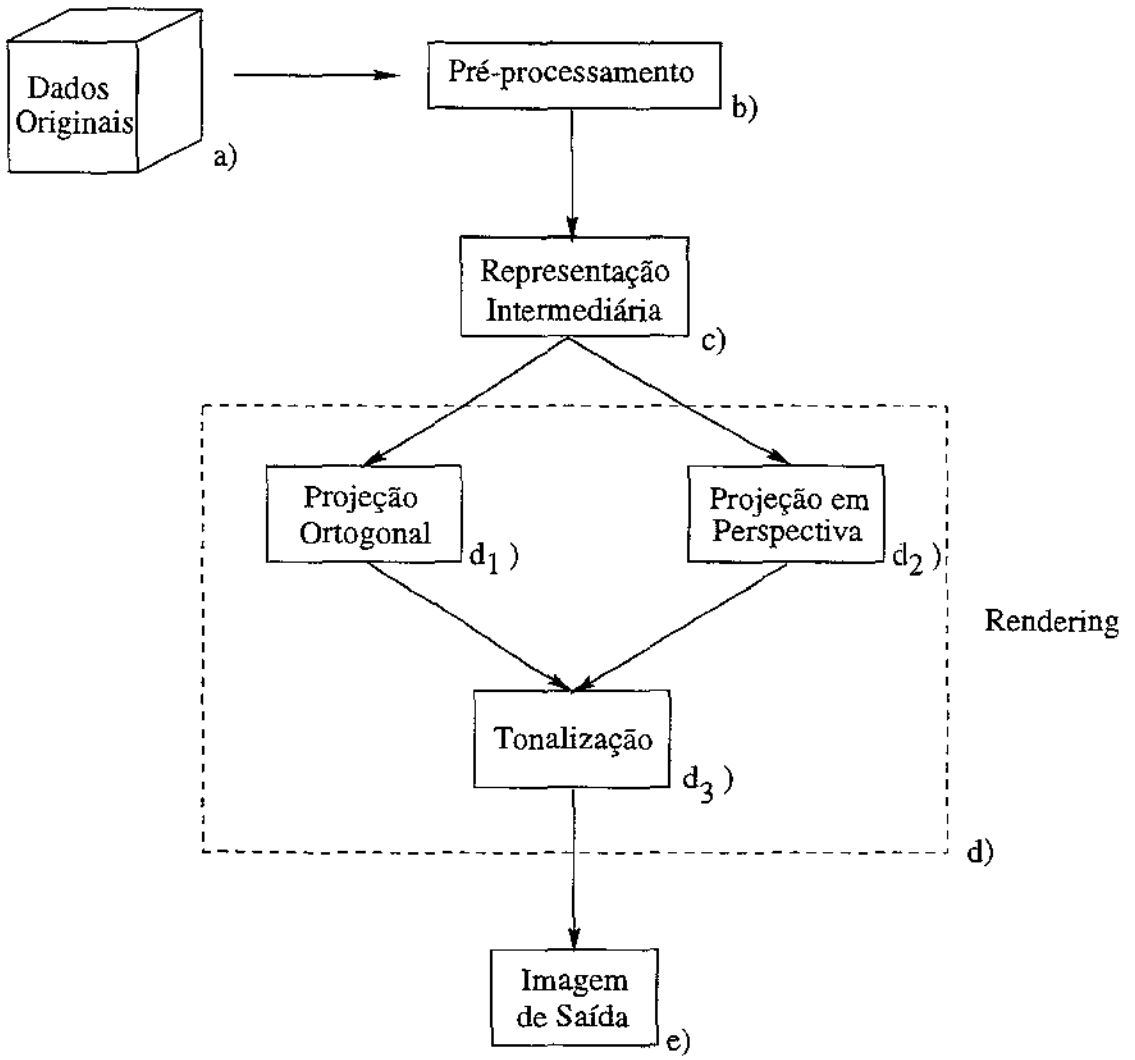



Figura 1.1: *Rendering* de Volume: esquema genérico.

basicamente no modelo adotado para a representação dos dados pré-processados, e no algoritmo de projeção.

## 1.3 Objetivos

Em 1993, Udupa e Odhner propuseram uma estrutura de dados, chamada *shell*, que codifica apenas as regiões de transição entre tecidos de interesse [29]. O método deles foi denominado *shell rendering* e representa uma solução eficiente para ambas técnicas de visualização, *rendering* de volumes e *rendering* de superfícies. Recentemente, *shell rendering* foi considerado um dos métodos mais rápidos de rendering de superfícies, mes-

mo quando comparado a métodos baseados em *hardwares* dedicados [30]. Ele também tem sido usado com sucesso em outras aplicações que demandam um método rápido de visualização [31]. Entretanto, *shell rendering* tem sido usado apenas com projeção ortogonal [32]. Em muitas aplicações, tais como endoscopia virtual, a projeção em perspectiva é fundamental para simular um passeio do observador no interior das estruturas. Portanto, visando o desenvolvimento futuro de aplicações clínicas que requerem interatividade e projeção em perspectiva, tais como endoscopia virtual, o objetivo principal deste trabalho foi a extensão do método *shell rendering* para prover projeção em perspectiva. A solução adotada introduz o conceito de perspectiva digital e assume, inicialmente, o observador no exterior das estruturas. O método desenvolvido foi denominado *shell rendering com perspectiva digital.*

A idéia principal em perspectiva digital é que as estruturas devem ser projetadas com tamanho inversamente proporcional à distância ao observador e que esta distância pode ser discretizada. Esta modificação introduziu algumas dificuldades que foram tratadas levando em conta o compromisso entre o tempo de processamento e a qualidade da imagem final.

## 1.4 Organização do Texto

Esta seção apresenta a forma como o texto foi estruturado, descrevendo brevemente o conteúdo de cada capítulo. O Capítulo 2 apresenta os conceitos básicos sobre visualização de volumes, necessários ao entendimento dos demais capítulos. Estes conceitos, tomados em conjunto, contituem um modelo tradicional de *rendering* de volumes, e envolvem a representação dos dados volumétricos originais (Figura 1.1a), reamostragem e classificação (Figura 1.1b) gerando um conjunto de dados pré-processados (Figura 1.1c), e finalmente o *rendering* (Figura 1.1d). O Capítulo 3 descreve o método *shell rendering* que propõe uma representação particular para os dados pré-processados. Este método é caracterizado pela estrutura de dados *shell* e pelo algoritmo de projeção ortogonal, que utiliza esta estrutura. Os problemas relacionados com a extensão do *shell rendering* para projeção em perspectiva são discutidos no Capítulo 4. Este capítulo também descreve o novo método *shell rendering* com perspectiva digital, que tem como característica uma modificação da estrutura de dados original (e.g. outra representação para os dados pré-processados), e um algoritmo rápido para projeção perspectiva. O Capítulo 5 discute os experimentos e resultados obtidos comparando o novo método com o *shell rendering,* sob os aspectos de tempo computacional e qualidade de imagem. Este capítulo também apresenta uma comparação entre perspectiva digital e tradicional. Os comentários conclusivos são finalmente apresentados no Capítulo 6.

# Capítulo 2

# Visualização de Volumes

Este capítulo apresenta as técnicas de processamento de imagens [33] e de computação gráfica [32] que são utilizadas em visualização de volumes, para gerar imagens de estruturas 3D a partir de um conjunto de dados volumétricos (Figura 2.1). Este processo está dividido em quatro etapas. A primeira etapa é a aquisição e representação dos dados volumétricos (Seção 2.1). Como os dados são amostrados normalmente de forma anisotrópica, a segunda etapa envolve técnicas de reamostragem dos dados (Seção 2.2). Muito embora a ordem entre as etapas de reamostragem e classificação das estruturas de interesse possa ser invertida, técnicas de classificação são apresentadas como uma terceira etapa na Seção 2.3. Após classificação, as imagens das estruturas são geradas usando técnicas de *rendering* . Na Seção 2.4, portanto, é descrito o *rendering* de volumes e o caso particular de *rendering* de superfícies.

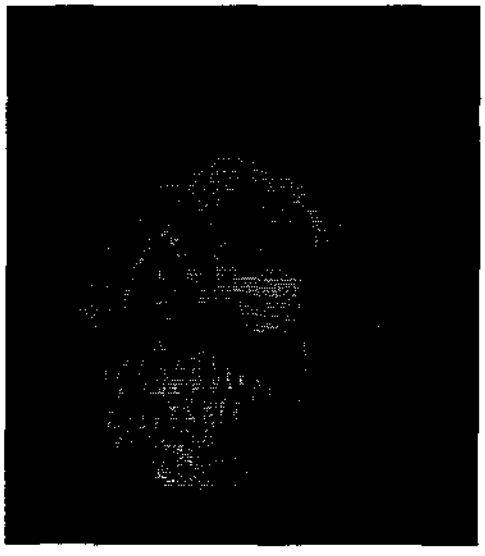


Figura 2.1: Visualização 3D de um crânio.

## 2.1 Representação de Dados Volumétricos

Uma imagem digital monocromática é vista como uma matriz $m \times n$, onde os índices de linha e coluna identificam um ponto na imagem e o valor numérico do elemento correspondente na matriz está associado a alguma grandeza física medida neste ponto. Os elementos desta matriz são chamados pixels ou pels (abreviações para *picture elements*). Em medicina [34], equipamentos tomográficos (e.g. CT, MRI, SPECT) geram dados volumétricos adquiridos em forma de imagens digitais monocromáticas de fatias paralelas e uniformemente espaçadas (Figura 2.2a), representando normalmente cortes transversais ao eixo longitudinal do paciente [1] (Figura 2.3). As imagens obtidas por equipamentos tomográficos possuem tipicamente resolução de $256 \times 256$ ou $512 \times 512$ pixels, com profundidade de 1 ou 2 bytes por pixel. O número de fatias pode variar em função do comprimento da região em estudo e do grau de detalhes desejado nesta região. Geralmente, um exame requer em torno de 40 fatias. Portanto, um conjunto de dados, com dimensões típicas, pode ocupar cerca de 20 Mbytes de memória ($512 \times 512 \times 40 \times 2 bytes$).

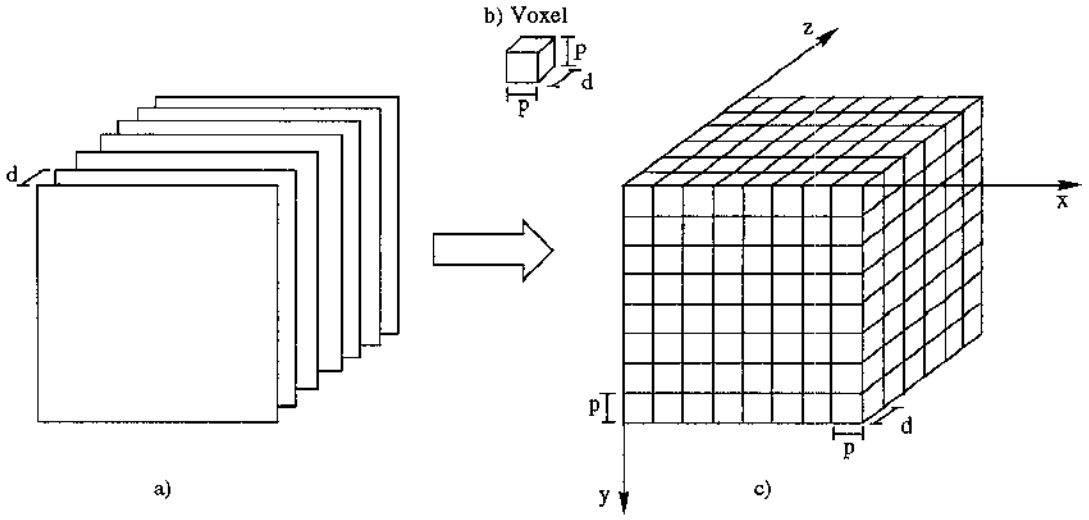



Figura 2.2: Formação da Cena 3D.

Considerando as dimensões $p \times p$ de um pixel nestas imagens e o espaçamento $d$ entre os cortes, a extensão do pixel em 3D forma um pequeno paralelepípedo de dimensões $p \times p \times d$ que é chamado voxel (Figure 2.2b). O centro dos voxels coincide com o centro dos pixels. O valor associado a cada voxel é um número inteiro, proporcional ao tom de cinza do pixel, na imagem correspondente, e representa a integração de alguma propriedade física que está sendo mensurada no interior do volume associado ao voxel. No caso da tomografia

[1] É comum a utilização de espaçamento variável entre os cortes quando existem regiões de maior interesse. Nestas regiões são feitos cortes mais próximos, permitindo uma melhor visualização dos detalhes.

por raios-x, por exemplo, a grandeza física medida é a densidade do tecido. Por essa razão, em visualização de imagens médicas, o valor associado a um pixel é frequentemente denominado *densidade do pixel*. O conhecimento de como as propriedades medidas variam de um tecido para outro é crucial na determinação de uma estrutura de interesse. As informações sobre essas propriedades físicas são usadas para reconstruir no computador a forma ou função de estruturas 3D. O conjunto de todos os voxels é chamado cena 3D, representando os dados volumétricos deste tipo de aplicação (Figura 2.2c).

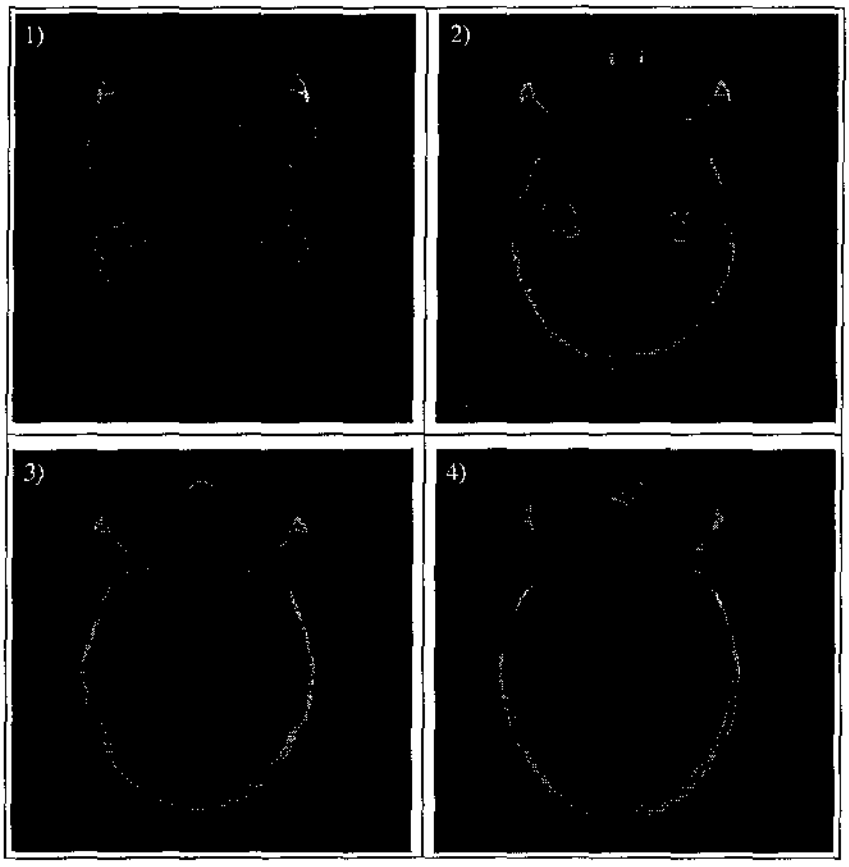
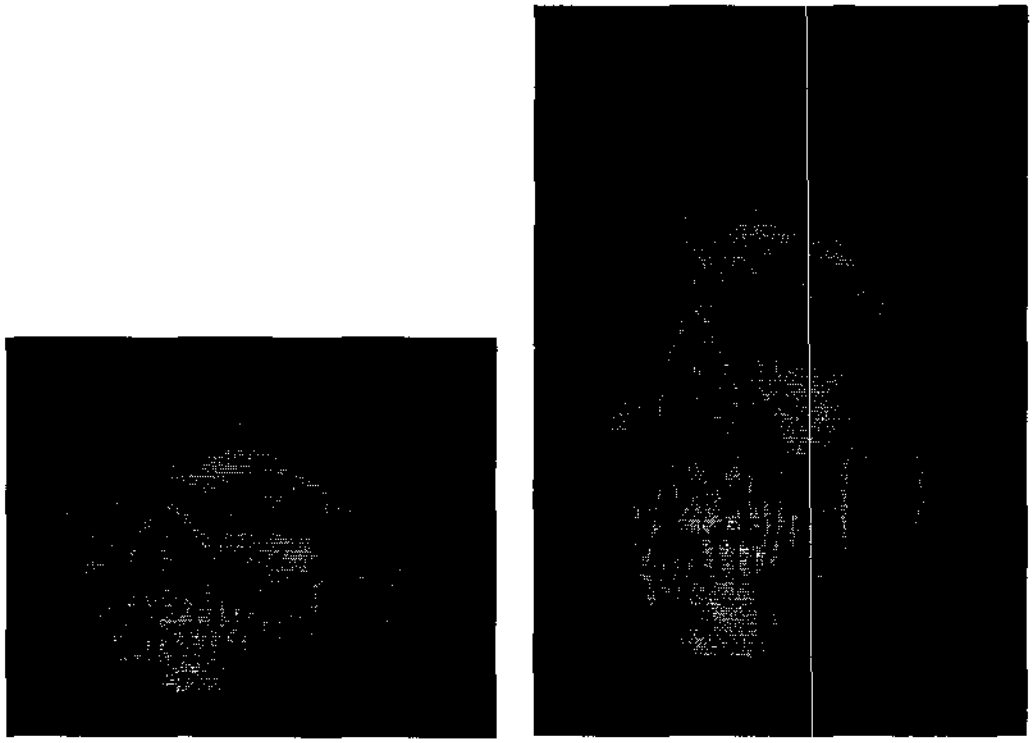



Figura 2.3: Exemplo de imagens de CT, obtidas ao longo do eixo longitudinal de um crânio seco.

Na maioria das situações, o espaçamento $d$ entre os cortes é diferente da dimensão $p$ dos pixels. Este espaçamento $d$, normalmente, é cerca de 10 a 15 vezes a dimensão $p$ dos pixels. Para evitar possíveis distorções na visualização 3D, devido a diferença entre as dimensões $p$ e $d$, a cena original precisa ser transformada em uma cena isotrópica. Esta transformação é chamada reamostragem [3, 34], e o objetivo desta operação é estimar o valor dos voxels numa nova escala, cujas dimensões destes voxels são as mesmas nas três direções principais (amostragem isotrópica). Algumas técnicas de reamostragem serão discutidas a seguir.

## 2.2 Reamostragem

A reamostragem, comumente chamada de *interpolação*, pode ser utilizada para aumentar ou reduzir o espaçamento entre os voxels em qualquer direção $x$, $y$ ou $z$. Ela também pode ser aplicada para gerar novas fatias ao longo de uma direção arbitrária. Neste caso, a técnica é chamada refatiamento (*reslicing*). Nesta seção, são apresentadas algumas técnicas comumente utilizadas em reamostragem, doravante chamada interpolação. Os exemplos e figuras ilustram o processo onde o objetivo é reduzir o espaçamento entre os voxels, principalmente ao longo da direção $z$, por ser a situação mais comum em computação de imagens médicas.



(a) (b)

Figura 2.4: Visualização 3D de um crânio com amostragens anisotrópicas: (a) $d > p$; (b) $d < p$.

A Figura 2.4 ilustra as distorções geradas na visualização 3D do crânio da Figura 2.1, em duas situações de amostragem anisotrópica. A Figura 2.4a mostra uma situação onde $d > p$, e a Figura 2.4b uma situação onde $d < p$.

A visualização sem distorções, portanto, é obtida quando após a interpolação temos $d = p$ (amostragem isotrópica). Esta interpolação pode ser feita tanto na cena original quanto numa cena previamente classificada, como veremos a seguir.

### 2.2.1 Interpolação na cena original

Existem diversas técnicas de interpolação na cena original. A principal diferença entre estas técnicas está na escolha dos elementos, pertencentes à amostragem original, usados na composição de um novo elemento interpolado $v_m$. Esta escolha geralmente recai sobre um conjunto de voxels contidos numa certa vizinhança de $v_m$. Os métodos de interpolação mais comuns [2] em computação de imagens médicas são:

1. *Regra do vizinho mais próximo*;
2. *Interpolação Linear*;
3. *Interpolação Trilinear*;

A regra do vizinho mais próximo e a interpolação linear são métodos que usam a vizinhança do elemento interpolado, em uma única direção. A Figura 2.5 ilustra o processo para interpolar uma fatia $m$, entre as fatias $n$ e $n+1$ da cena original. Neste caso, os vizinhos são tomados na direção de $z$.

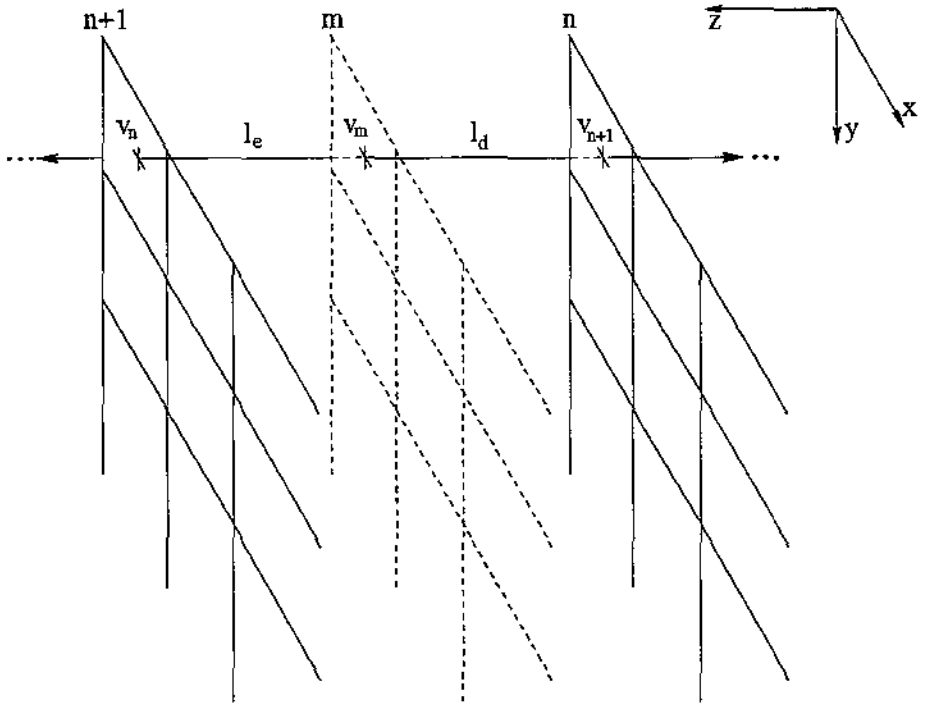



Figura 2.5: Ilustração para interpolação em uma única direção.

[2] Existem outros métodos de interpolação [35, 36] pertencentes a classes diferentes de algoritmos, como *truncated sinc* e *cubic spline* [9].

- **Vizinho mais próximo** (*Nearest Neighbor Rule*): é a forma mais simples de se determinar o valor dos voxels que serão introduzidos na cena interpolada. Neste tipo de interpolação, o valor da densidade (ou qualquer que seja o fenômeno físico representado) $d_o(v_m)$, do voxel $v_m$, é feito igual à densidade do voxel mais próximo de $v_m$ na cena original: $d_o(v_n)$ caso $l_e < l_d$, ou $d_o(v_{n+1})$ caso $l_d \leq l_e$, onde $l_d$ é a distância entre $v_m$ e $v_n$, e $l_e$ é a distância entre $v_m$ e $v_{n+1}$ (Figura 2.5).

- **Interpolação Linear**: na interpolação linear, assume-se que a variação de densidade é linear na direção $z$, entre os voxels $v_n$ e $v_{n+1}$. Sendo a densidade interpolada $d_i(v_m)$, do voxel $v_m$, obtida segundo a equação abaixo:

$$d_i(v_m) = d_o(v_n) + \frac{[d_o(v_{n+1}) - d_o(v_n)] \, . \, l_e}{l_e + l_d} \tag{2.1}$$

  onde: $(l_e + l_d) = d$, que é o espaçamento originalmente existente entre as fatias $n$ e $n+1$ (Figura 2.5).

É fácil concluir que a interpolação por vizinho mais próximo equivale à duplicação das fatias da imagem original. Já a interpolação linear é uma aproximação bem melhor, e normalmente apresenta bons resultados.

A técnica de interpolação trilinear usa, para a interpolação de $v_m$, os vizinhos presentes nas três direções principais $(x, y, z)$. A Figura 2.6 mostra o processo de interpolação de uma fatia $m$, entre fatias $n$ e $n+1$ da cena original. Observa-se que a fatia $m$ possui uma resolução maior que as fatias originais, portanto, a interpolação também é feita em $x$ e $y$.

- **Interpolação Trilinear**: neste método, a densidade interpolada $d_i(v_m)$ deve levar em consideração as densidades $d_o(v_k)$, $k = 1, 2, \ldots, 8$, dos oito voxels vizinhos, mais próximos de $v_m$: os voxels $v_1, v_2, v_3$ e $v_4$ na fatia da direita, e $v_5, v_6, v_7$ e $v_8$ na fatia da esquerda (Figura 2.6a). A união do centro dos voxels $v_i$, $i = 1, 2, ..., 8$, forma o cubóide da Figura 2.6b. A interpolação trilinear assume que a variação de densidade é linear em qualquer aresta deste cubóide. Portanto, utilizando as medidas de distância $\Delta_x = \Delta_y = p$, dos voxels da cena original, calcula-se a densidade $d_i(v_m)$ após sete operações similares à equação 2.1: primeiro é calculada a densidade em $p_1$, com base nas densidades em $v_1$ e $v_2$, em $p_2$, com base nas densidades em $v_3$ e $v_4$, e depois em $p_3$, com base nas densidades em $p_1$ e $p_2$. Similarmente, a densidade em $p_6$ é calculada usando mais três operações. Finalmente, a densidade $d_i(v_m)$ é calculada conhecendo as densidades em $p_3$ e $p_6$. É fácil concluir que, se o voxel $m$ estiver em uma das faces do cubóide $v_1, v_2, ..., v_8$, apenas três operações para estimar $d_i(v_m)$ são necessárias, e se estiver em uma aresta, apenas uma. O processo acima

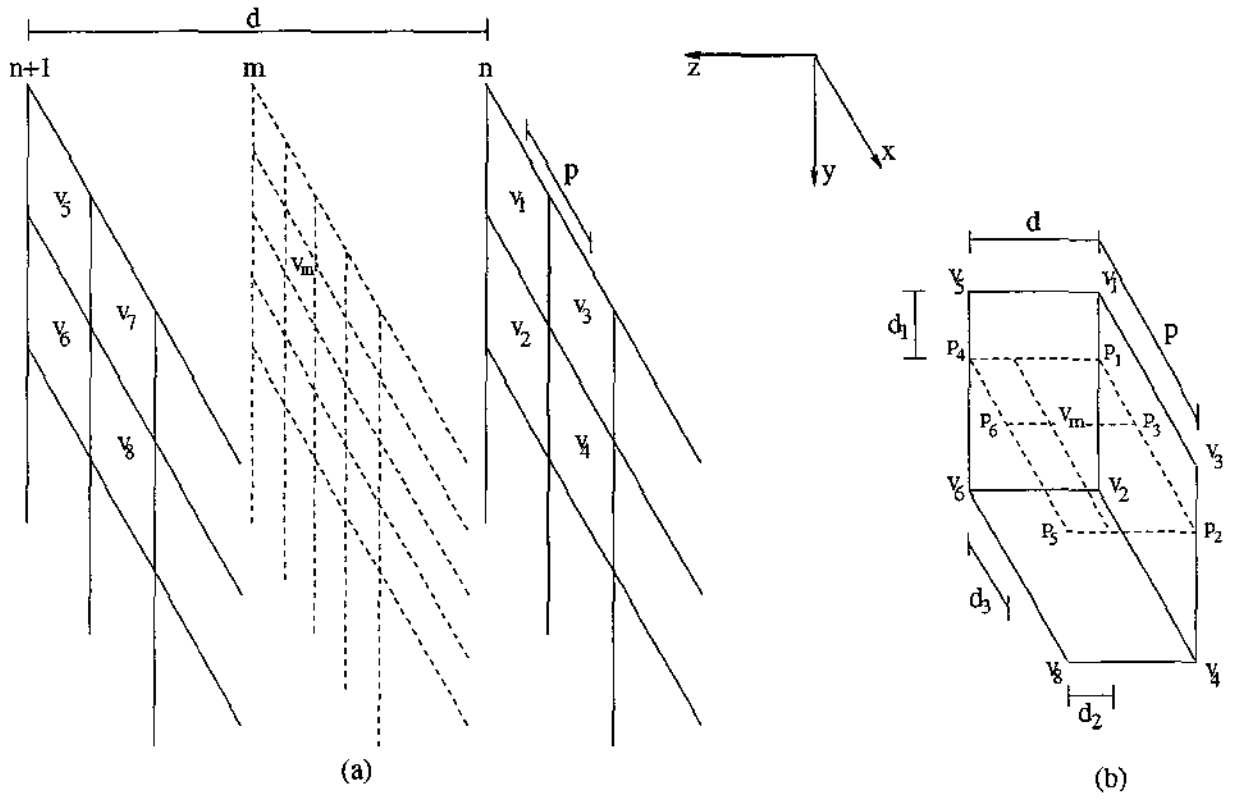



Figura 2.6: Interpolação trilinear.

é repetido para todos os outros voxels da fatia $m$. Observa-se que a interpolação linear é um caso particular da interpolação trilinear.

A interpolação trilinear também pode utilizar como base mais do que os oito voxels vizinhos mais próximos.

Fica evidente, à medida em que se analisa os métodos de interpolação apresentados, que a interpolação baseada na regra do vizinho mais próximo, em termos computacionais, é o método mais simples. Enquanto que o método de interpolação trilinear é o mais caro dos três, embora seja o método que apresenta o melhor resultado visual.

### 2.2.2 Interpolação após classificação

A classificação (Seção 2.3) é a operação que tem por objetivo selecionar as estruturas, pertencentes à cena 3D, que são de maior interesse ao usuário, para visualização. Em *rendering* de volumes, isto é feito atribuindo valores de opacidade, aos voxels da cena, proporcionais ao grau de interesse na visualização

Frequentemente, é desejável realizar a classificação antes da operação de reamostragem. A razão é que em muitos casos, a classificação requer a interação do usuário fatia

por fatia da cena, e se a reamostragem for feita antes, esta interação será necessária em um número muito maior de fatias. Como um exemplo, seja uma cena 4D representando imagens cardíacas de ressonância magnética (MRI) variando no tempo, consistindo de 10 fatias para cada 8 instâncias de tempo, com voxels de tamanho uniforme igual a $0.7mm \times 0.7mm \times 5mm$. Assumindo a criação de uma cena interpolada com voxels de tamanho $0.7mm \times 0.7mm \times 0.7mm$, a cena interpolada terá perto de 560 fatias. Se for necessário criar, por exemplo, mais 16 instâncias de tempo (para se obter maior suavidade na representação da dinâmica), também através de reamostragem, a cena 4D resultante terá mais que 1100 fatias. Nestas circunstâncias, a classificação interativa, fatia por fatia, seria inviável.

Considerando que a operação de classificação atribui valores de opacidade $\alpha \in [0,1]$, o resultado será uma cena em níveis de cinza, como a cena original. Neste caso, a interpolação pode ser realizada usando qualquer um dos métodos descritos para o caso de reamostragem na cena original.

Uma outra alternativa, porém, é a classificação binária (segmentação - Seção 2.3), que gera valores de opacidade $\alpha \in \{0,1\}$ (um voxel pertence ou não a uma estrutura de interesse). Muito embora este caso seja mais comum em técnicas de *rendering* de superfícies, pode também funcionar como uma etapa preliminar para geração de opacidades $\alpha \in [0,1]$, na vizinhança da superfície das estruturas, como veremos posteriormente.

O resultado da classificação binária é, portanto, uma cena binária. Neste caso, a técnica de interpolação baseada na forma (*shape-based interpolation* [37]) é a opção mais adequada.

- **Interpolação baseada na forma** (*Shape-based Interpolation*):
  A idéia principal nesta técnica é primeiro converter a cena de densidades (tons de cinza) em uma cena binária tal que os voxels com densidade 1 representam as estruturas de interesse, e aqueles com densidade 0 representam o resto da cena(classificação binária).

  A interpolação baseada na forma usa esta cena binária como entrda, e cria outra cena binária, onde os voxels possuem as novas dimensões especificadas. O método, ilustrado pela Figura 2.7a, consiste em primeiro converter a cena binária de entrada $S_b$ em uma cena $S_g$ cinza, ambas com o mesmo domínio, atribuindo a cada voxel $v$ em $S_g$ um valor que representa a menor distância entre $v$ e a borda (2D) em $S_b$, na fatia que contém $v$. Esta operação é conhecida como *transformada de distância 2D.*

  O valor é positivo se $v$ tem densidade 1 em $S_b$, e negativo caso contrário. Em seguida a cena $S_g$ é interpolada usando um método linear, trilinear, ou outra regra de interpolação, para criar outra cena $S'_g$. Lembrando que os números associados aos voxels em $S'_g$ representam distâncias até a borda, cria-se a cena binária interpolada

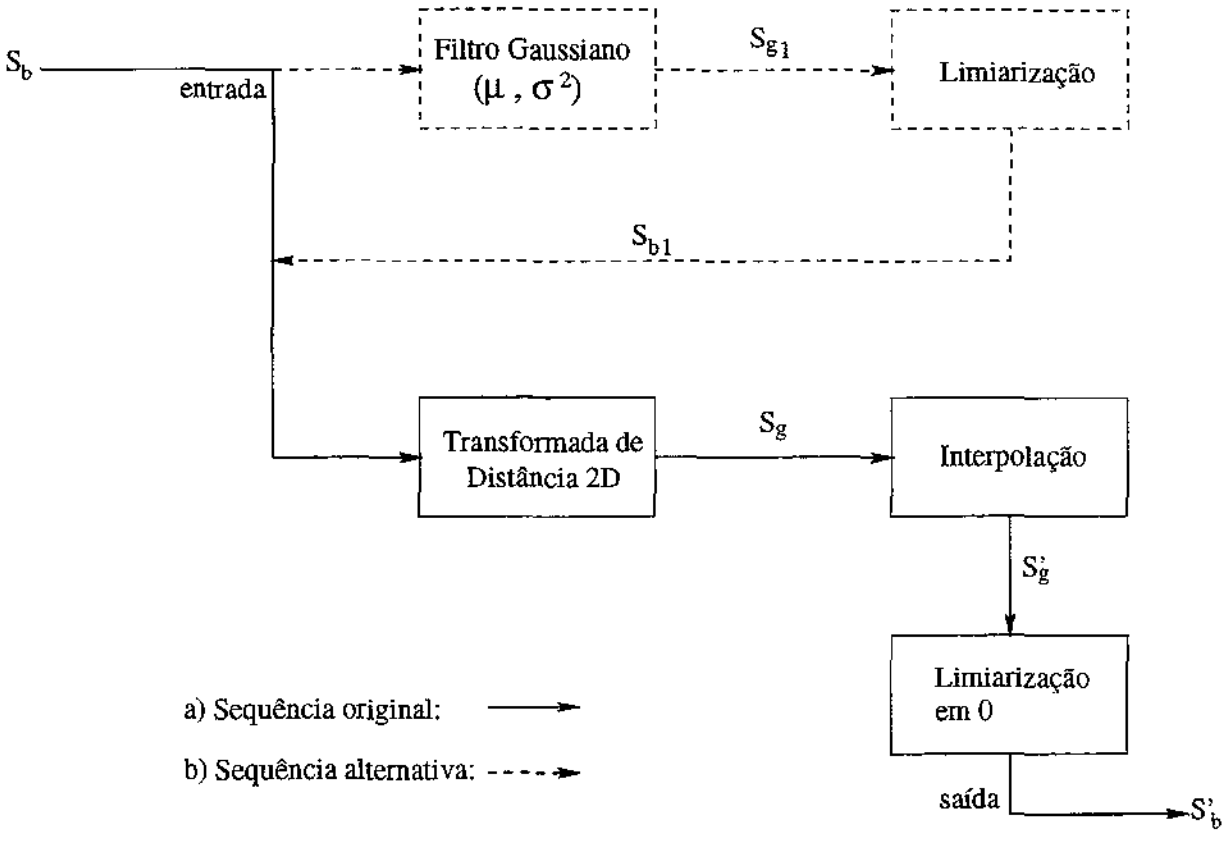



Figura 2.7: Fluxo de dados para a interpolação baseada na forma.

$S'_b$ pela atribuição de valores de densidade 1 aos voxel em $S'_g$ com valores positivos. A todos os outros voxels são atribuídos valores de densidade iguais a 0. Esta operação é equivalente a uma limiarização (*thresholding*) de $S'_g$ em 0.

Uma alternativa para este método é acrescentar uma filtragem gaussiana preliminar à cena binária de entrada, gerando uma cena cinza $S_{g1}$, e depois realizar uma limiarização, resultando em uma nova cena binária $S_{b1}$ (Figura 2.7b). Este processamento melhora a qualidade da cena binária interpolada, suavizando os contornos da estrutura de interesse, além de eliminar eventuais ruídos, resultantes do processo de classificação binária. Algumas outras variações deste método têm sido propostas [38, 39, 40]. Foi também demonstrado [37, 38, 39, 41] que a interpolação baseada na forma possibilita análises quantitativas, tais como medições de volume, mais precisas.

## 2.3 Classificação

Em termos simples, o *rendering* de volumes, em medicina, pode ser resumido como uma transformação projetiva de alguma forma similar, porém mais complexa, que operações radiográficas (CT, MRI, PET, etc). Enquanto que no segundo caso propriedades do tecido não são conhecidas em todos os pontos da trajetória de projeção, no primeiro caso, uma vez que os valores das propriedades associadas aos tecidos são conhecidas em todos os voxels. Isso permite criar a projeção de várias formas sofisticadas, assim como enfatizar ou obscurecer aspectos selecionados da cena.

O modelo conceitual usado em visualização de volumes é tal que a cena é considerada um bloco colorido e translúcido, cuja cor e opacidade (grau de transparência) pode ser diferente para diferentes regiões. A luz que simuladamente incide sobre este bloco pode ser transmitida e, através de sua reflexão, diferentes tipos de tecido podem ser visualizados ao mesmo tempo pelo observador. Fica claro que não existe a idéia de modelos geométricos para objetos, mas de regiões de interesse.

Desde que é necessário especificar quais aspectos devem ser enfatizados (baixo grau de transparência) e quais deve ser obscurecidos (alto grau de transparência), assim como sua cor, é imperativo que esses aspectos sejam identificados voxel a voxel através de algum tipo de processamento de cena.

Pesquisadores da Pixar, em 1985, foram os primeiros a usar estas técnicas de forma mais sofisticada em dados médicos 3D [43]. A técnica usada, descrita em termos gerais por Smith [44] e apresentada em detalhes por Drebin et al. [11], consiste em estimar a fração de cada material (ar, músculo, gordura, osso) no interior dos voxels e usar estas frações para calcular uma cor e uma opacidade parcial para cada voxel. O método de Levoy [12] é semelhante, mas calcula as cores e opacidades diretamente do valor escalar de cada voxel.

A classificação de uma estrutura em uma cena 3D também pode ser vista como uma segmentação nebulosa que associa a cada voxel da estrutura uma opacidade no intervalo $[0, 1]$ e uma opacidade nula ao restante dos voxels, fazendo com que a cena 3D seja vista como um volume semitransparente. No caso de múltiplas estruturas, a classificação pode associar um par ordenado $(\alpha, \beta)$ para cada voxel da cena 3D, onde $\alpha$ é o valor de opacidade no intervalo [0,1] e $\beta$ é um rótulo no conjunto $\{0, 1, \ldots, n-1\}$ que identifica a estrutura de interesse. O objetivo dos rótulos é a identificação posterior das estruturas para permitir visualização, manipulação e análise de cada estrutura de forma independente. A classificação, portanto, deve considerar valores maiores de opacidade para os voxels que pertencem às estruturas de interesse e valores nulos para os demais. Um caso particular é a classificação n-ária que associa valores $\alpha \in \{0, 1\}$ e $\beta \in \{0, 1, \ldots, n-1\}$. O resultado desse tipo de classificação, que normalmente é chamada de *segmentação*, é

essencialmente, uma cena n-ária.

Como exemplo, pode-se imaginar uma situação onde as estruturas de interesse são pele e osso. Neste caso, a classificação pode associar valores de opacidade no intervalo [0,1] para os voxels que pertencem à pele, valor 1 aos voxels que pertencem ao tecido ósseo e valor 0 aos outros voxels.

As dificuldades do processo de classificação podem ser ilustradas com o seguinte exemplo. Imagine uma função de opacidades $\alpha_d(v)$ aplicada à cena 3D que seja baseada apenas na densidade dos voxels. Para que esta função seja capaz de distinguir estruturas diferentes, estas estruturas devem ser representadas por intervalos disjuntos de densidade de voxel (Figura 2.8a).

Infelizmente, na maioria das situações isto não ocorre e a classificação se torna um problema mais complexo. Um outro problema ocorre quando a estrutura mais externa é espessa e mesmo associando valores baixos de opacidade para seus voxels, a estrutura mais interna nunca é visualizada em conjunto na mesma projeção. Este problema é melhor compreendido na Seção 2.4, mas sua solução pode ser encontrada acrescentando a função de opacidades $\alpha_g(v)$ aplicada ao gradiente dos voxels (Figura 2.8b). A idéia é atribuir valores de opacidade mais altos apenas para os voxels que estão na região em torno da superfície de cada estrutura. A função de opacidades pode então ser definida em função da densidade e do gradiente dos voxels, através da seguinte equação:

$$\alpha(v) = \alpha_d(v) \times \alpha_g(v) \tag{2.2}$$

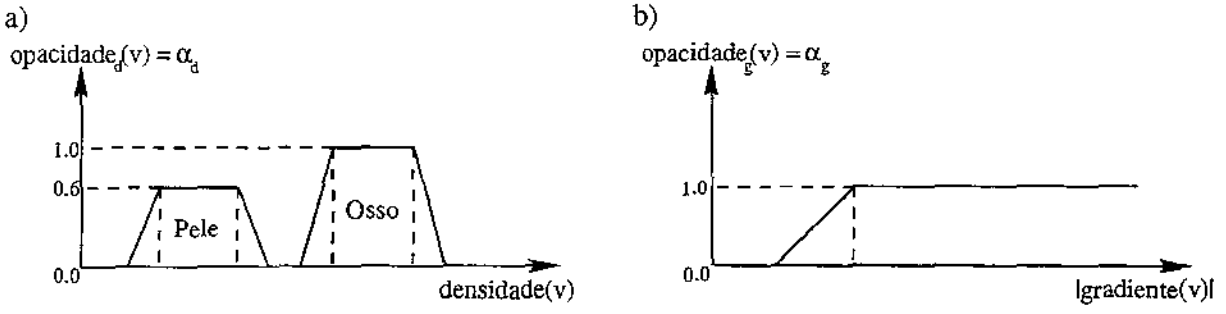



Figura 2.8: Classificação Baseada na Densidade e no Gradiente.

Uma opção interessante é realizar a atribuição de opacidades a partir de uma cena binária pré-segmentada (Figura 2.9a). A cena binária, sem nenhuma espécie de tratamento, resulta em um tipo de *rendering* de superfícies, pois todos os voxels, pertencentes à estrutura de interesse, seriam totalmente opacos. Isso faz com que apenas os voxels da superfície do objeto possam ser vistos. Este não é um procedimento recomendável, pois a cena binária mantém toda informação do interior da estrutura, sem que essa informação

seja efetivamente usada. Entretanto, é possível usar a cena binária para gerar um mapa de opacidades, cujos valores $\alpha \in [0, 1]$. Para tanto, um filtro gaussiano pode ser empregado, como método de classificar opacidades nesse intervalo. Essa técnica pode ser útil quando se deseja visualizar um tecido, de forma que a opacidade seja progressivamente aumentada, na medida em que se caminha para o interior do tecido (Figura 2.9b). A filtragem gaussiana, sobre a imagem binária, tem a propriedade de suavizar os contornos das estruturas, e se realizada com parâmetros de *média* e *variância* corretos, pode-se obter imagens de qualidade muito boa (a sugestão é usar média 0 e variância 1.5).

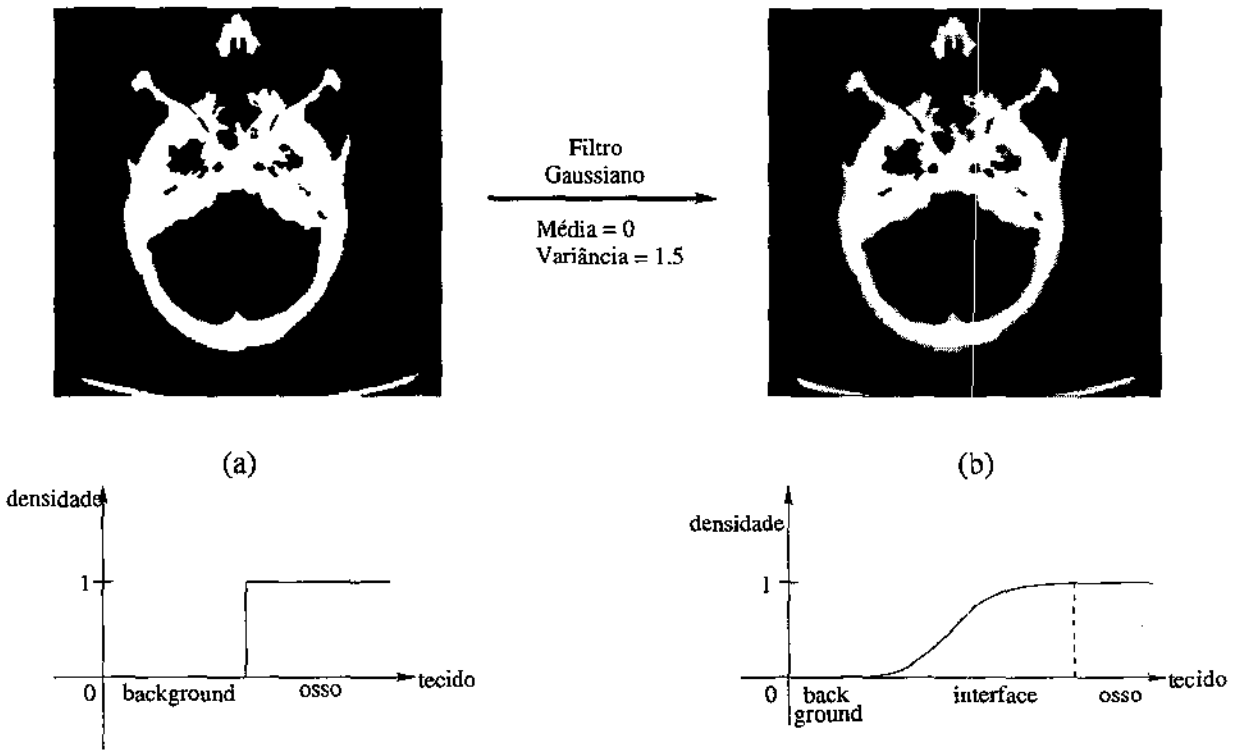



Figura 2.9: Opacidade baseada na densidade. (a) Tomada em uma cena binária e (b) Tomada na mesma cena binária, processada com filtro gaussiano.

## 2.4 *Rendering* de Volumes

Operações de *rendering* simulam um ambiente de visualização com fontes de luz, observador, objetos e plano de visualização em posições definidas. O ambiente utilizado neste trabalho é apresentado na Figura 2.10. Neste ambiente, o observador e uma fonte de luz branca estão na mesma posição sobre o semi-eixo $z^-$, a cena é centrada na origem do sistema $xyz$ e o plano de visualização é paralelo ao plano $xy$ e posicionado em $(0, 0, d/2)$, onde $d$ é a diagonal principal da cena. Assume-se que essa cena pode ser rodada na frente do observador, que permanece em uma posição $(0, 0, p)$, $p < -d/2$, sobre o semi-eixo $z^-$ para qualquer direção de visualização. As restrições para os valores de $p$ e para o posicionamento do plano de visualização fazem com que o observador esteja sempre fora da cena 3D e que toda a cena possa ser projetada sobre o plano de visualização.

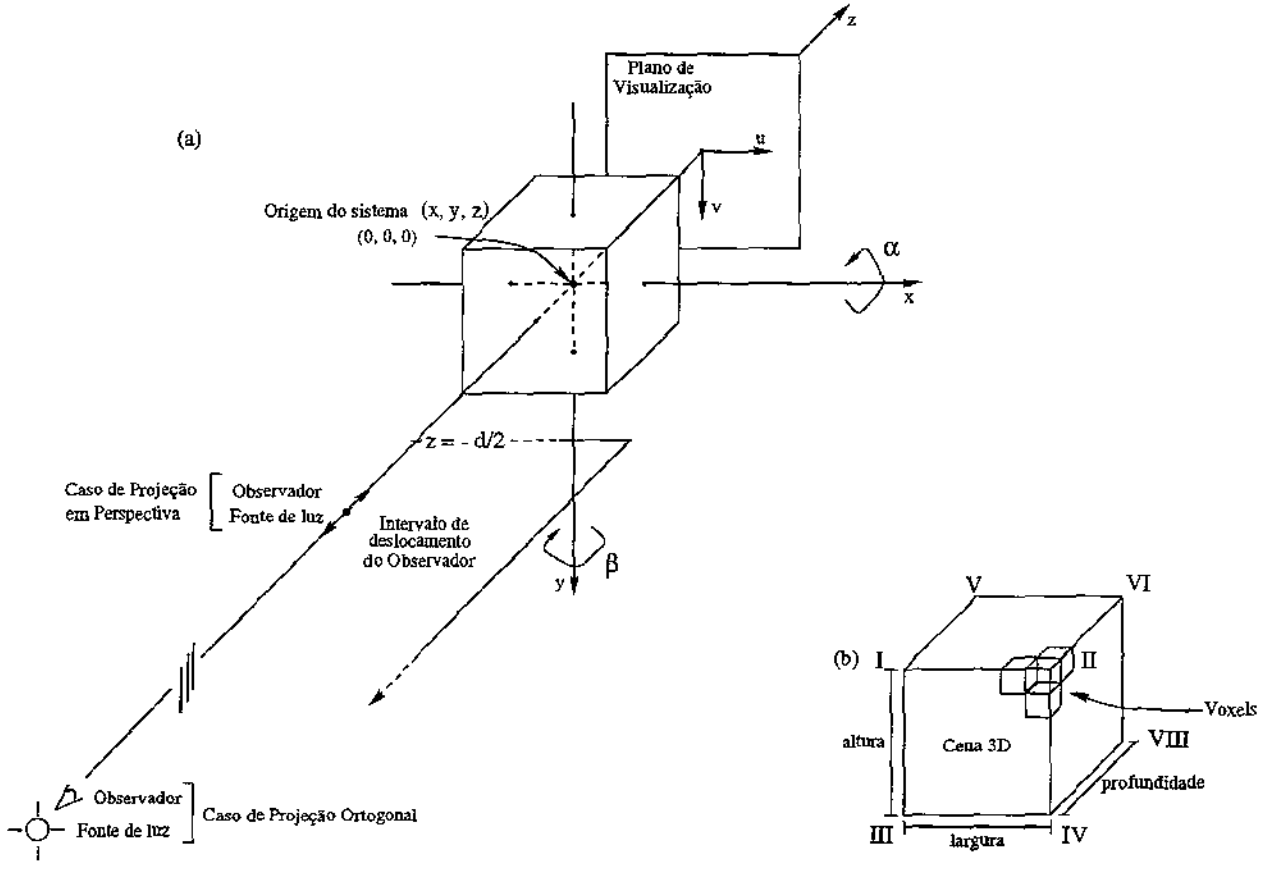



Figura 2.10: Ambiente de visualização.

No *rendering* de volumes, a cena é vista como um volume semitransparente, cuja imagem resulta da parcela de luz refletida e transmitida através dos voxels que chega aos olhos do observador. Para gerar uma imagem especifica-se inicialmente a direção

de visualização (Seção 2.4.1). O *rendering* de volumes propriamente dito envolve dois aspectos distintos. O primeiro diz respeito à identificação dos voxels do interior da cena 3D que serão projetados no plano de visualização (Seção 2.4.2). O segundo diz respeito à tonalização (e a possível utilização de cor) que será associada aos voxels projetados, para compor a imagem final (Seção 2.4.3).

### 2.4.1 Transformações geométricas aplicadas à cena 3D

Uma transformação geométrica pode ser aplicada a qualquer um dos elementos do ambiente de visualização. Estas transformações definem o posicionamento relativo entre os elementos do ambiente, e consistem em operações de rotação, translação e escalamento aplicadas a cada ponto $(x, y, z)$ , definindo seu novo posicionamento $(x', y', z')$. Inicialmente os elementos do ambiente de visualização têm o posicionamento absoluto definido em relação ao sistema de coordenadas $(x, y, z)$, conforme o modelo apresentado na ilustração da Figura 2.10a.

Neste modelo, transformações geométricas são aplicadas apenas sobre a cena, que pode ser rodada em torno dos eixos $(x, y)$, definindo dois ângulos de rotação $\alpha$ e $\beta$ (dois graus de liberdade). Essas rotações são realizadas sempre em torno do centro do volume, de modo que a visualização de qualquer uma das faces da cena pode ser obtida.

As transformações geométridas de rotação, aplicadas sobre os voxels da cena, podem ser representadas pela seguinte equação:

$$[x', y', z'] = [x, y, z] \times M \tag{2.3}$$

onde
$M$ é a matriz de rotação, em função dos ângulos $\alpha$ e $\beta$.
$[x, y, z]$ são as coordenada originais de um voxel da cena.
$[x', y', z']$ são as coordenada do mesmo voxel após a rotação.

A matriz $M$ é a composição (multiplicação) das conhecidas matrizes de rotação em torno dos eixos $x$ e $y$, que correspondem aos ângulos $\alpha$ e $\beta$, respectivamente. A ordem das matrizes na equação abaixo, indica que a rotação é feita primeiro em torno do eixo $x$ (as matrizes são aplicadas da direita para a esquerda).

$$M = R_\beta \times R_\alpha \tag{2.4}$$

portanto,

$$M = \begin{bmatrix} cos\beta & 0 & -sen\beta & 0 \\ sen\alpha \times sen\beta & cos\alpha & sen\alpha \times cos\beta & 0 \\ cos\alpha \times sen\beta & -sen\alpha & cos\alpha \times cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

É importante ressaltar que a transformação definida pela matriz $M$, com dois graus de liberdade (ângulos $\alpha$ e $\beta$), é suficiente para que todas as faces da cena possam ser visualizadas.

### 2.4.2 Projeção

A projeção é uma transformação de 3D para 2D aplicada aos pontos da cena. A projeção pode ser ortogonal, supondo que o observador está no infinito (raios de visualização são paralelos), ou em perspectiva. Na projeção ortogonal as estruturas aparecem sem escalamento enquanto que na projeção em perspectiva as estruturas mais próximas do observador aparecem maiores do que as estruturas mais distantes. Para facilitar a explicação, assume-se a princípio o caso de projeção ortogonal.

Técnicas baseadas em *rendering* de volumes geram imagens diretamente dos dados volumétricos, sem a necessidade de uma representação intermediária real [42, 45, 46], como por exemplo modelos de superfícies baseados em triângulos.

Uma importante distinção, entre técnicas de *rendering* de volumes, está na forma e na ordem em que os voxels são processados na criação da imagem para visualização. Existem duas maneiras de realizar a projeção: algoritmos baseados em *image-order* e algoritmos baseados em *object-order*.

Algoritmos baseados em *image-order* varrem pixel a pixel o plano de visualização, onde a imagem final será formada, e determinam quais voxels da cena contribuem para a composição do pixel corrente.

A técnica de projeção que melhor representa a classe baseada em *image-order*, por ser a técnica mais difundida dentro dessa classe de algoritmos, é a técnica de *ray-casting*. Exemplos de algoritmos de *rendering* de volumes que usam *ray-casting* podem ser encontrados nos trabalhos de Levoy, Sabella e Upson [12, 47, 48].

*Ray-casting* constrói as imagens pelo lançamento de raios através da cena. Estes raios são lançados, paralelos à direção de visualização, a partir de cada pixel da imagem em construção. Operadores de classificação de superfície são aplicados a todos os voxels encontrados. Uma cor e uma opacidade específicas são atribuídas às estruturas, e finalmente todos esses componentes são compostos para formar o valor associado ao pixel da

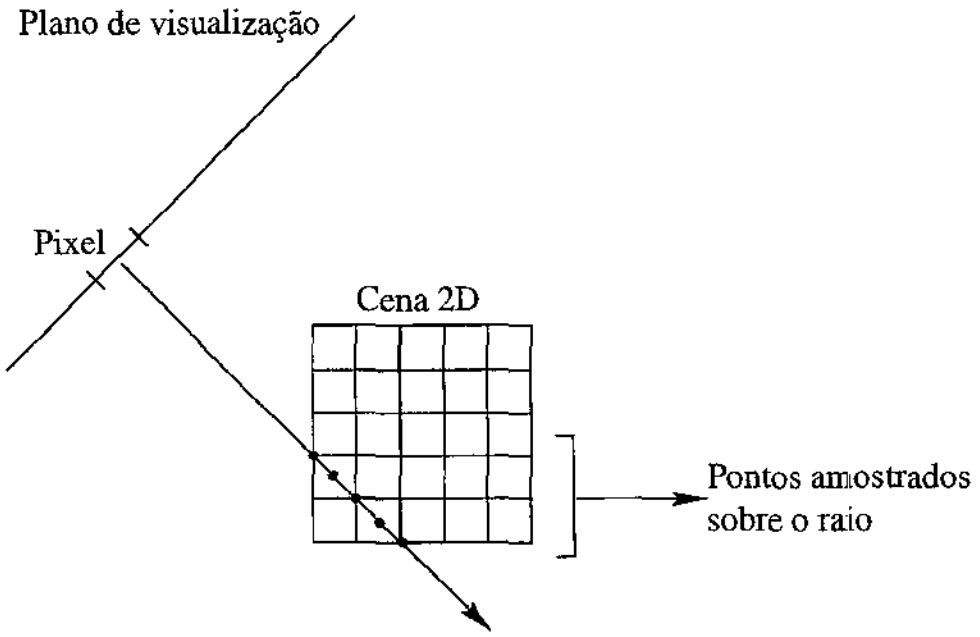



Figura 2.11: Ilustração em 2D da técnica de *ray-casting*.

imagem (Figura 2.11). A principal desvantagem deste método é a sua complexidade computacional. Apesar de ser possível realizar o cálculo da cor e da opacidade numa etapa de pré-processamento, a característica principal dessa técnica é que ela funciona através da amostragem de pontos sobre os raios de busca, e as componentes calculadas sobre esses pontos são usadas na composição do valor do pixel associado ao raio. O problema decorre do fato de que os pontos amostrados sobre os raios não estarão sempre posicionados nos centros dos voxels (ver Figura 2.11). Essa característica implica no emprego de técnicas de interpolação para calcular o valor das componentes nas posições amostradas. A interpolação realizada em tempo de *rendering* torna o processo de *ray-casting* computacionalmente caro.

Os algoritmos baseados em *object-order*, ao contrário daqueles baseados em *image-order*, varrem primeiro os voxels da cena e determinam quais pixels um determinado voxel pode afetar. Exemplos de métodos de *rendering* de volumes que usam *object-order* podem ser encontrados nos trabalhos de Drebin, Lacroute, Upson e Westover [11, 16, 48, 49]. Como a varredura ocorre sobre a cena, os pontos podem sempre ser escolhidos de forma que coincidam com os centros dos voxels. Por isso, técnicas baseadas em *object-order* podem evitar o emprego de interpolação em tempo de *rendering*, tipicamente utilizada em métodos baseados em *image-order*.

A técnica de acesso que melhor representa a classe *object-order*, também por ser a técnica mais usada, é a técnica de *voxel projection* [34].

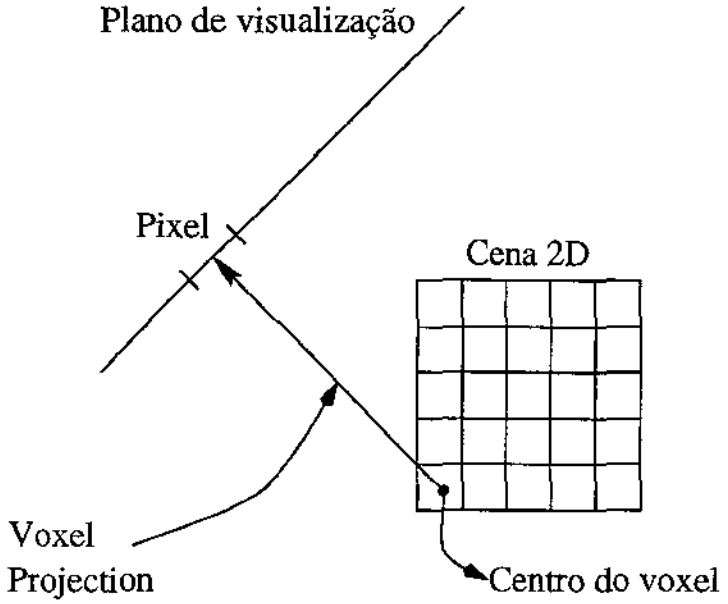



Figura 2.12: Ilustração em 2D da técnica de *voxel projection*.

*Voxels projection* constrói as imagens pela projeção dos voxel, a partir da cena, sobre o plano onde a imagem está sendo formada. Este método rastreia os voxels da cena projetando cada um deles, respeitando a direção de visualização, sobre um ou mais pixels da imagem em construção. As contribuições de todos os voxels projetados sobre um mesmo pixel $p_i$ são combinadas, de acordo com seus atributos de cor (ou brilho) e opacidade, para formar o valor de tonalização de $p_i$ (ver Seção 2.4.3). Cada voxel é projetado a partir de seu centro (Figura 2.12).

Considerando a possível saturação da luz que atravessa o volume semitransparente, deve-se determinar, para cada direção de visualização, a ordem de proximidade dos voxels que estão sendo projetados, em relação ao observador. Essa operação tem por objetivo possibilitar a remoção de estruturas escondidas. No *ray casting*, esta ordem é obtida naturalmente à medida que o raio penetra na cena. Porém, para a técnica de *voxels projection* essa ordem deve ser obtida pelo emprego de algum algoritmo específico. A principal diferença, entre os métodos que utilizam projeção de voxels, é exatamente o tipo de algoritmo de remoção de superfícies escondidas utilizado. Existem várias técnicas de remoção de superfícies escondidas [34], dentre outros algoritmos, os mais comuns são: *depth-sort*, *z-buffer*, *back-to-front* (BTF) e *front-to-back* (FTB).

A remoção de superfícies escondidas consiste em identificar sobre cada raio de projeção, o ponto a partir do qual as contribuições dos voxels não-nulos (voxels cuja opacidade é diferente de zero), projetados sobre um pixel $p_i$, deixam de influir na composição da cor desse pixel. Isso ocorre 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 plano de visualização é maior ou igual a 1 (opacidade absoluta).

Os algoritmos para remoção de estruturas escondidas, mais comumente usados com o método de projeção de voxels, são descritos abaixo:

- **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 projeção) entre o voxel que está sendo projetado e o pixel. Um novo valor de tonalização e um novo valor de distância, referentes a um outro voxel não-nulo, são associados ao pixel apenas se o novo valor de distância for menor que o anterior. Isto garante que, entre os voxels não-nulos interceptados pelo raio de projeção, 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.

- **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 observador. Esse processo de classificação torna-o mais lento e ligeiramente mais complexo que o algoritmo *z-buffer*, descrito no tópico anterior. Porém, este método permite que a cor de um pixel seja calculada pela combinação das contribuições de vários voxels. Logo, é adequado também para *rendering* de volumes.

- **back-to-front**(BTF): o algoritmo BTF [12, 50, 51] explora a informação de proximidade com o observador, implícita na estrutura do modelo de volume que armazena a cena (coluna-por-coluna, linha-por-linha, fatia-por-fatia), evitando a utilização de qualquer tipo de classificação de distâncias. Entretanto requer que todos os voxels sejam rastreados. 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 (Figura 2.13a).

- **front-to-back**(FTB): este algoritmo baseia-se na mesma idéia do algoritmo anterior, mas acessa os elementos do modelo de volume na ordem em que eles se afastam do observador (Figura 2.13b). A vantagem do algoritmo FTB [34, 51] (que também não precisa classificar distâncias) em relação ao BTF, é que na projeção com remoção de superfícies escondidas, o FTB pode acelerar o processo reduzindo

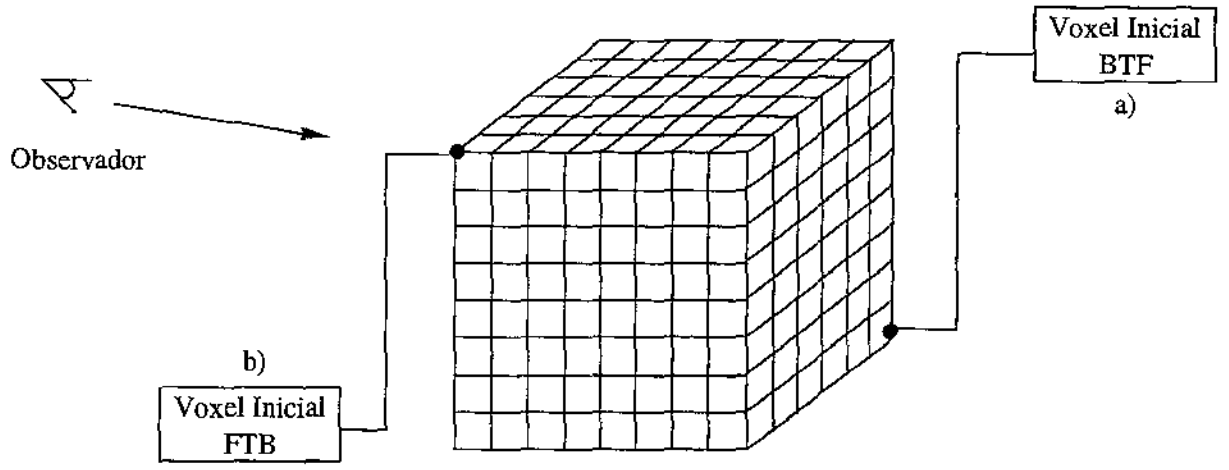



Figura 2.13: Cena 3D, ilustrando a forma de acesso para os casos BTF e FTB. a) Voxel inicial do ordenamento BTF. b) Voxel inicial do ordenamento FTB.

o número de voxels examinados, evitando varrer toda a cena [34] como o método BTF necessariamente faz. Esta observação é melhor compreendida na Seção 2.4.3.

Um problema encontrado nas técnicas que usam projeção de voxels é que alguns pixels no espaço imagem podem não ser tonalizados devido a problemas de erros de truncamento e precisão aritmética, ficando com a cor do fundo. Isto acaba provocando o aparecimento de pequenos "buracos"na imagem final. Existem várias formas de corrigir esse problema. Certos métodos empregam algum tipo de filtragem sobre a imagem final, como por exemplo o cálculo da tonalização dos pixels, onde aparecem os "buracos"por interpolação da tonalização de seus vizinhos. Outros métodos procuram cobrir esses pixels através de mudanças de escala, onde consideramos as dimensões de um pixel menores que as de um voxel, portanto, um mesmo voxel passa a cobrir mais de um pixel.

Nas técnicas de *ray-casting*, esse problema não ocorre, porque necessariamente existe um raio de busca lançado a partir de cada pixel do espaço imagem.

### 2.4.3 Tonalização

A tonalização é a operação que associa a cada pixel da imagem, que está sendo formada no plano de visualização, um valor de cinza (ou cor), visando a distribuição de luminosidade sobre essa imagem. Essa operação possibilita a ilusão de tridimensionalidade perdida na projeção do objeto. Adicionalmente, uma matiz pode ser associada a cada estrutura, de modo que a tonalização passa a associar uma cor a cada pixel da imagem [3, 34].

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 $p_i$ (Figura 2.14).

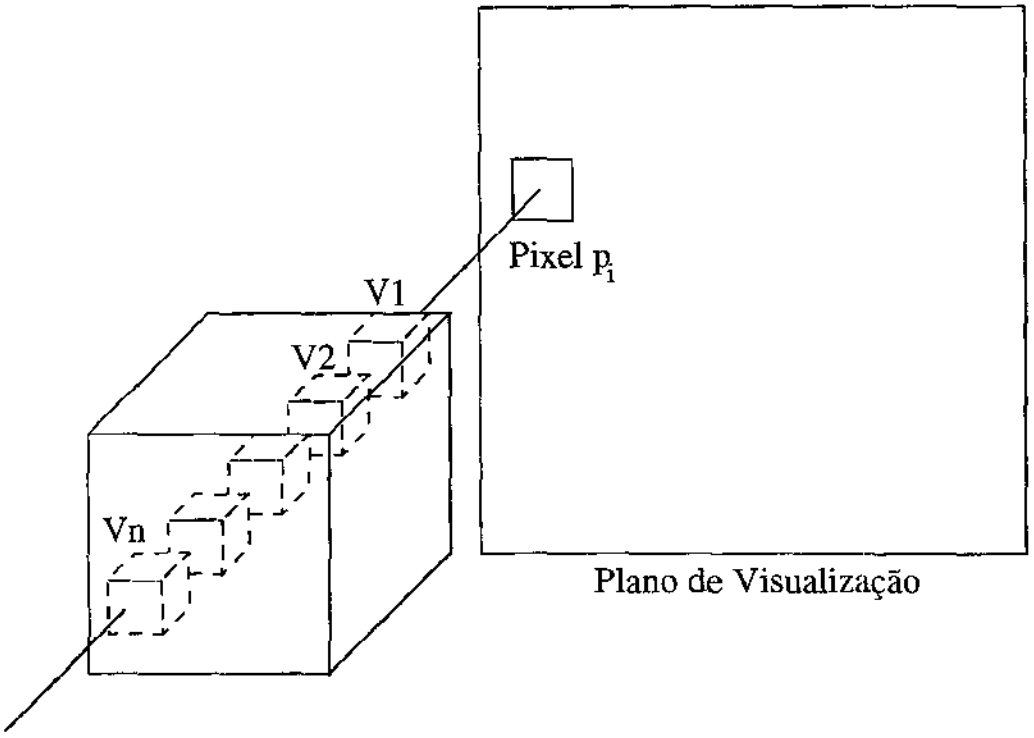



Figura 2.14: Tonalização do pixel $p_i$, calculada através da combinação das contribuições dos voxels $(v_1, \ldots, v_n)$.

A combinação dos voxels $v_1, v_2, ..., v_n$ deve levar em conta a opacidade acumulada a cada passo do processo, pois se essa opacidade acumulada atinge um valor muito próximo de 1 (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.

É importante determinar o momento em que um pixel $p_i$ atingiu o estado de saturação, pois nesse caso, não existe a necessidade de se continuar calculando a contribuição relativa a cada um dos voxels restantes, projetados sobre $p_i$. Esse tipo de medida aumenta significativamente o desempenho do algoritmo de *rendering* .

A equação abaixo mostra uma forma como a tonalização associada ao pixel $p_i$ pode ser estimada (ver Figura 2.14):

$$S_{p_i} = \sum_{i=1}^{n} I_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{2.6}$$

A componente $I_i \alpha_i$, da equação 2.6, 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 [52] ou Phong [53]; e $\alpha_i$ 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 semitransparente 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 2.6 expressa esse efeito através da 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 plano de visualização.

O modelo de iluminação mais difundido na área de visualização de imagens médicas [3] 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 linear 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 escuro será esse ponto, e quanto mais próximo do plano, mais clara será sua tonalização. Esse efeito é modelado pela equação 2.7.

$$I_{dist}(u,v) = \frac{I_{max} - I_{min}}{d_{min} - d_{max}}[d - d_{max}] + I_{min} \tag{2.7}$$

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:

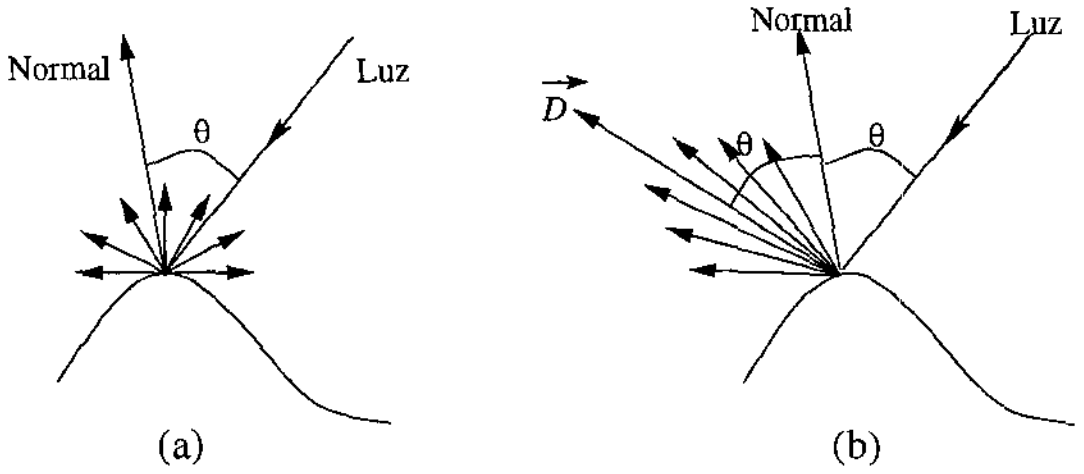



Figura 2.15: (a) Reflexão difusa. (b) Reflexão especular.

**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 direção, mas varia com o cosseno do ângulo $\theta$ entre a direção do raio de luz incidente e a direção do vetor normal, estimado neste voxel (Figura 2.15a)

**Reflexão Especular:** Parcela da intensidade de luz, refletida em um voxel, que é máxima na direção $D$ de reflexão, que forma um ângulo $\theta$ com o vetor normal, no lado oposto ao da luz incidente, e vai diminuindo com o afastamento em relação à direção $D$, de acordo com alguma regra (Figura 2.15b). Esta intensidade varia com o cosseno do ângulo $\alpha$, entre a direção $D$ e a direção do observador, elevado a um expoente $n$, relativo ao tipo de tecido. Como a direção do observador e da fonte de luz são iguais, $\alpha = 2\theta$.

**Influência da 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:

$$I(u,v) = I_{max}K_a + I_{dist}(u,v)(K_d cos\theta + K_s cos^n\alpha) \tag{2.8}$$

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 coeficiênte de reflexão para a luz ambiente;
$K_d$ é o coeficiênte de reflexão difusa;
$K_s$ é o coeficiênte de reflexão especular;
$I_{dist}(u, v)$ é o valor da tonalização baseada em distância, calculado através da (equação 2.7);
$\theta$ é o ângulo entre o vetor normal à superfície e o raio de incidência. Devem ser considerados os valores de $\theta$ entre 0 e 90 graus para a componente difusa, e valores entre 0 e 45 graus para a componente especular.
$\alpha$ 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 $\alpha$ é igual a $2\theta$.

Os valores dos parâmetros da equação 2.8 são escolhidos pelo usuário de forma empírica, até a obtenção de um resultado satisfatório. Os coeficientes de relfexão $(K_d, K_s, K_a)$ 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 $\alpha$ e $\theta$ são obtidos pelo produto interno entre os vetores normalizados das respectivas direções.

A direção normal em cada voxel pode ser estimada de diversas formas: usando o gradiente na cena original, o *z-buffer*, o gradiente da cena de opacidades, ou o gradiente da cena binária filtrada. Devido à sua influência na qualidade da imagem, deve-se escolher com muito cuidado o método utilizado para estimar as normais. Essa escolha depende também do tipo de técnica utilizada na captura das imagens e da constituição da estrutura que se deseja visualizar (e.g. o gradiente na cena original é recomendado para tecidos ósseos em CT, mas não para estes mesmos tecidos em MRI).

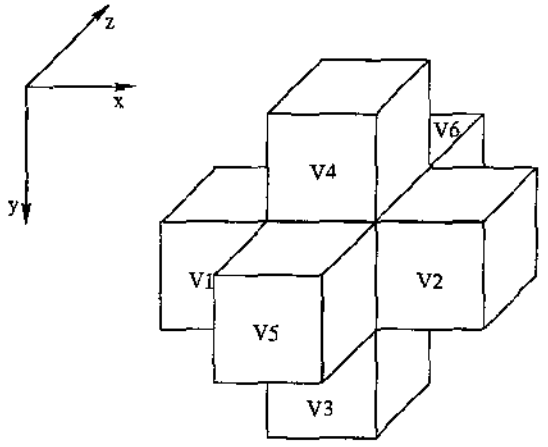



Figura 2.16: Vizinhança-6, utilizada no cálculo das normais.

A Figura 2.16 ilustra uma vizinhança do tipo 6 de voxels, para o cálculo das normais. Esse cálculo pode ser feito da seguinte forma:

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

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

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

$$N_{(x,y,z)} = \frac{(N_x, N_y, N_z)}{||(N_x, N_y, N_z)||} \tag{2.9}$$

onde:
$N_x, N_y, N_z$ são as componentes do vetor normal , estimado no voxel ($x$, $y$, $z$);
$f(x,y,z)$ pode ser o tom de cinza na cena original, a opacidade, o valor filtrado da cena binária, etc.;
$N_{(x,y,z)}$ é o vetor gradiente, normalizado, no voxel ($x$, $y$, $z$).

### 2.4.4 Conclusões

A eficiência relativa das técnicas de *rendering* de superfícies e volumes é objeto de ativa investigação [15, 54]. Enquanto as duas metodologias compartilham de fraquezas comuns, atribuídas a falhas nos processos de segmentação e classificação, o *rendering* de volumes parece ser uma escolha mais natural em situações onde uma segmentação correta e sem equívocos não é possível, mas ainda pode ser feita mais realisticamente através da atribuição de valores aos voxels de uma forma nebulosa. Essa flexibilidade é a principal razão das pesquisas em torno de *rendering* de volumes e suas aplicações.

Em *rendering* de volumes, através de uma classificação adequada dos dados e do equacionamento do comportamento da luz transmitida através do volume, é possível a observação de diferentes tipos de tecidos. Além disso, este modelo oferece uma importante vantagem sobre as técnicas baseadas em superfície, substituindo a classificação binária (segmentação) e a etapa de extração e modelagem de superfície, pela classificação nebulosa.

Observa-se, entretanto, que a classificação binária é um caso particular da classificação nebulosa (ver Seção 2.3). Portanto, pode-se considerar que o *rendering* de superfícies é um caso particular do *rendering* de volumes, onde a opacidade dos voxels, pertencentes à superfície do objeto de interesse, é igual a 1. Nesse caso, a combinação das contribuições dos voxels, projetados sobre um determinado pixel $p_i$ (Equação 2.6), pode ser ignorada. Apenas o voxel mais próximo, sobre o raio de projeção, afeta o pixel $p_i$ (Figura 2.14).

Além dos aspectos relativos à visualização, vimos que aplicações típicas em computação de imagens médicas também requerem operações de manipulação (e.g. rotação, escalamento, cortes, etc.) e análise (e.g. medidas de volume, área, distâncias, movimento relativo entre estruturas, etc.) de estruturas, de forma **interativa** com o usuário.

O *rendering* de volumes, no presente estado, ainda está confinado apenas a aplicações de visualização, pois possuem algumas desvantages que limitam significantemente seu poder e utilização. Entre essas desvantagens estão os altos custos de armazenamento e tempo computacional. Existe um considerável interesse, incluindo interesses comerciais, na comunidade de visualização, em resolver as questões relacionadas à velocidade.

Sabe-se que na prática a classificação nebulosa deve associar opacidades mais altas aos voxels que estão nas regiões de interface entre as estruturas. A razão para este procedimento é possibilitar a visualização clara das estruturas contidas na cena, pois se os voxels do interior tiverem opacidades altas, as estruturas mais internas não seriam vistas. Esta característica conduz ao resultado de que 70 - 95% dos voxels em uma cena são, normalmente, classificados como transparentes, e os que não são transparentes estão localizados nas fronteiras das estruturas. Portanto, a idéia principal do trabalho de Udupa e Odhner [29] foi criar uma estrutura de dados especial, chamada *Shell*, que codifica apenas estes voxels de interesse para visualização.

# Capítulo 3

# *Shell Rendering*

Este capítulo apresenta o método *shell rendering* , como proposto por Udupa e Odhner. Esta técnica tem mostrado ser uma solução viável, tanto para questões relativas a velocidade quanto questões relativas ao armazenamento, conseguindo um desempenho substancialmente melhor que outros algoritmos previamente publicados [29]. O *shell* é uma estrutura de dados especial, que armazena apenas os voxels localizados dentro de uma certa vizinhança da borda da estrutura, ao invés de manter as informações de todo o volume da cena original. Assim, os voxels que são armazenados e realmente são usados nos cálculos, são apenas aqueles que potencialmente contribuem no processo de *rendering*. Essa estrutura permite ainda flexibilidade de acesso aos voxels e a seus atributos, possibilitando uma redução dramática nos requerimentos computacionais.

As diferenças fundamentais, entre o método de *shell rendering* e o *rendering* de volumes tradicional (descrito na seção anterior), estão na representação da cena 3D e no algoritmo de projeção (Figura 1.1c e d, respectivamente). No caso do *rendering* tradicional, todos os voxels da cena são armazenados em uma matriz, e o algoritmo de projeção deve possibilitar o acesso aos elementos desta matriz, respeitando a metodologia adotada (*image-order* ou *object-order*). Já no caso do *shell rendering*, apenas a parcela dos voxels com significância para a visualização é armazenada na estrutura *Shell*. Portanto, o algoritmo de projeção deve se encarregar de realizar o acesso sobre esta estrutura específica. A estrutura *Shell*, o mecanismo de projeção e o algoritmo de *rendering* são descritos nas seções seguintes.

## 3.1 Estrutura Shell

Para uma dada cena semitransparente (cena de opacidades), o *shell* consiste de um conjunto de voxels com valores de opacidade diferentes de 0, em uma vizinhança da borda da estrutura, junto com um número de atributos associados aos voxels nesse conjunto. Em *rendering* de superfícies, a espessura do *shell* torna-se igual a 1 voxel, contendo voxels

cujo valor de opacidade é igual a 1.

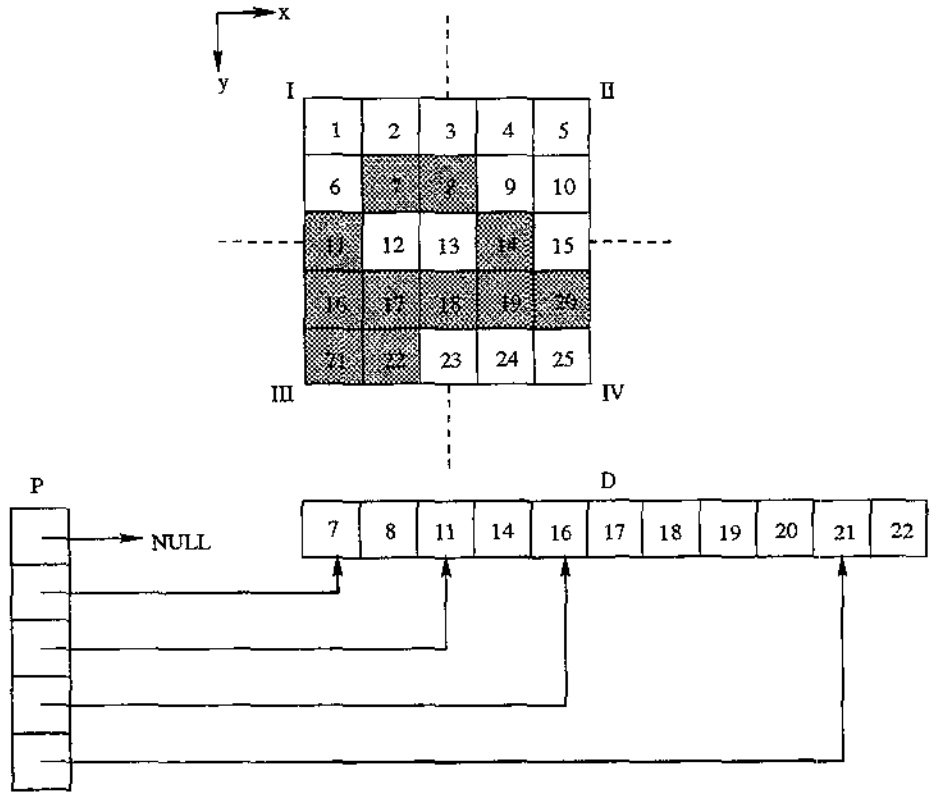



Figura 3.1: Exemplo de um *shell*.

A estrutura de dados *shell* é composta por uma lista indexada $D$ e uma matriz de ponteiros $P$. A lista $D$ é uma estrutura que contém um conjunto de atributos de visualização associados aos voxels do *shell*. Os voxels do *shell* são armazenados em $D$ a partir de (0, 0, 0), seguindo na direção positiva de $(x, y, z)$ em uma ordem coluna por coluna, linha por linha, fatia por fatia, dos voxels pertencentes à cena. A matriz de ponteiros $P$ tem dimensões $m \times n$, onde $m$ é o número de linhas em uma fatia (eixo $y$ da Figura 2.10) e $n$ é o número de fatias na cena (eixo $z$ da Figura 2.10). Cada elemento $(y, z)$ de $P$ é um ponteiro para o primeiro elemento do *shell* (opacidade diferente de 0) armazenado em $D$, que pertence à linha $y$ da fatia $z$.

Uma ilustração da forma de armazenamento do *shell*, para um caso bidimensional, pode ser visto na Figura 3.1. 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 do *shell*, com a matriz de ponteiros $P$ e a lista $D$ ajustadas para esse exemplo. Nota-se que, a estrutura de dados *shell* é genérica o suficiente para armazenar, se necessário, toda a cena 3D, ou apenas uma camada de 1 voxel de espessura, na superfície da estrutura de interesse.

Os atributos armazenados em $D$, associados com cada voxel são: a coordenada $x$ do voxel, o valor de sua opacidade, uma estimativa do vetor normal à superfície que passa pelo voxel. Outras informações úteis para manipulação e aceleração da visualização de estruturas também podem ser armazenadas. Porém, a escolha dessas informações depende principalmente das necessidades da aplicação. Uma análise mais profunda sobre a estrutura *shell* é apresentada em [29].

No presente trabalho, optou-se pela configuração da Figura 3.2b, para a estrutura *shell* usada em projeção ortogonal. Essa ilustração mostra ainda a associação entre as coordenadas $(y, z)$ da cena semitransparente e as dimensões $(m, n)$ da matriz $P$.

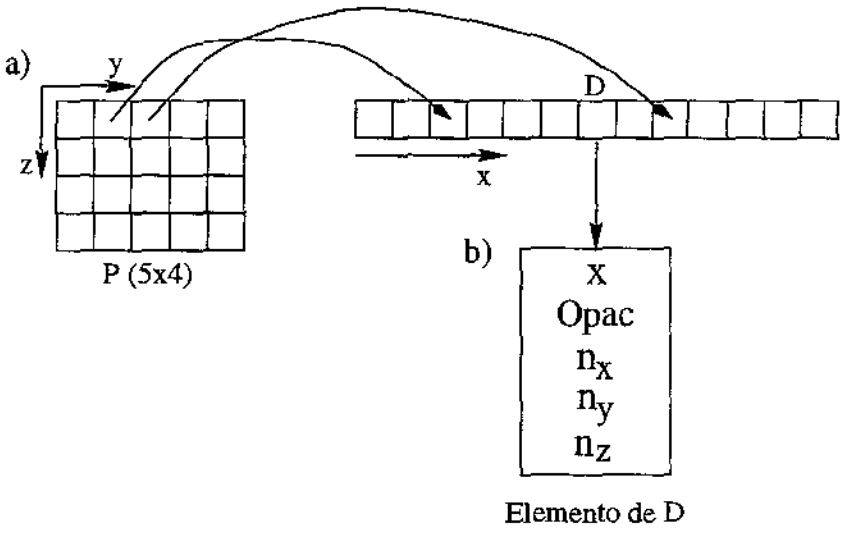



Figura 3.2: Configuração da estrutura *shell* utilizada

A lista $D$ é uma lista indexada onde cada elemento possui as seguintes informações (Figura 3.2b):

- $x$: valor da coordenada $x$ do voxel na cena;
- *Opac*: valor da opacidade do voxel, atribuído no processo de classificação;
- $n_x$, $n_y$, $n_z$: valores das componentes ($x$, $y$, $z$) do vetor normal estimado no centro do voxel;

Como mencionado anteriormente, a escolha dos atributos armazenados na lista $D$, depende das necessidades da aplicação. Por exemplo, em aplicações que possuam operações de manipulação, um atributo *Visível* poderia ser útil na medida em que serve para indicar quais voxels permanecem visíveis após cada interação (e.g. um corte).

A estrutura *shell* apresenta características únicas, que demonstram e fundamentam sua eficiência. Entre essas características, pode-se citar as seguintes:

- Em situações típicas de *rendering* de volumes cerca de 70% a 95% dos voxels classificados têm opacidade nula. Isto significa que eles não contribuem para formação da imagem no plano de visualização e não existe razão para armazená-los e considerá-los no *rendering* . A estrutura *shell* explora este fato e armazena apenas os voxels com opacidade diferente de zero.

- A estrutura *shell* explora as características geométricas da cena, quando armazena os voxels de forma a agilizar o acesso FTB durante o *rendering* .

- O *rendering* de volumes com projeção ortogonal baseado nesta estrutura pode ser considerado um dos métodos mais rápidos na literatura de visualização volumétrica.

## 3.2 Projeção Ortogonal usando o *Shell*

Em projeção ortogonal assume-se o observador no infinito e raios de projeção paralelos e ortogonais ao plano de visualização, portanto as estruturas aparecem sem escalamento.

Como já havia sido mencionado na Seção 2.4.2, a identificação dos pontos para projeção pode ser feita de duas maneiras: (a) usando-se técnicas de *ray-casting*, ou (b) técnicas de *voxels projection*.

Em *shell rendering* , a técnica adotada para projeção deve ser *voxels projection*. 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 original, são armazenados. Esta característica torna o emprego da técnica de *ray-casting* computacionalmente inviável. Nessa situação, a tarefa de se localizar no *shell*, cada voxel atravessado por um determinado raio de busca, é extremamente complexa.

Dentre as técnicas de remoção, disponíveis para *voxels projection*, optou-se em utilizar a técnica FTB, pois ela realiza a remoção de estruturas escondidas sem precisar classificar distâncias e sem precisar rastrear todos os voxels da cena.

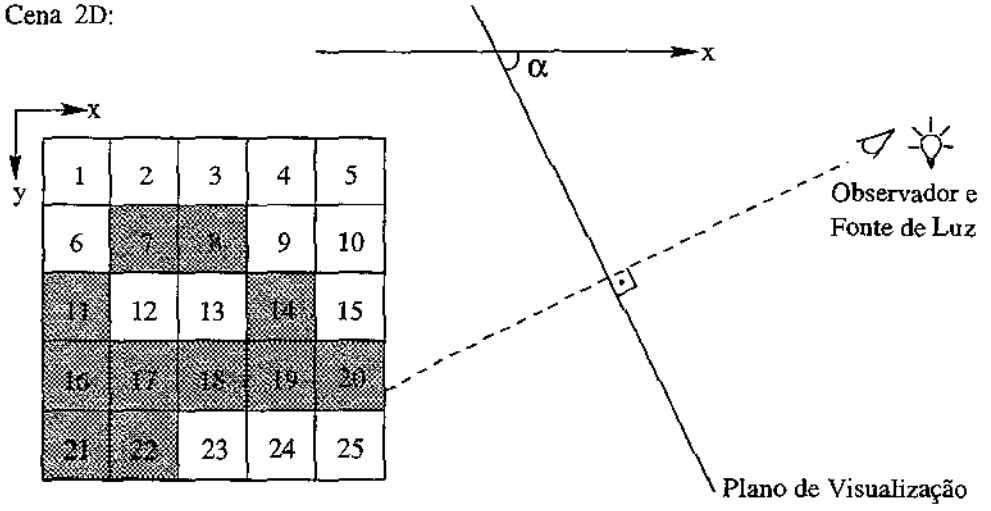



Figura 3.3: Projeção em *Shell Rendering* - Exemplo em duas dimensões.

A projeção ortogonal usando a estrutura *shell* é ilustrada com o seguinte exemplo 2D (Figura 3.3). Seja uma cena 2D semitransparente, constituída de 25 pixels, onde aqueles que aparecem sombreados representam o *shell* (como na Figura 3.1).

Na projeção de voxels usando FTB é necessário determinar, para uma dada direção de visualização, qual é a ordem correta para rastrear os elementos da cena . A técnica FTB requer que os voxels sejam acessados do mais próximo para o mais distante, em relação ao

plano de visualização. Isso implica em definir uma estratégia que possibilite determinar a ordem correta de acesso, como uma função dos ângulos $\alpha$ e $\beta$ de rotação da cena.

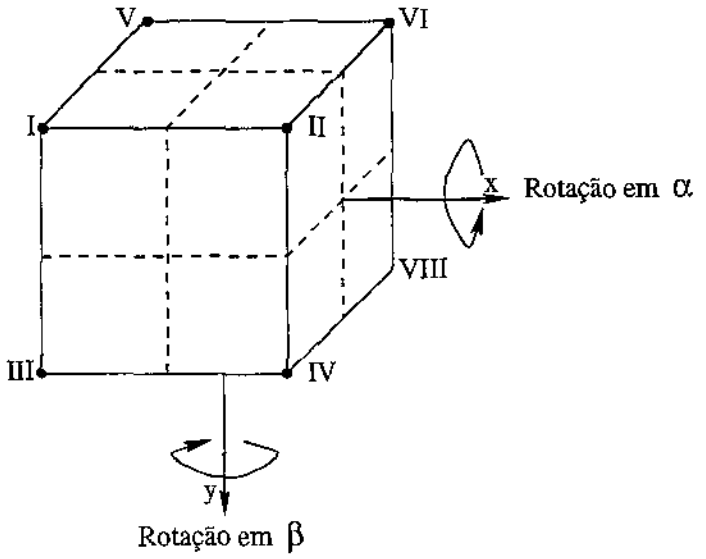



Figura 3.4: Casos para acesso FTB no espaço 3D, com projeção ortogonal.

No caso do exemplo descrito acima, a ordem de acesso FTB aos voxels da estrutura *shell*, para projeção ortogonal, pode ser facilmente identificada para qualquer direção de visualização. Para representar a direção de visualização, considera-se o ângulo $\alpha$ entre o plano de visualização e o eixo x (Figura 3.3). Baseado no valor de $\alpha$, pode-se identificar quatro casos de rastreamento FTB, para o exemplo bidimensional. A lista abaixo apresenta, para cada uma das quatro regiões delimitadas pelos intervalos de 90 graus do ângulo $\alpha$, a ordem FTB associada:

1) Para $0^0 \leq \alpha \leq 90^0$.

FTB: 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 15, 14, 13, 12, 11, 20, 19, 18, 17, 16, 25, 24, 23, 22, 21.

2) Para $90^0 < \alpha \leq 180^0$.

FTB: 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

3) Para $180^0 < \alpha \leq 270^0$.

FTB: 21, 22, 23, 24, 25, 16, 17, 18, 19, 20, 11 ,12, 13, 14, 15, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5.

4) Para $270^0 < \alpha < 360^0$.

FTB: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25.

No caso 3D é fácil perceber que, para intervalos de 90 graus dos ângulos $\alpha$ e $\beta$, existirão oito ordens de acesso FTB aos voxels da cena. Cada ordem de acesso está associada com uma das regiões, delimitadas pelos intervalos dos ângulos $\alpha$ e $\beta$, ou mais epecificamente a um dos octantes numerados na Figura 3.4.

Cada octante numerado representa uma ordem distinta para o acesso FTB. Se, por exemplo, o observador estiver no octante II (esse posicionamento pode ser obtido com $0^0 \leq \alpha < 90^0$ e $0^0 \leq \beta < 90^0$), a varredura deve ser feita de forma que o acesso aos voxels seja iniciado no vétice II e vá gradativamente se afastando, até atingir o vértice oposto VII.

Pode-se notar que a estrutura *shell* suporta facilmente todos os casos de acesso FTB com projeção ortogonal, também para o espaço 3D. Tomando por referência a ilustração da Figura 3.2, e lembrando sempre que o acesso relacionado a um determinado vértice é feito no sentido de se afastar desse vértice e se aproximar do vértice oposto, o rastreamento da estrutura *shell*, para cada caso mostrado na Figura 3.4, é feito segundo o código da Tabela 3.1. A notação $x^- \rightarrow x^+, y^- \rightarrow y^+, z^- \rightarrow z^+$ indica que os voxels devem ser indexados da menor coordenada para a maior coordenada, em cada eixo $x, y$ e $z$, respectivamente. É importante notar que, na projeção ortogonal, não existe nenhuma restrição quanto à ordem de precedência entre os eixos $x$, $y$ e $z$, diferentemente do que ocorre na projeção em perspectiva, como será visto no Capítulo 4.

| Caso de Acesso | Vétice Inicial | Vértice Final | Sentido de indexação dos voxels |
|---|---|---|---|
| 1 | I | VIII | $x^- \rightarrow x^+, y^- \rightarrow y^+, z^- \rightarrow z^+$ |
| 2 | II | VII | $x^- \rightarrow x^+, y^- \rightarrow y^+, z^+ \rightarrow z^-$ |
| 3 | III | VI | $x^- \rightarrow x^+, y^+ \rightarrow y^-, z^- \rightarrow z^+$ |
| 4 | IV | V | $x^+ \rightarrow x^-, y^- \rightarrow y^+, z^- \rightarrow z^+$ |
| 5 | V | IV | $x^- \rightarrow x^+, y^+ \rightarrow y^-, z^+ \rightarrow z^-$ |
| 6 | VI | III | $x^+ \rightarrow x^-, y^- \rightarrow y^+, z^+ \rightarrow z^-$ |
| 7 | VII | II | $x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^- \rightarrow z^+$ |
| 8 | VIII | I | $x^+ \rightarrow x^-, y^+ \rightarrow y^-, z^+ \rightarrow z^-$ |

Tabela 3.1: Esta tabela mostra o sentido de indexação dos voxels para cada um dos oito casos de acesso, onde a notação $x^- \rightarrow x^+, y^- \rightarrow y^+, z^- \rightarrow z^+$ indica que os voxels devem ser indexados da menor coordenada para a maior coordenada, em cada eixo $x, y$ e $z$, respectivamente, para uma projeção FTB correta.

Uma vez definida a regra de acesso aos voxels do *shell*, conforme o descrito na Tabela 3.1, os voxels são visitados um a um. Para cada voxel visitado, a matriz de rotação $M$ é aplicada, convertendo sua coordenada $(x, y, z)$ original para $(x', y', z')$, segundo a Equação 2.3. A projeção ortogonal deste voxel é obtida considerando $x' = u$ e $y' = v$, onde $u, v$ define o plano de visualização (Figura 2.10).

## 3.3 Algoritmo *Shell Rendering* - SR

Esta seção descreve o algoritmo SR, que implementa o *shell rendering* com projeção ortogonal. De acordo com a Tabela 3.1, as informações relevantes para percorrer os voxels do *shell*, para um dado octante, são:

- $y_i$ : coordenada $y$ do primeiro voxel a ser acessado no *shell*;
- $y_f$ : coordenada $y$ do último voxel a ser acessado no *shell*;
- $z_i$ : coordenada $z$ do primeiro voxel a ser acessado no *shell*;
- $z_f$ : coordenada $z$ do último voxel a ser acessado no *shell*;
- $d_x$ : indica o sentido de varredura, na direção do eixo $x$. Se $d_x = +1$, o acesso é feito no sentido crescente das coordenadas de $x$, e se $d_x = -1$, o acesso é feito no sentido decrescente;
- $d_y$ e $d_z$ : têm a mesma função de $d_x$, mas em relação aos eixos $y$ e $z$, respectivamente.

**Exemplo:** considerando a cena 2D, apresentada na Figura 3.1, e assumindo a direção de visualização no quadrante III, temos que a regra de indexação para os voxels é a seguinte: $x^- \rightarrow x^+, y^+ \rightarrow y^-$. Neste caso, as informações para o acesso assumiriam os seguintes valores:

$$y_i = 21;$$
$$y_f = 1;$$
$$d_x = +1;$$
$$d_y = -1.$$

Observa-se que não existem informações sobre as posições inicial e final para o acesso em $x$, apenas o sentido de varredura. Isto se deve ao fato de que, no *shell*, o acesso a $x$ é feito, de forma sequencial, sobre a lista $D$. Portanto, cada linha possui seus próprios índices inicial e final, dentro de $D$.

Estas informações, necessárias para o acesso ao *shell*, são armazenadas numa estrutura de dados como a apresentada na Figura 3.5. Esta estrutura é formada por um vetor de oito posições, indexado pelo número do octante de onde parte a direção de visualização.

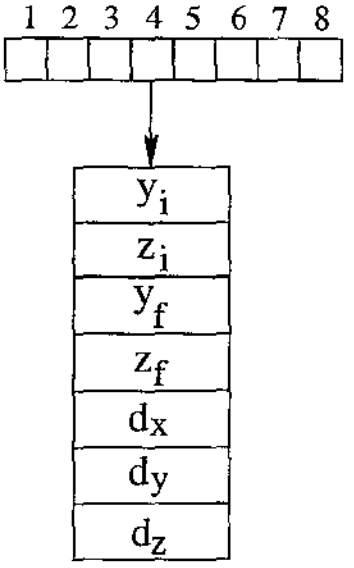



Figura 3.5: Lista de acesso e seus elementos.

Como cada octante está associado a uma determinada regra de indexação, cada posição do vetor contém um registro que armazena as informações usadas na execução dessa regra específica.

O algoritmo SR é apresentado a seguir, junto com a especificação dos dados de entrada e saída, estruturas auxiliares, e comentários.

**Entrada:**

- lista $D$: vetor de registros, componente da estrutura *shell*, onde os atributos de visualização dos voxels são armazenados. A lista $D$ e estes atributos são mostrados na Figura 3.2b;
- matriz $P$: matriz de ponteiros, também componente do *shell*, usada no acesso aos elementos da lista $D$. Esta matriz mantém o registro das coordenadas $y$ e $z$ de cada linha da cena (Figura 3.2a);
- ângulos $\alpha$ e $\beta$: ângulos que definem a rotação da cena em torno dos eixos $x$ e $y$ (Figura2.10), respectivamente. Estes dados são informados pelo usuário;
- $Ph$: estrutura de dados contendo as componentes $(K_s, K_d, K_a)$, o coeficiente $n$ e o valor $I_{max}$ (intensidade máxima desejada na imagem), utilizados pelo modelo de

iluminação de Phong. A descrição destes itens é apresentada na Seção 2.4.3. Estes dados podem ser informados pelo usuário ou assumirem valores pré-definidos.

**Saída:**

- matriz *Pixels*: esta matriz é usada para representar a imagem que será formada pelo *rendering* . Cada posição da matriz, que representa um pixel da imagem de saída, armazena o valor de tonalização, calculado para esse pixel.

**Estruturas Auxiliares:**

- $LA[i]$: lista indexada pelos octantes, contendo informações para a execução do acesso à estrutura *shell* (Figura 3.5);
- matriz *Opac*: esta matriz possui a mesma dimensão de *Pixels*, e seus elementos também estão associados a aos pixels da imagem de saída. Porém, esta matriz é usada no controle de saturação, e armazena os valores de opacidade acumulada em cada pixel;
- *oct*: índice para $LA$ que indica o octante correspondente à direção de visualização dada por $\alpha$ e $\beta$;
- *prim*: endereço do primeiro voxel do *shell*, pertencente a uma determinada linha, dentro da lista $D$;
- *ult*: endereço do último voxel do *shell*, pertencente a uma determinada linha, dentro da lista $D$;
- *ind*: índice para o acesso aos elementos da lista $D$;
- $y$: índice para a matriz de ponteiros $P$, representando a coordenada $y$;
- $z$: índice para a matriz de ponteiros $P$, representando a coordenada $z$;
- $(x', y', z')$: coordenada do voxel $(x, y, z)$, após a aplicação da matriz $M$;
- $(n'_x, n'_y, n'_z)$: componentes do vetor normal ao voxel $(x, y, z)$, após a aplicação da matriz $M$;
- $(u, v)$: valores usados como índices para a matriz *Pixels*, representam as coordenadas de um pixel da imagem de saída no plano de visualização;
- *tonaliz*: armazena o valor de tonalização, calculado para o voxel corrente.

*Início*

I. Calcula matriz $M$ através dos ângulos $\alpha$ e $\beta$ (Equação 2.3);

II. Calcula o octante *oct*, através de $\alpha$ e $\beta$;

III. ***para*** $(z \leftarrow LA[oct].z_i$ ***até*** $LA[oct].z_f$ ***c/ incremento*** $LA[oct].d_z)$ ***faça***

1. ***para*** $(y \leftarrow LA[oct].y_i$ ***até*** $LA[oct].y_f$ ***c/ incremento*** $LA[oct].d_y)$ ***faça***

(a) ***se*** $(LA[oct].d_x > 0)$ ***então***

/* *Direção crescente de x* */

a. $prim \leftarrow P[y, z]$;

b. $ult \leftarrow P[y, z] + 1$; /* *Próximo ponteiro não nulo* */

***senão***

/* *Direção decrescente de x* */

c. $prim \leftarrow P[y, z] + 1$; /* *Próximo ponteiro não nulo* */

d. $ult \leftarrow P[y, z]$;

***fim_se***

(b) ***para*** $(ind \leftarrow prim$ ***até*** $ult$ ***c/ incremento*** $LA[oct].d_x)$ ***faça***

a. $(x', y', z') \leftarrow (D[ind].x, y, z) \times M$; /* *Aplica matriz de rotação* */

b. $(u, v) \leftarrow (x', y')$; /* *Projeção* */

c. ***se*** $Opac[u, v] < 1$ ***então***

/* *Se pixel* $(u, v)$ *não saturado* */

i. $(n'_x, n'_y, n'_z) \leftarrow (D[ind].n_x, D[ind].n_y, D[ind].n_z) \times M$;

ii. Calcula $I_{dist}$, através da Equação 2.7, usando $z'$ como $d(u, v)$;

iii. Calcula o ângulo $\theta$, entre a normal e a direção de visualização, usando $(n'_x, n'_y, n'_z)$;

iv. $tonaliz \leftarrow Ph.I_{max} \times Ph.K_a + I_{dist}(Ph.K_d cos\theta + Ph.K_s cos^n\theta)$; /* *Phong* */

v. $Pixels[u, v] \leftarrow Pixels[u, v] + (tonaliz \times D[ind].opac \times Opac[u, v])$; /* *Equação 2.6* */

vi. $Opac[u, v] \leftarrow Opac[u, v] + (1.0 - D[ind].opac)$;

***fim_se***

***fim_para***

***fim_para***

***fim_para***

*fim*

O algoritmo anterior descreve como é feito o acesso à estrutura *shell* e o processo de *rendering*, executado sobre esta estrutura. O *rendering* envolve, entre outras operações, a projeção dos voxels sobre o plano de visualização (item $III.(b).b.$ do algoritmo), o cálculo da tonalização de cada voxel individual (item $III.(b).c.iv.$), e a composição destes valores de tonalização, para a determinação da cor de cada pixel na imagem de saída (item $III.(b).c.v.$).

O algoritmo inicia com o cálculo da matriz de transformação $M$ (item $I.$), que é utilizada para rodar as coordenadas originais dos voxels, antes da projeção (item $III.(b).a.$), e rodar os vetores normais, para o cáculo de tonalização (item $III.(b).c.i$). Em seguida, o octante de onde parte a direção de visualização é determinado, mediante os ângulos $\alpha$ e $\beta$ (item $II.$). O número do octante escolhido é usado como índice para a lista $LA[i]$, cujos dados orientam o acesso de acordo com a regra de indexação pertinente a este octante.

Este algoritmo apresenta alguns pontos que merecem um esclarecimento mais detalhado. Como pode ser observado no item $III.(b).c.i.$, todas as componentes do vetor normal, associadas ao voxel corrente, são rodadas pela aplicação da matriz $M$. Esta operação é necessária para manter a coerência com a rotação dos voxels, pois o ângulo $\theta$, entre a normal e a direção de visualização, é utilizado no cálculo da tonalização. Porém, como no modelo adotado a direção de visualização está fixa sobre o eixo $z$, foi possível simplificar esse cálculo, rodando apenas a componente $n_z$. O ângulo $\theta$ é calculado pelo produto interno entre o vetor normal e a direção de visualização, mas já que a direção de visualização é descrita pelo vetor $(0, 0, 1)$, apenas a componente $n_z$ é usada neste cálculo.

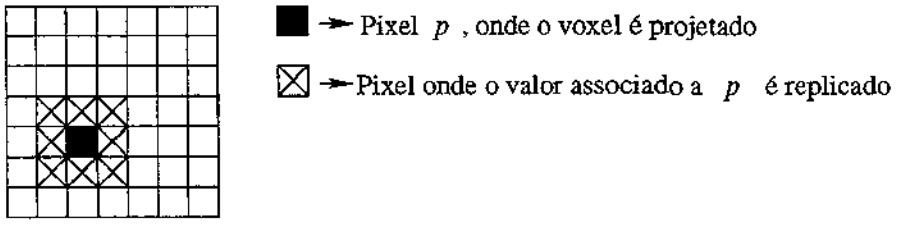



Figura 3.6: Vizinhança onde o valor projetado em $p$ é replicado.

O outro detalhe importante é o fato conhecido, e já mencionado, de que a projeção de voxels provoca o aparecimento de artefatos (buracos) na imagem final (ver Seção 2.4.2). A eliminação desses buracos pode ser feita por uma filtragem na imagem de saída, ou através de uma mudança nas proporções entre o tamanho de um voxel e o tamanho de um pixel, fazendo as dimensões do voxel maiores que a do pixel. O método usado neste trabalho foi o segundo, por apresentar uma qualidade melhor que a filtragem. A mudança de escala, entre um voxel e um pixel, pode ser representada através da replicação do valor de tonalização, destinado a um determinado pixel $p$, pelos pixels vizinhos (Figura 3.6). Ou seja, um mesmo voxel cobre mais de um pixel.

O outro método, baseado na filtragem da imagem de saída, foi inicialmente adotado para eliminação destes artefatos. Embora ligeiramente mais eficiente, a qualidade da imagem era sensivelmente pior. Outro fator importante na escolha do método de mudança de escala, foi manter a coerência com o algoritmo de *rendering* com projeção em perspectiva, que usa um recurso semelhante para representar os efeitos da distorção perspectiva (detalhes na Seção 4.3).

A comparação dos resultados, obtidos com os método de eliminação de buracos, pode ser observada na Figura 3.7. Em 3.7a, está a imagem criada sem o uso de nenhum método para eliminação de buracos. A imagem apresenta grande quantidade de artefatos. A Figura 3.7b, mostra a imagem que foi submetida a uma filtragem. Apesar da qualidade desta ser bem melhor que a da Figura 3.7a, ainda contém vários buracos, além de apresentar um aspecto borrado. A imagem mostrada em (c), foi criada com o uso do método de mudança de escala, e possui qualidade visivelmente superior: nítida e sem artefatos.

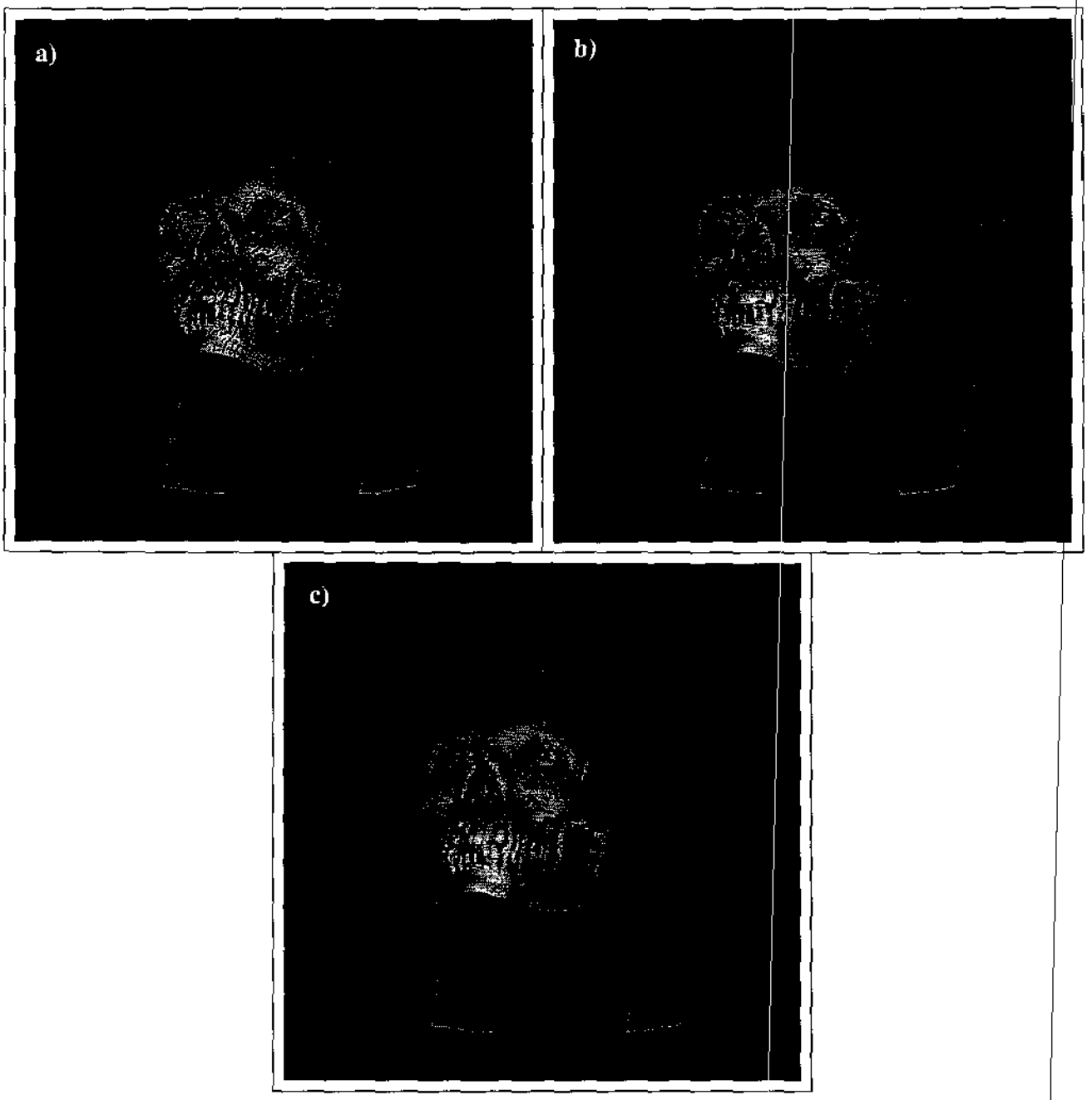



Figura 3.7: Crânio seco reconstruído de uma cena CT. (a) *Rendering* da imagem, sem aplicação de nenhum método para eliminação de buracos (b) *Rendering* com aplicação de filtro na imagem de saída. (c) *Rendering* usando replicação de pixels.

# Capítulo 4

# *Shell Rendering* em Perspectiva Digital

A razão para o desenvolvimento de um algoritmo de *rendering* com projeção em perspectiva é a possibilidade de viabilizar aplicações cuja implementação não seria possível, apenas com o emprego de métodos baseados em projeção ortogonal. Por exemplo, aplicações de endoscopia virtual.

Este trabalho procurou então uma solução viável, do ponto de vista de interatividade e qualidade da imagem, para o problema de projeção em perspectiva. Para atingir esse objetivo, através da utilização da estrutura *shell*, algumas dificuldades tiveram que ser tratadas. Estas dificuldades levaram ao desenvolvimento de uma nova estrutura *shell*, capaz de comportar as restrições impostas pelo novo algoritmo de projeção. Este capítulo aborda estes problemas e as soluções adotadas, descreve a nova estrutura *shell* e seu funcionamento, e apresenta o novo algoritmo de projeção, que utiliza o conceito de perspectiva digital.

## 4.1 Problemas encontrados

O *shell* é uma estrutura de dados especificamente desenvolvida para prover alta performance a algoritmos de visualização com projeção ortogonal. Esta estrutura armazena apenas os voxels que podem efetivamente ser vizualisados, e esses voxels estão localizados dentro de uma vizinhança da fronteira da estrutura de interesse. Essa característica é a principal responsável pela sua eficiência, uma vez que os cálculos de tonalização e transformações geométricas só serão aplicados em voxels que efetivamente contribuem para a formação da imagem de saída. Porém, este tipo de armazenamento resulta na compactação dos dados originais que, neste caso, é feita na direção do eixo $x$, como mostra a Figura 4.1.

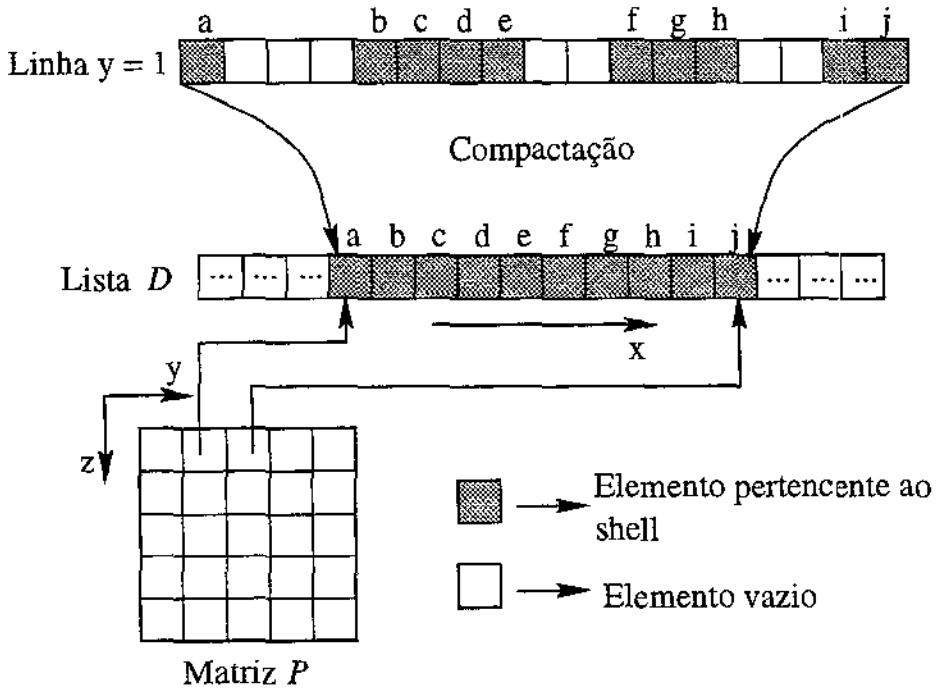



Figura 4.1: Compactação dos dados na direção $x$.

Toda a modelagem da estrutura do *shell* focaliza a forma como essa compactação é feita, e posteriormente como as informações sobre o posicionamento original são recuperadas. As diferentes formas de acesso ao *shell* representam as diferentes formas de acesso aos elementos da cena. Porém a forma como essa esturutura foi concebida, apesar de eficiente e totalmente adequada aos seus propósitos originais, oferece algumas restrições quanto ao modo de acesso às informações nela armazenadas.

Por causa desta compactação dos dados na lista $D$, perde-se o registro de coordenadas da cena original. Muito embora $D$ seja implementada como um vetor de registros, ela é uma lista de acesso sequencial. Ou seja, os voxels do *shell* são acessados sequencialmente ao longo das linhas, e o acesso a uma linha é indexado por $P$. Isto torna o acesso ao *shell* similar ao caso de uma lista ligada. Numa estrutura sequencial indexada, o acesso pode ser feito de forma pontual, ou seja, é possível visitar um elemento em um único passo, através do seu índice na lista. Este tipo de acesso é possível na cena original (Figura 4.2a). Entretanto, a lista $D$ não possui o registro da posição de seus elementos, então um elemento só pode ser localizado através de uma varredura sequencial dessa lista. A estrutura *shell* pode ser vista como um conjunto de listas ligadas, onde cada lista armazena os elementos, com opacidade diferente de zero, de uma determinada linha (Figura 4.2b).

É possível realizar o acesso pontual a uma determinada linha, porém seus elementos

só podem ser visitados sequencialmente. Isto dificulta o acesso onde a varredura é feita primeiro numa direção diferente da direção $x$, ao longo da qual é feita a compactação, e torna impraticável o acesso em uma direção arbitrária.

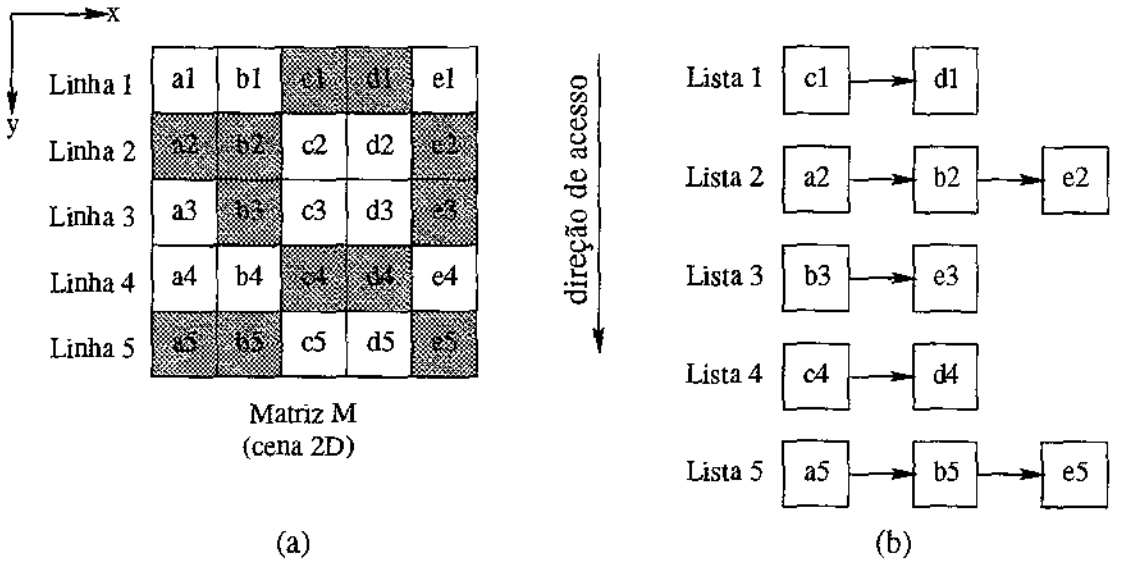



Figura 4.2: Compactação em $x$ e acesso em $y$. a) Cena 2D apresentada como uma matriz $C$, onde o acesso pode ser pontual. b) Armazenamento da mesma cena 2D, em forma de listas ligadas, onde o acesso deve ser sequencial.

Para ilustrar esta dificuldade, pode-se tomar o exemplo da Figura 4.2, no qual a compactação é feita na direção $x$ e o acesso é realizado primeiro na direção $y$. A matriz $C$, da Figura 4.2a, representa a cena 2D original e as listas da Figura 4.2b representam a estrutura *shell* que armazena essa cena.

Nota-se que o acesso na direção $y$ implica na necessidade de, para cada coluna de $C$, percorrer cada lista até que um elemento de coordenada $x$, igual à coluna corrente, seja encontrado (sucesso). Ou, caso a lista não possua elemento na coluna corrente, a busca deve prosseguir até que se encontre o primeiro elemento de coordenada $x$ maior que a coluna corrente, ou ainda até o final da lista (pior caso). Considerando por exemplo, o acesso feito sobre a coluna $e$, conclui-se que seria necessário visitar todos os elementos de todas as listas em 4.2b. As Listas 1 e 4 teriam todos os elementos visitados, mas nenhum seria processado, pois não possuem elemento na coluna $e$. Nas Listas 2 e 5 o acesso seria feito primeiro nas colunas $a$ e $b$, antes que $e2$ e $e5$ fossem localizados. Finalmente, a Lista 3 possui o elemento $e3$, que só pode ser acessado após $b3$.

Embora seja uma solução possível, torna-se impraticável à medida que o custo computacional desta operação anula qualquer ganho obtido com o uso do *shell*. Portanto, como a compactação é feita na direção do eixo $x$, a única forma eficiente de se realizar o acesso sobre a estrutura *shell* original, é varrer primeiro na direção de $x$.

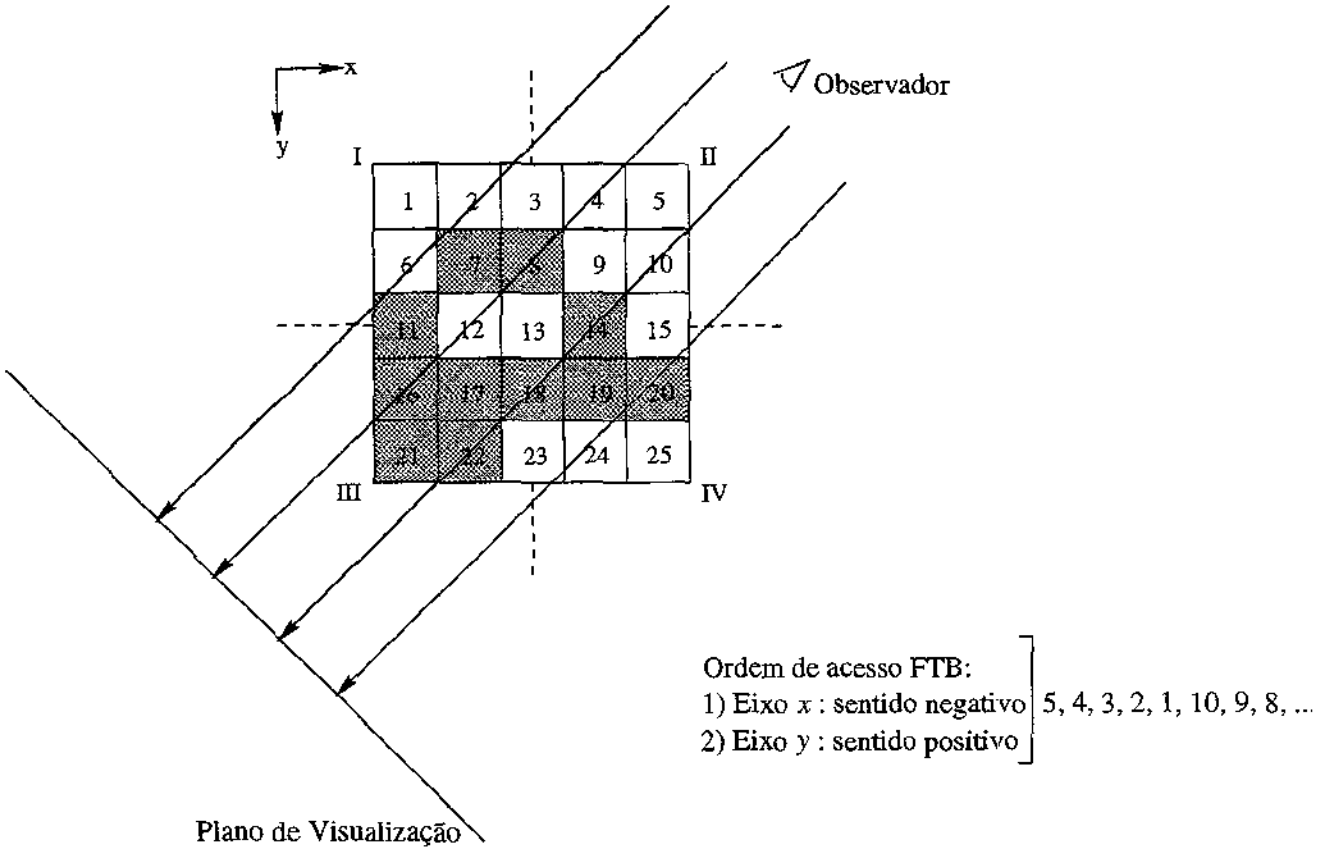



Figura 4.3: Acesso FTB, para o caso de projeção ortogonal.

No caso de projeção ortogonal, essa restrição não implica em nenhum tipo de inconveniente, pois o importante é garantir que a projeção dos voxels não viole a ordem FTB [1], para que a imagem de saída seja formada corretamente. Como na projeção ortogonal o observador encontra-se no infinito, todos os raios de projeção são paralelos entre si e perpendiculares ao plano de visualização (Figura 4.3). Isto permite que a projeção FTB seja preservada, independente da ordem de precedência escolhida para o acesso aos eixos. Portanto, para o caso de projeção ortogonal, o único fator que importa é a direção de indexação dos voxels em cada eixo (ver notação da Tabela 3.1).

Para qualquer direção de visualização, o acesso pode sempre ser feito primeiro em $x$, sem que isso viole a projeção FTB, em qualquer um dos oito casos de acesso mostrados na Tabela 3.1. Pode-se observar na Figura 4.3 que, varrendo-se $x$ antes de $y$, nenhum voxel mais distante do observador é projetado sobre um outro que está mais próximo. Isto é válido para qualquer posição do plano de visualização em torno da cena.

Apesar de que esta característica da estrutura *shell* não representa nenhum tipo de

[1] O que realmente importa, para um *rendering* correto, é que os voxels sejam **projetados** em ordem FTB, mesmo que o acesso não respeite essa ordem.

limitação, no caso de algoritmos para projeção ortogonal, tal discussão assume um aspecto bem mais importante quando o contexto é projeção em perspectiva.

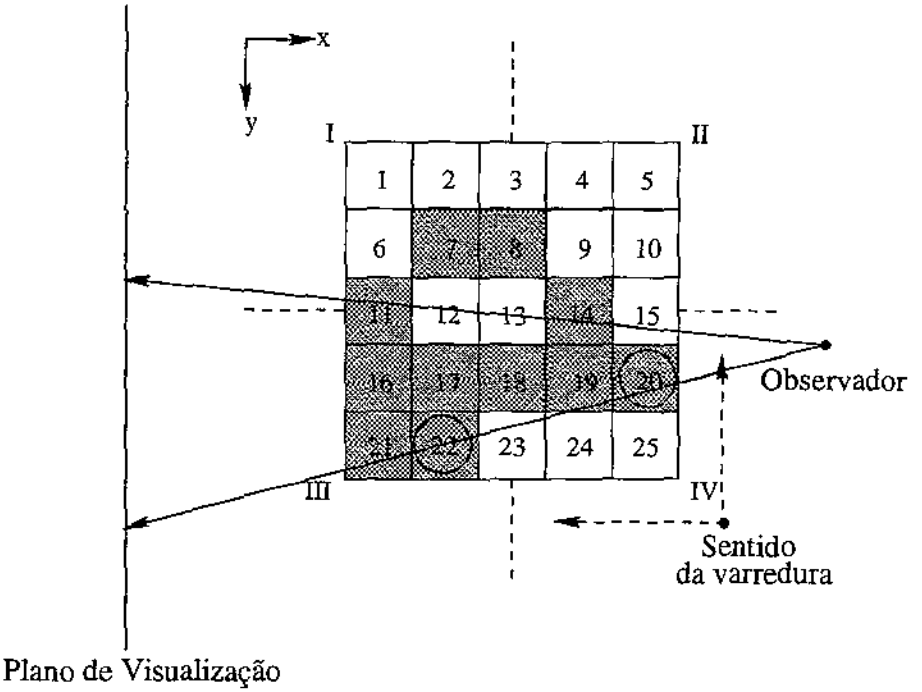



Figura 4.4: Problema da precedência entre $x$ e $y$, no acesso FTB. Quando o acesso em $x$ é feito primeiro, o voxel 22 é projetado antes que o voxel 20.

No modelo para projeção em perspectiva, o observador é colocado em uma posição finita e os raios de projeção são lançados a partir de sua localização (os detalhes serão apresentados na Seção 4.3). Portanto, os raios não são paralelos entre si, nem necessariamente perpendiculares ao plano de visualização. Logo, a situação apresentada na Figura 4.3 não se aplica a este caso.

A projeção em perspectiva pode ser vista conforme a ilustração apresentada na Figura 4.4. No exemplo apresentado nesta figura, o observador está localizado numa posição finita do quadrante IV. A projeção FTB determina uma ordem de varredura que se afasta do vértice mais próximo ao observador. Portanto, no presente exemplo, a varredura deve ser iniciada no vértice IV e seguir em direção ao vértice I. Entretanto, neste caso, a questão da precedência de acesso entre os eixos $x$ e $y$ é importante, pois determina se a projeção FTB será ou não violada.

Se o acesso for realizado primeiro em $x$, de acordo com a restrição imposta pela estrutura *shell*, o voxel 22 será projetado antes que o voxel 20. Como esses dois voxels são projetados sobre o mesmo pixel, e 20 está mais próximo do observador que 22, conclui-se que esta projeção não respeita a ordem FTB. O voxel 20 deve ser projetado primeiro, mas para isso o acesso ao *shell* deve ser feito primeiro na direção $y$.

## 4.2 *Shell* extendida

Pela forma como a estrutura *shell* foi definida, a maneira mais rápida de acessar seus voxels para projeção, é partindo do voxel mais próximo de um dos oito vértices da cena, e varrendo os outros voxels ao longo dos eixos $x, y, z$. Portanto, a princípio, a projeção FTB em perspectiva deveria levar em conta a posição do observador, o voxel do *shell* mais próximo desta posição deveria ser identificado e o acesso aos outros voxels deveria ser feito a partir dele, em direção ao mais afastado.

Em projeção ortogonal, isto foi possível apenas determinando a direção de indexação para cada eixo (ver Tabela 3.1, em função da direção de visualização. Nesta seção, mostramos que também é possivel definir o acesso FTB, para projeção em perspectiva, sem usar a posição do observador. Isto é, levando em conta apenas a direção de visualização, como se o observador estivesse no infinito. Só que neste caso torna-se necessário definir, também em função da direção de visualização, uma ordem de precedência entre os eixos.

A solução adotada consiste em determinar primeiro a correta ordem de indexação por octante, exatamente como nos casos associados à Figura 3.4 (Seção 3.2). Em seguida, determina-se a ordem de precedência entre $x$, $y$ e $z$. Isso é feito com base num modelo que define seis pirâmides infinitas $P_1, P_2, ..., P_6$, cujos vértices superiores coincidem com o centro da cena 3D, e as bases são paralelas às faces dessa cena. A direção de visualização indica, portanto, em qual dessas seis pirâmides o observador está localizado.

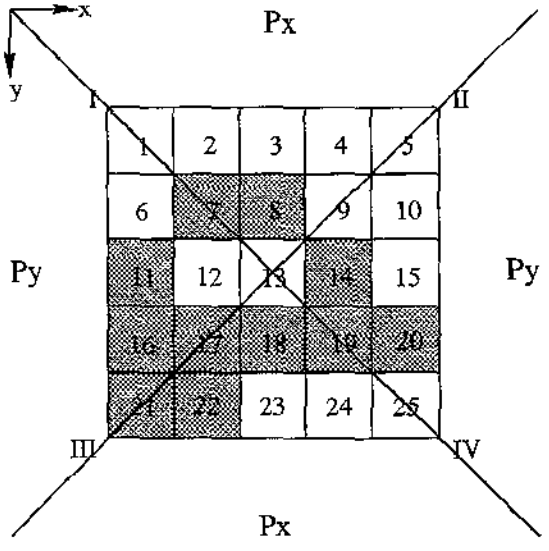



Figura 4.5: Pirâmides $P_x$ e $P_y$ para o caso 2D.

Para uma dada pirâmide, a ordem de precedência é tal que a varredura é feita primeiro nos eixos paralelos à base da pirâmide (em qualquer ordem), antes de ir ao longo do eixo perpendicular a ela. Como essas seis pirâmides possuem bases paralelas duas a duas, o

conjunto inicial de seis pirâmides $P_1, P_2, ..., P_6$ pode ser reduzido para três, onde duas pirâmides $P_n$ e $P_m$, cujas base são paralelas, são agrupada num único caso. Então, as duas pirâmides com bases paralelas ao plano $xy$ são agrupadas sob o nome de $P_{xy}$, e de maneira análoga, as pirâmides paralelas aos planos $yz$ e $xz$ são agrupadas como $P_{yz}$ e $P_{xz}$, respectivamente. A Tabela 4.1 mostra a ordem de precedência entre os eixos $(x, y, z)$, para cada um do casos representados pelas três novas pirâmides.

| Pirâmides | Ordem de Precedência entre os eixos |
|---|---|
| $P_{xy}$ | $xyz$ |
| $P_{yz}$ | $zyx$ |
| $P_{xz}$ | $xzy$ |

Tabela 4.1: Esta tabela mostra a ordem de precedência entre $x$, $y$ e $z$, para cada pirâmide $P_{xy}, P_{yz}$ e $P_{xz}$, onde a notação $xyz$ indica que a varredura deve ser feita ao longo de $x$ e $y$ antes de $z$. Com esses três casos de ordenação é possível realizar o acesso FTB, dos elementos de uma cena 3D, para qualquer direção de visualização.

Para ilustrar esta idéia, a Figura 4.5 mostra as duas pirâmides $P_x$ e $P_y$, definidas para o caso 2D. Neste exemplo, a ordem de precedência é determinada pela seguinte tabela:

| Pirâmides (2D) | Ordem de Precedência entre os eixos |
|---|---|
| $P_x$ | $xy$ |
| $P_y$ | $yx$ |

Tabela 4.2: Tabela de ordem de precedência, para o caso 2D.

Considerando a mesma situação mostrada na Figura 4.4, onde o observador está no quadrante IV, temos que a direção de visualização indica a pirâmide $P_y$ e portanto a ordem de acesso aos eixos deve ser $yx$.

O modelo das pirâmides soluciona a questão de se identificar a ordem de precedência entre os eixos, em função da direção de visualização. Entretanto, já foi demonstrado que o acesso realizado sobre o *shell*, cuja ordem de precedência não é iniciada pelo eixo $x$, é extremamente ineficiente. Isto torna-se um obstáculo quando a direção de visualização está em $P_{yz}$ (ver Tabela 4.1).

Para resolver esse problema, a estrutura de dados *shell* foi extendida de maneira a comportar a ordem de acesso $zyx$, da mesma forma que comporta os outros dois casos, sem

comprometer o desempenho. Essa extensão consiste em uma nova matriz de ponteiros $P'$ e outra lista indexada $D'$ (Figura 4.6b) . A idéia é usar $D'$ para codificar os voxels da cena na direção do eixo $z$, assim como é feito com $D$ na direção de $x$. Em $D'$ são armazenadas as coordenadas $z$ de cada voxel do *shell* e um ponteiro para o voxel correspondente em $D$. Esse ponteiro serve para evitar o armazenamento desnecessário dos atributos, associados a cada voxel, também na lista $D'$ (ver Figura 3.2). De maneira similar à matriz $P$, cada ponteiro em $P'$ aponta para o primeiro voxel em $D'$, associado com uma coordenada particular $(x, y)$ da cena. Portanto, as estrutura $P'$ e $D'$ são usadas quando a direção de visualização está em $P_{yz}$. Caso contrário, as estruturas originais $P$ e $D$ (Figura 4.6a) são acionadas.

Esta modificação, na estrutura original do *shell*, permite que o acesso seja feito na ordem de precedência $zyx$ sem nenhuma redução no desempenho, em relação aos casos de acesso realizados sobre a estrutura original.

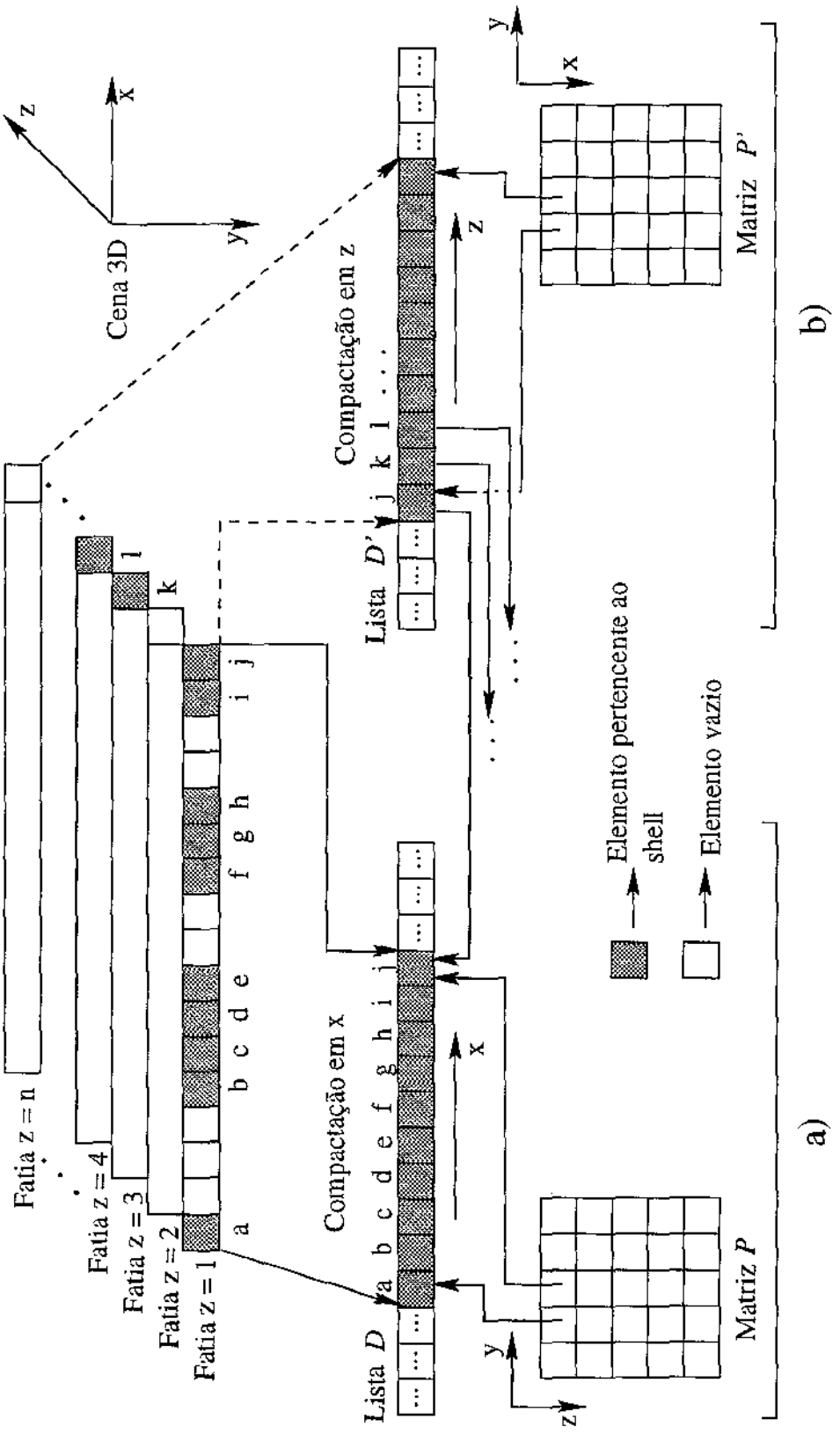



Figura 4.6: Ilustração da estrutura *shell* extendida, usada na projeção em perspectiva.

## 4.3 Projeção em perspectiva digital

A principal diferença, entre os modelos para projeção ortogonal e para projeção em perspectiva, está no fato de que no primeiro modelo assume-se que o observador está posicionado no infinito, então os raios de projeção são paralelos entre si, e chegam perpendiculares ao plano de visualização. O resultado disso é que as estruturas aparecem sem escalamento (Figura 4.7a). Já no segundo modelo, definido como *one-point perspective projection*, o observador encontra-se numa posição finita, portanto os raios de projeção não são paralelos. Isso faz com que as estruturas sofram efeitos de escalamento (Figura 4.7b).

(a) (b)

Figura 4.7: Imagens de um crânio, criadas através de: (a) Projeção ortogonal (b) Projeção em perspectiva.

Como vimos anteriormente, a projeção FTB em perspectiva requer que sejam definidas duas informações, em função da direção de visualização: (1) octante que define a direção de indexação dos voxels, ao longo dos eixos $x, y, z$, (2) pirâmide que define a ordem de precedência entre estes eixos.

Para acomodar a idéia da divisão da cena em pirâmides, usada na determinação da ordem de precedência entre os eixos (Tabela 4.1), e também para se conseguir um mapeamento, desta mesma cena, em regiões que permitam definir o sentido de varredura sobre os eixos (como o mapeamento baseado em octantes, ilustrado na Figura 3.4), a cena foi dividida em regiões delimitadas pela variação dos ângulos $\alpha$ e $\beta$ em intervalos de $45^0$.

Essa divisão permite a composição das pirâmides e dos octantes através de regiões

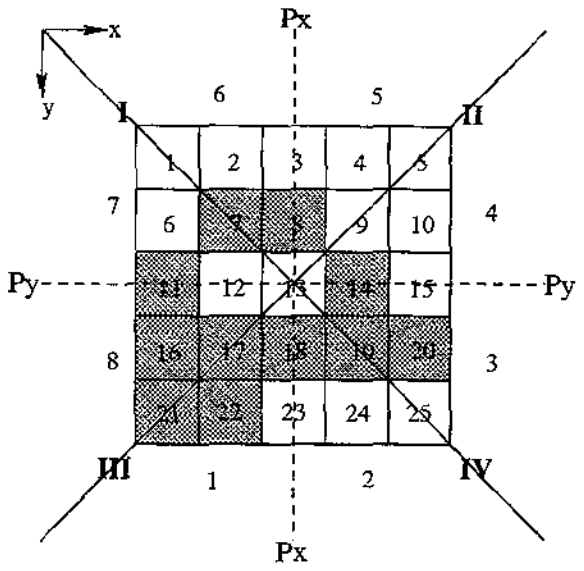



Figura 4.8: Divisão da cena em intervalos de $45^0$, para o caso 2D.

adjacentes. No exemplo bidimensional, ilustrado na figura 4.8, verifica-se que a pirâmide $P_x$ pode ser composta pelas regiões $(1, 2, 5, 6)$, e o octante $IV$ pode ser formado pelas regiões $(2, 3)$. Como a divisão da cena, usada para o caso de projeção ortogonal, é baseada em octantes, fica claro que trata-se de um subconjunto desta nova configuração, e portanto também pode ser obtida por composição. A divisão, em intervalos de $90^0$ do ângulo $\alpha$, usada no exemplo ilustrado pela Figura 3.3, também pode ser gerada pela união de regiões adjacentes:

1) $0^0 \leq \alpha \leq 90^0 \Rightarrow 5 \cup 4$

2) $90^0 < \alpha \leq 180^0 \Rightarrow 3 \cup 2$

3) $180^0 < \alpha \leq 270^0 \Rightarrow 8 \cup 1$

4) $270^0 < \alpha < 360^0 \Rightarrow 7 \cup 6$

Esta característica é desejável, pois permite que um mesmo modelo seja usado tanto na projeção ortogonal quanto na projeção em perspectiva.

Considerando o presente modelo, no qual os ângulos de rotação $\alpha$ e $\beta$ são tomados em intervalos de $45^0$, verifica-se que a cena será dividida em 64 regiões, delimitadas por esses intervalos. Para a determinação da ordem de precedência de acesso, entre os eixos $(x, y, z)$, as 64 regiões iniciais são agrupadas em três conjuntos, cada um definindo uma das pirâmides $(P_{xy}, P_{yz}, P_{xz})$ e representando uma ordem de precedência diferente (Tabela 4.1).

Da mesma forma, para a determinação do sentido de varredura em cada um dos eixos, essas 64 regiões são agrupadas em oito conjuntos, cada um determinando um octante da cena e representando um sentido de varredura diferente (ver Tabela 3.1).

A diferença existente entre este caso, e o caso de projeção ortogonal é que, no primeiro a cena foi dividida em oito regiões (octantes), tendo associado a cada uma delas uma ordem de indexação diferente. Neste caso, cada ordem de indexação não está associada a uma única região, mas sim a um conjunto de regiões que compartilham a proximidade de um mesmo vértice (formam um octante).

Outra consideração, relativa à projeção em perspectiva, ainda deve ser feita. O fato do observador estar localizado em uma posição finita do eixo $z$ (Figura 2.10), faz com que os objetos sejam visualizados com escalamento. Este escalamento gera um efeito de distorção, onde estruturas mais próximas parecem ser maiores que as estruturas que estão mais afastadas (distorção perspectiva).

Para simular este efeito de escalamento, os voxels mais próximos do observador devem parecer maiores que os voxels mais distantes. Isso pode ser conseguido fazendo com que os voxels mais próximos cubram mais pixels que os voxels mais distantes. Uma operação semelhante é feita na projeção ortogonal, para eliminar ruídos na imagem de saída (Seção 3.3). Porém, neste caso, o número de pixels afetados por um voxel não é constante, e depende da distância do voxel ao observador.

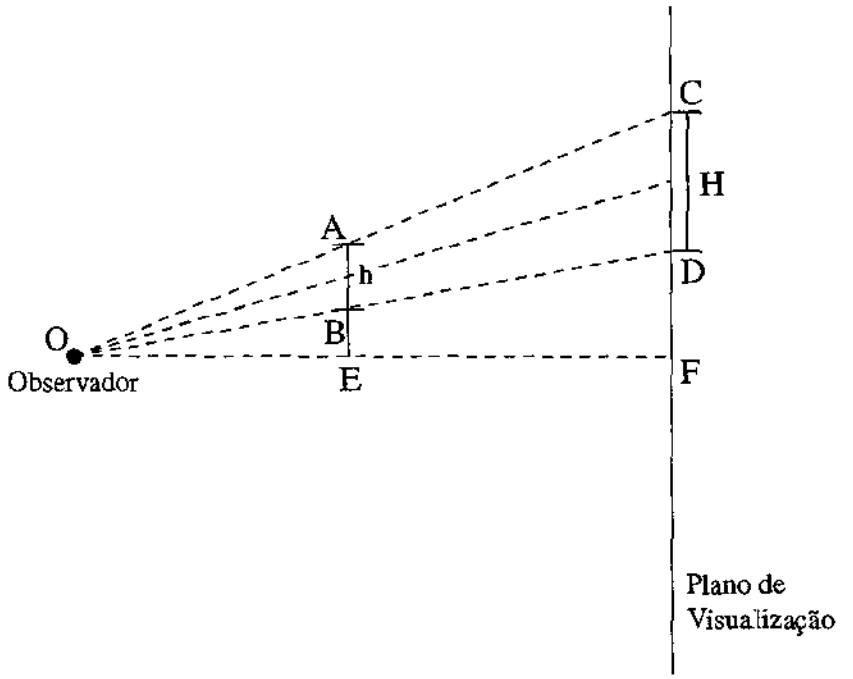



Figura 4.9: Efeito da distorção perspectiva sobre o tamanho do voxel (caso 1D).

Em perspectiva tradicional, um voxel é visto como um cubo unitário, e sua imagem projetada é definida pela posição onde seus vértices são lançados. Então, a área compreendida entre estes pontos forma a imagem do voxel sobre o plano de visualização. Essa abordagem, apesar de ser a mais correta, é computacionalmente cara pois a área coberta por um voxel geralmente não é regular.

O modelo de projeção adotado neste trabalho, usa uma simplificação razoável para esse caso: a área coberta por um voxel é sempre aproximada por uma região retangular. As dimensões dessa região retangular são determinadas em função da distância do voxel ao observador e da distância do observador ao plano de projeção, conforme o modelo unidimensional ilustrado na Figura 4.9.

Na ilustração da Figura 4.9, um voxel de altura $h$ é projetado com altura $H > h$ sobre o plano de visualização. Pela relação de similaridade entre os triângulos $AOB$ e $COD$, nós temos que:

$$\frac{H}{h} = \frac{d_{op}}{d_{ov}} \tag{4.1}$$

onde:
$d_{op}$ é a distância entre o observador e o plano de projeção, passando através do voxel (a linha mediana do triângulo $COD$);
$d_{ov}$ é a distância entre o observador e o voxel (a linha mediana do triângulo $AOB$).

Com o objetivo de aumentar a velocidade dos cálculos de *rendering*, a primeira aproximação usada foi, de acordo com o descrito acima, considerar a projeção de todo e qualquer voxel como regiões retangulares no plano de visualização. Por esse motivo, o número de pixels pintados deve ser proporcional a $H^2$. A segunda aproximação foi substituir os cálculos de $d_{op}$ e $d_{ov}$ pelas $z$-distâncias $z_{op}$ entre o observador e o plano de visualização (a altura do triângulo $COF$), e $z_{ov}$ entre o observador e o voxel (a altura do triângulo $AOE$), respectivamente.

Uma vez que $z_{op}$ depende somente da posição do observador, já que a posição do plano de visualização é fixa, os valores de $z_{ov}$ podem ser discretizados dentro do intervalo $[1, z_{op}]$. Finalmente, o mapeamento entre cada possível valor de $z_{ov}$, dentro de $[1, z_{ov}]$, em um número inteiro $n$ de pixels que serão pintados no plano de visualização, pode ser realizado através de uma *look-up table* como a apresentada a seguir:

$$n = \left\lceil round\left(h * \frac{z_{op}}{z_{ov}}\right)\right\rceil^2 \tag{4.2}$$

Este modelo de projeção em perspectiva, que usa as simplificações definidas pela Equação 4.2, define o conceito de *Projeção em Perspectiva Digital*.

Como já foi visto, a operação de projeção usualmente deixa "buracos"na imagem formada sobre o plano de visualização. Este problema foi resolvido, para o caso de projeção ortogonal, considerando os voxels não como cubos unitários, mas de dimensões maiores que 1. Portanto, cada voxel passou a cobrir uma área ligeiramente maior que as próprias dimensões originais.

Em projeção perspectiva, esse mesmo efeito pode ser obtido fazendo o parâmetro $h$, da Equação 4.2, assumir um valor maior que 1. Essa solução tem a vantagem de manter a consistência entre os modelos adotados, pois em ambos os casos a solução reside em modificar a escala inicialmente estabelecida para as dimensões de um voxel, de forma que ele cubra mais pixels na imagem de saída.

Outro efeito, provocado pela distorção perspectiva, que deve ser considerado, é que, após a transformação que leva $(x, y, z)$ a $(x', y', z')$, a projeção não pode mais ser executada apenas fazendo $(x', y') = (u, v)$, e $z = 0$, como no caso de projeção ortogonal. Na projeção em perspectiva, os voxels devem sofrem um desvio, quando lançados sobre o plano de visão. Esse efeito é ilustrado pela Figura 4.10.

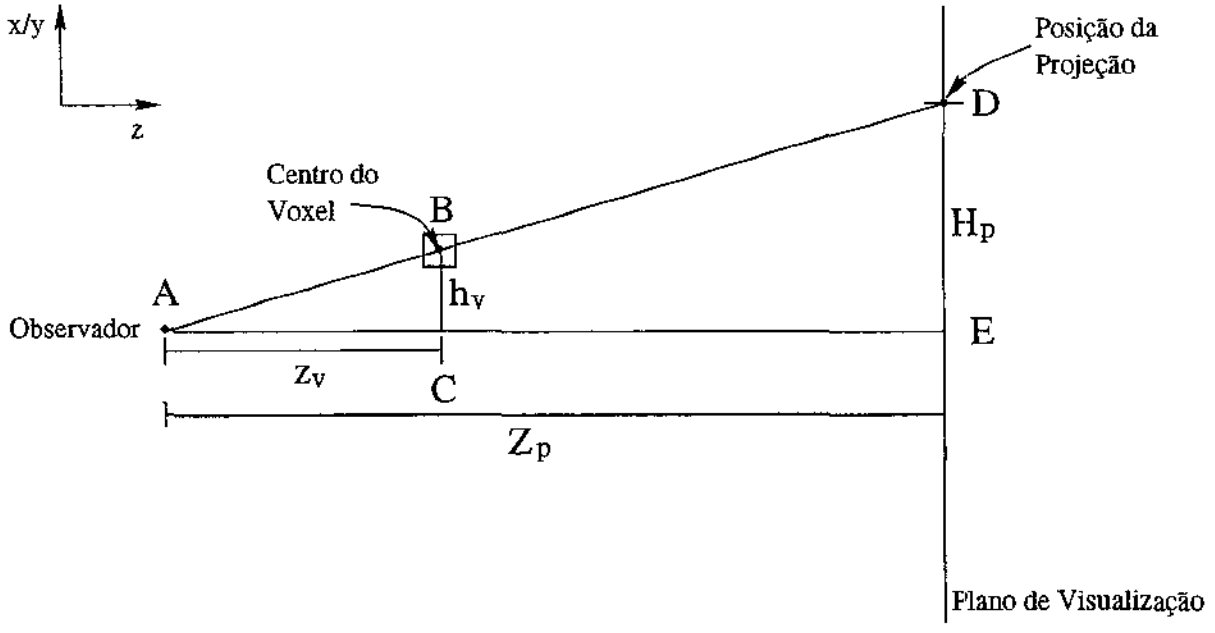



Figura 4.10: Efeito de afastamento (caso 1D).

O cálculo, para se determinar a posição $(u, v)$ em que o centro de um voxel é projetado, é feito com base na semelhança dos triângulos presentes na Figura 4.10. Nesta figura temos os triângulos $ABC$ e $ADE$, através dos quais podemos extrair a seguinte relação:

$$H_p = h_v * \left(\frac{Z_p}{z_v}\right) \tag{4.3}$$

onde:
$h_v$ é a altura do centro do voxel, em relação ao eixo $z$;
$z_v$ é a distância entre a posição do observador e a posição do voxel, sobre o eixo $z$;
$Z_p$ é a distância entre o observador e o plano de projeção, também sobre o eixo $z$.

Para que as coordenadas $(u, v)$ do voxel possam ser obtidas, essa operação é executada, após a rotação usando a matriz $M$, tanto na direção do eixo $x$ quando na direção do eixo $y$. Então, a Equação 4.3 é expressa em função dos respectivos eixos:

- *Eixo X*:

$$u = x' * \left(\frac{z_p - z_o}{z' - z_o}\right) \tag{4.4}$$

- *Eixo y*:

$$v = y' * \left(\frac{z_p - z_o}{z' - z_o}\right) \tag{4.5}$$

onde:
$z_p$ é a coordenada $z$ do plano de visualização, que no modelo adotado (Figura 2.10) assume o valor $d/2$, e $d$ é a diagonal principal da cena;
$z_o$ é a coordenada $z$ do observador, que no modelo adotado é definido por: $z_o < -d/2$;
$(z_p - z_o) = z_{op}$;
$(z' - z_o) = z_{ov}$;
$(x', y', z')$ são as coordenadas do voxel projetado, após a rotação.

## 4.4 Algoritmo *SR com Perspectiva Digital* - SRPD

Esta seção descreve o algoritmo SRPD, que implementa o *shell rendering* com projeção em perspectiva digital. Este algoritmo será apresentado a seguir, junto com a especificação dos dados de entrada e saída, estruturas auxiliares, e comentários.

De acordo com as Tabelas 3.1 e 4.1, que estabelecem as regras para o acesso, as informações necessárias para se percorrer os voxels do *shell*, para um dado octante, são os seguintes dados de entrada:

**Entrada:**

- lista $D$: vetor de registros, componente da nova estrutura *shell*, onde os atributos de visualização dos voxels são armazenados. A lista $D$ e estes atributos são mostrados na Figura 4.6a;
- matriz $P$: matriz de ponteiros, também componente do novo *shell*, usada no acesso aos elementos da lista $D$. Esta matriz mantém o registro das coordenadas $y$ e $z$ de cada linha da cena Figura 4.6a;
- lista $D'$: vetor de índices para $D$, trata-se de uma das estruturas acrescentadas ao *shell* original. Este vetor guarda o valor da coordenada $z$ dos voxels, mas não contém seus atributos de visualização. Entretanto, guarda ponteiros $ptr\_D$, para a lista $D$, onde estes atributos estão armazenados (Figura 4.6b);
- matriz $P'$: outra estrutura acrescentada ao *shell* original. $P'$ é uma matriz de ponteiros que mantém o registro das coordenadas $y$ e $x$, dos voxels em $D'$, para o tipo de acesso feito primeiro na direção $z$ (Figura 4.6b);
- ângulos $\alpha$ e $\beta$: ângulos que definem a rotação da cena em torno dos eixos $x$ e $y$ (Figura2.10), respectivamente. Estes dados são informados pelo usuário;
- $Ph$: estrutura de dados contendo as componentes $(K_s, K_d, K_a)$, o coeficiente $n$ e o valor $I_{max}$ (intensidade máxima desejada na imagem), utilizados pelo modelo de iluminação de Phong. A descrição destes itens é apresentada na Seção 2.4.3. Estes dados podem ser informados pelo usuário ou assumirem valores pré-definidos;
- $z_p$: coordenada $z$ do plano de vizualização. Este valor geralmente é fixado em $d/2$, onde $d$ é a diagonal principal da cena (Figura 2.10);
- $z_o$: valor da coordenada $z$ do observador. Este valor deve ser informado pelo usuário, pois como mostra a Figura 2.10a, o observador pode se deslocar sobre o eixo $z$, assumindo valores $z_o < -d/2$;

**Saída:**

- matriz *Pixels*: esta matriz é usada para representar a imagem que será formada pelo *rendering* . Cada posição da matriz, que representa um pixel da imagem de saída, armazena o valor de tonalização, calculado para esse pixel.

**Estruturas Auxiliares:**

- $LA[i]$: lista indexada pelos octantes, contendo informações para a execução do acesso à estrutura *shell*. Porém, diferente da estrutura apresentada na Figura 3.5, neste caso a lista $LA[i]$ contém também os dados $x_i$ e $x_f$;
- matriz *Opac*: esta matriz possui a mesma dimensão de *Pixels*, e seus elementos também estão associados a aos pixels da imagem de saída. Porém, esta matriz é usada no controle de saturação, e armazena os valores de opacidade acumulada em cada pixel;
- *oct*: índice para $LA$ que indica o octante correspondente à direção de visualização dada por $\alpha$ e $\beta$;
- *piram*: valor que identifica o tipo de acesso, em relação à ordem de precedência entre os eixos, baseado no esquema de pirâmides da Tabela 4.1;
- *prim*: endereço do primeiro voxel do *shell*, pertencente a uma determinada linha (direção $x$), dentro da lista $D$, ou a uma determinada profundidade (direção $z$), dentro da lista $D'$;
- *ult*: endereço do ultimo voxel do *shell*, pertencente a uma determinada linha (direção $x$), dentro da lista $D$, ou a uma determinada profundidade (direção $z$), dentro da lista $D'$;
- *ind*: índice para o acesso aos elementos das listas $D$ ou $D'$;
- *ind2*: índice auxiliar, usado para receber o valor referenciado por algum elemento da lista $D'$;
- $x$: índice para a matriz de ponteiros $P'$, representando a coordenada $x$;
- $y$: índice para $P$ ou $P'$, representando a coordenada $y$;
- $z$: índice para a matriz de ponteiros $P$, representando a coordenada $z$;
- $n$: número de pixels, dentro de uma área quadrada, afetados pela projeção de um determinado voxel;
- $(x', y', z')$: coordenada do voxel $(x, y, z)$, após a aplicação da matriz $M$;
- $(n'_x, n'_y, n'_z)$: componentes do vetor normal ao voxel $(x, y, z)$, após a aplicação da matriz $M$;

- $(u, v)$: valores usados como índices para a matriz $Pixels$, representam as coordenadas de um pixel da imagem de saída no plano de visualização;
- *tonaliz*: armazena o valor de tonalização, calculado para o voxel corrente.

*Início*

I. Calcula matriz $M$ através dos ângulos $\alpha$ e $\beta$ (Equação 2.3);

II. Calcula o octante *oct*, através de $\alpha$ e $\beta$; */* Direção de indexação */*

III. Calcula a pirâmide *piram*, através de $\alpha$ e $\beta$; */* Ordem de precedência */*

IV. ***caso*** (piram)

a1. $(x \rightarrow y \rightarrow z)$: Executar acesso_$(x, y, z)$;

a2. $(x \rightarrow z \rightarrow y)$: Executar acesso_$(x, z, y)$;

a3. $(z \rightarrow y \rightarrow x)$: Executar acesso_$(z, y, x)$;

***fim_caso***

*fim*

O caso de acesso apresentado no ítem *IV.a3.*, difere consideravelmente dos demais. Como o acesso a $x$ é feito por último, as estruturas convencionais do *shell* não podem ser utilizadas (ver discussão sobre essa limitação na Seção 4.1). Portanto, o item *IV.a3.* representa o caso de acesso onde a varredura é feita sobre as estruturas $D'$ e $P'$. Este detalhe, juntamente com a implementação da distorção perspectiva, são mostrados no algoritmo a seguir:

/* Caso de acesso IV.a3. */

I. **para** ($x \leftarrow LA[oct].x_i$ **até** $LA[oct].x_f$ **c/ incremento** $LA[oct].d_x$) **faça**

A. **para** ($y \leftarrow LA[oct].y_i$ **até** $LA[oct].y_f$ **c/ incremento** $LA[oct].d_y$) **faça**

1. **se** ($LA[oct].d_z > 0$) **então**

/* Direção crescente de z */

a. $prim \leftarrow P'[y, x]$;

b. $ult \leftarrow P'[y, x] + 1$; /* Próximo ponteiro não nulo */

**senão**

/* Direção decrescente de z */

c. $prim \leftarrow P'[y, x] + 1$; /* Próximo ponteiro não nulo */

d. $ult \leftarrow P'[y, x]$;

**fim_se**

2. **para** ($ind \leftarrow prim$ **até** $ult$ **c/ incremento** $LA[oct].d_z$) **faça**

a. $ind2 \leftarrow D'[ind].ptr\_D$;

b. $(x', y', z') \leftarrow (x, y, D'[ind].z) \times M$; /* Aplica matriz de rotação*/

/* Projeção em perspectiva digital */

c. $u \leftarrow x' * \left(\frac{z_p - z_o}{z' - z_o}\right)$;

d. $v \leftarrow y' * \left(\frac{z_p - z_o}{z' - z_o}\right)$;

e. Calcula número de pixels $n$, através da Equação 4.2;

f. **para** ($(u - \sqrt{n}) \leq u \leq (u + \sqrt{n})$ **e** $(v - \sqrt{n}) \leq v \leq (v + \sqrt{n})$) **faça**

a) **se** $Opac[u, v] < 1$ **então**

/* Se pixel $(u, v)$ não saturado */

i. $(n'_x, n'_y, n'_z) \leftarrow (D[ind2].n_x, D[ind2].n_y, D[ind2].n_z) \times M$;

ii. Calcula $I_{dist}$, através da Equação 2.7, usando $z'$ como $d(u, v)$;

iii. Calcula o ângulo $\theta$, entre a normal e a direção de visualização, usando $(n'_x, n'_y, n'_z)$;

iv. $tonaliz \leftarrow Ph.I_{max} \times Ph.K_a + I_{dist}(Ph.K_d cos\theta + Ph.K_s cos^n\theta)$; /* Phong */

v. $Pixels[u, v] \leftarrow Pixels[u, v] + (tonaliz \times D[ind2].opac \times Opac[u, v])$; /* Equação 2.6 */

vi. $Opac[u, v] \leftarrow Opac[u, v] + (1.0 - D[ind2].opac)$;

*fim_se*
*fim_para*

*fim_para*
*fim_para*
*fim_para*

Os outros casos de acesso, referenciados como *IV.a*1. e *IV.a*2. são muito semelhantes entre si, e *IV.a*1. tem a mesma estrutura do algoritmo apresentado para o caso ortogonal (Seção 3.3), exceto pelo fato de *IV.a*1. incluir o código para realizar a projeção com distorção perspectiva (itens *I.A.*2.*c.*, *I.A.*2.*d.*, *I.A.*2.*e.*, *I.A.*2.*f.*, do algoritmo anterior). Este código é responsável pela obtenção das coordenadas $(u, v)$, onde o voxel será projetado, através das equações Equações 4.4 e 4.5. A diferença entre *IV.a*1. e *IV.a*2. está na ordem de execução dos laços envolvendo as coordenadas $y$ e $z$. Em *IV.a*1., o laço $z$ é o mais externo, portanto $y$ é executado antes. Em *IV.a*2., ao contrário, o laço $y$ é o mais externo, então $z$ é executado primeiro. O importante é que, tanto em *IV.a*1. quanto em *IV.a*2, o laço $x$ é o mais interno, então é executado primeiro. Portanto, o acesso é feito sobre as estruturas tradicionais do *shell*, $D$ e $P$.

O função de projeção em perspectiva é muito semelhante à função usada no caso ortogonal. A maior diferença está na definição do número de pixels afetados pela projeção de um voxel. No caso ortogonal, todos os voxels afetam um número constante de pixels, sempre uma pequena vizinhança em torno do pixel sobre o qual é projetado o centro do voxel em questão (Figura 3.6). Entretanto, no caso de projeção em perspectiva, esse número é variável e deve ser calculado em função da distância entre o voxel e o observador (ver Equação 4.1). A outra diferença significativa é a realização de um deslocamento do voxel, após sua rotação. Esse deslocamento é feito para a obtenção das coordenadas $(u, v)$, onde o voxel será projetado (ver Equações 4.4 e 4.5). O cálculo de tonalização, propriamente dito, é realizado exatamente como no caso ortogonal, com os mesmos parâmetros e os mesmos coeficiêntes de reflexão.

# Capítulo 5

# Resultados

O objetivo desta seção é apresentar os resultados de implementação dos métodos *Shell Rendering* com projeção ortogonal e em perspectiva digital, bem como descrever a metodologia e os experimentos realizados para comparar estes métodos.

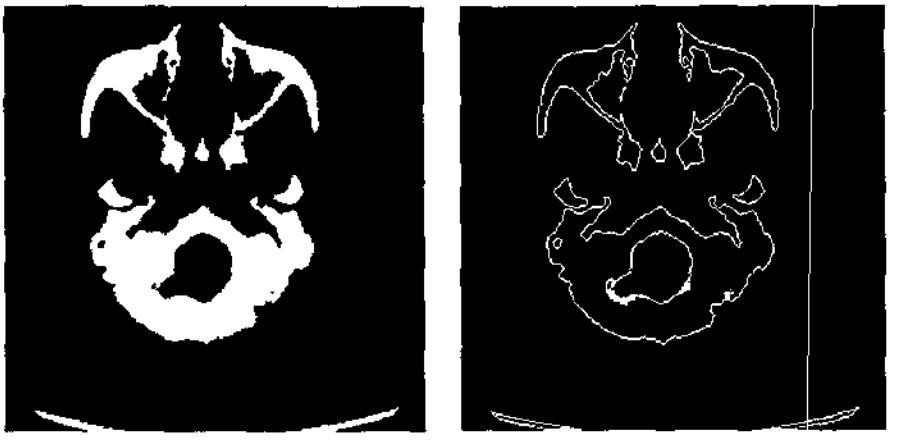
Este trabalho, teve seu desenvolvimento orientado pela preocupação em manter a coerência entre os módulos para projeção ortogonal e em perspectiva digital. Esse empenho foi motivado principalmente pela necessidade de integração, de ambos os módulos, em um único sistema de visualização. Portanto, a implementação do módulo de projeção em perspectiva digital foi orientada no sentido de se afastar o mínimo possível da implementação do módulo para projeção ortogonal. Esse esforço possibilitou a utilização indistinta da maioria das estruturas e rotinas em ambos os módulos. Permitiu também uma uniformização dos conceitos teóricos aplicados no desenvolvimento deste sistema.

Os programas foram implementados em ANSI-C e testados em uma plataforma *Linux*, com o uso do compilador GCC - *v*2.7. Foram testados também sobre plataformas *Unix*, como a *Solaris*. Os programas se dividem em dois blocos funcionais. O primeiro é responsável pela leitura e tratamento das imagens que formam a cena original, possibilitando a construção do *shell* a partir delas. Algumas opções são possíveis nesse momento, como a geração do *shell* para *rendering* de superfícies ou volume.

Quando a opção de *rendering* de superfícies é escolhida, o *shell* é reduzido ao conjunto de voxels pertencentes à borda do objeto, como mostra a imagem da Figura 5.1b. Nesse caso, todos os voxels contidos no *shell* possuem opacidade igual a 1.

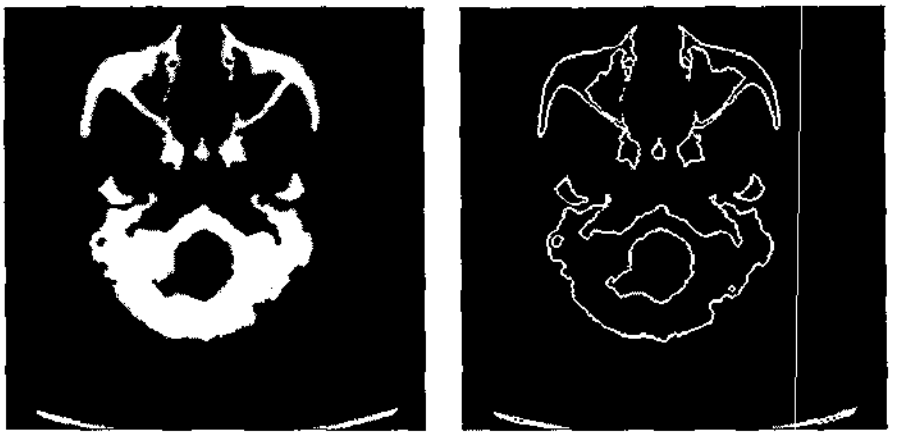
Na opção de *rendering* de volumes, a estrutura *shell* deve conter os voxels numa certa vizinhança das bordas do objeto de interesse, de forma que se preserve não só os seus contornos (Figura 5.1b), mas também uma certa camada do interior, adjacente à borda (Figura 5.2b). Esta nova cena é obtida a partir da cena de opacidades (Figura 5.2a) e contém as informações necessárias para se obter o *shell* para *rendering* de volumes. Neste caso, os voxels devem possuir valores de opacidade dentro do intervalo $[0, 1]$.

Nos experimentos realizados, o cálculo das normais foi feito ou sobre a cena cinza original, ou sobre a cena binária, filtrada como descrito na Seção 2.3 (Figura 2.9). Os possíveis caminhos de processamento para a cena original, que levam à criação de um *shell* contendo a estrutura de interesse, estão descritos na Figura 5.3.



(a) (b)

Figura 5.1: (a) Cena binária original. (b) Cena binária, da qual é extraído o *shell* para rendering de superfície.



(a) (b)

Figura 5.2: (a) Cena de opacidades. (b) Imagem cinza contendo um *shell* de espessura igual a 5.

O segundo bloco funcional, citado acima, agrupa os programas responsáveis pelo *rendering* das estruturas contidas em um *shell*, e pela sua visualização, a partir da posição especificada pelo usuário. Esse módulo recebe como parâmetros os arquivos contendo um *shell*, os ângulos de rotação da cena (que vão definir o ponto de visualização dessa cena), e os coeficientes de iluminação (bem como o expoente relativo ao tipo de material).

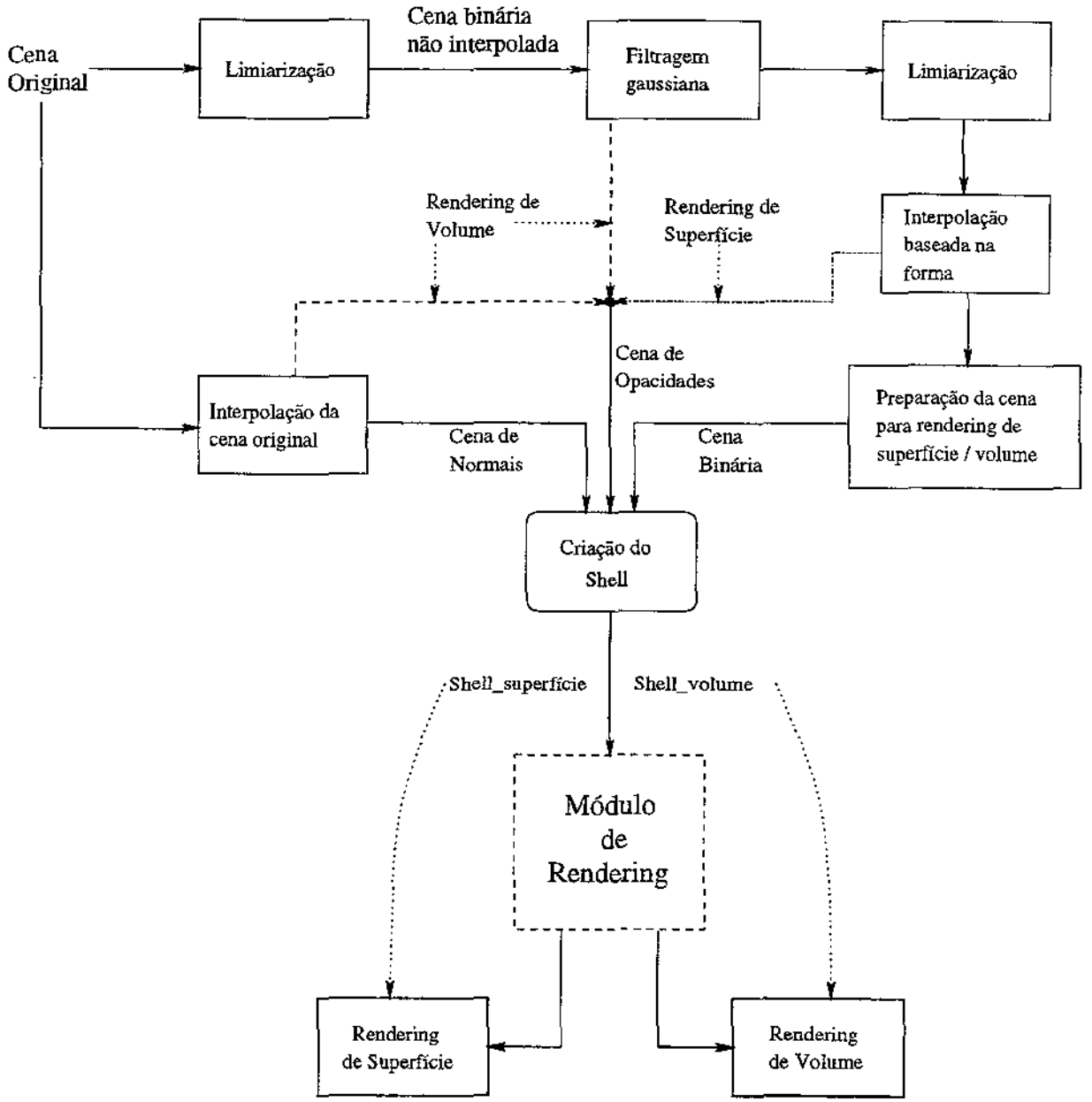



Figura 5.3: Fluxo de dados representando as etapas de processamento de uma cena cinza, até a criação do *shell* correspondente, e seu uso no *rendering* .

A seguir serão demonstrados os resultados comparativos, obtidos com o uso da nova técnica de *rendering* com perspectiva digital. Foram comparados o *shell rendering* em perspectiva digital, e o *shell rendering* com projeção ortogonal, sob dois aspectos: qualidade de imagem e tempo computacional, para *rendering* de superfícies e de volumes. Esses experimentos levaram em conta os seguintes métodos:

- **RSPO** - *rendering* de superfícies com projeção ortogonal;
- **RSPD** - *rendering* de superfícies em perspectiva digital;
- **RVPO** - *rendering* de volumes com projeção ortogonal;
- **RVPD** - *rendering* de volumes em perspectiva digital.

Foram escolhidas imagens de CT de um crânio seco e de um paciente real. As cenas 3D do crânio e do paciente contém 16,45M voxel e 23.72M voxels, respectivamente. Inicialmente, em ambos os conjuntos de dados (ainda não interpolados), a classificação foi feita por limiarização (*thresholding*). As cenas binárias geradas foram suavizadas através de um filtro gaussiano, seguido de outra operação de limiarização. Após essa etapa, uma operação de interpolação baseada na forma foi aplicada sobre essas cenas binárias, assim como as cenas originais também foram interpoladas, para eliminar a anisotropia. As cenas binárias, resultantes da interpolação, foram preparadas para a criação de um *shell* com espessura 1, para *rendering* de superfícies. Já em *rendering* de volumes, as cenas obtidas através da filtragem gaussiana foram usadas para a criação de um *shell* com espessura 5 (Figura 5.2b).

A cena de opacidades, para *rendering* de volumes, foi computada na saída do filtro gaussiano. As normais foram calculadas, no caso do crânio seco, sobre a cena cinza original, e no caso do paciente, na saída do filtro gaussiano. Desses conjuntos de dados, foram calculadas três estruturas de interesse, e criados *shells* binário e nebuloso para cada estrutura:

- $\mathbf{S_1}$ - o crânio seco;
- $\mathbf{S_2}$ - o crânio do paciente;
- $\mathbf{S_3}$ - a face do paciente.

O tamanho dos *shells* binário e nebuloso $S_1$, $S_2$ e $S_3$ são 0.32M e 0.53M voxels, 0.40M e 0.70M voxels, 0.63M e 1.10M voxels, respectivamente. Nota-se que o tamanho dos *shells* representam, neste caso, uma redução em torno de $97-98\%$ do tamanho da cena original.

A estrutura $S_1$ foi usada para se comparar a qualidade de imagem entre $RSPO$ (Figura 5.4a) e $RSPD$ (Figura 5.4b), e entre $RVPO$ (Figura 5.5a) e $RVPD$ (Figura 5.5b).

Nota-se que *shell rendering* com perspectiva digital oferece a mesma boa qualidade de imagem que a projeção ortogonal, tanto no caso de *rendering* de superfícies quanto no *rendering* de volumes. Também é possível notar que o *rendering* de volumes mostrado na Figura 5.5 aparenta ser similar, porém mais suave que o *rendering* de superfícies mostrado na Figura 5.4. De fato, o *shell rendering* usando uma "casca" fina pode remover uma grande quantidade de *aliasing*, que aparece no *rendering* de superfícies.

Outra característica do *rendering* de volumes é a possibilidade de se visualizar diversas estruturas ao mesmo tempo. No *rendering* de volumes tradicional, a imagem que combina o *rendering* de múltiplas estruturas é geralmente nebulosa, tornando difícil a visualização de detalhes das estruturas. No presente trabalho, as estruturas $S_2$ e $S_3$ foram combinadas para mostrar que o *shell rendering* pode criar o *rendering* de múltiplas estruturas, resultando em imagens muito nítidas. As Figuras 5.6a e 5.6b mostram esse resultado, comparando $RSPD$ com $RVPD$, respectivamente.

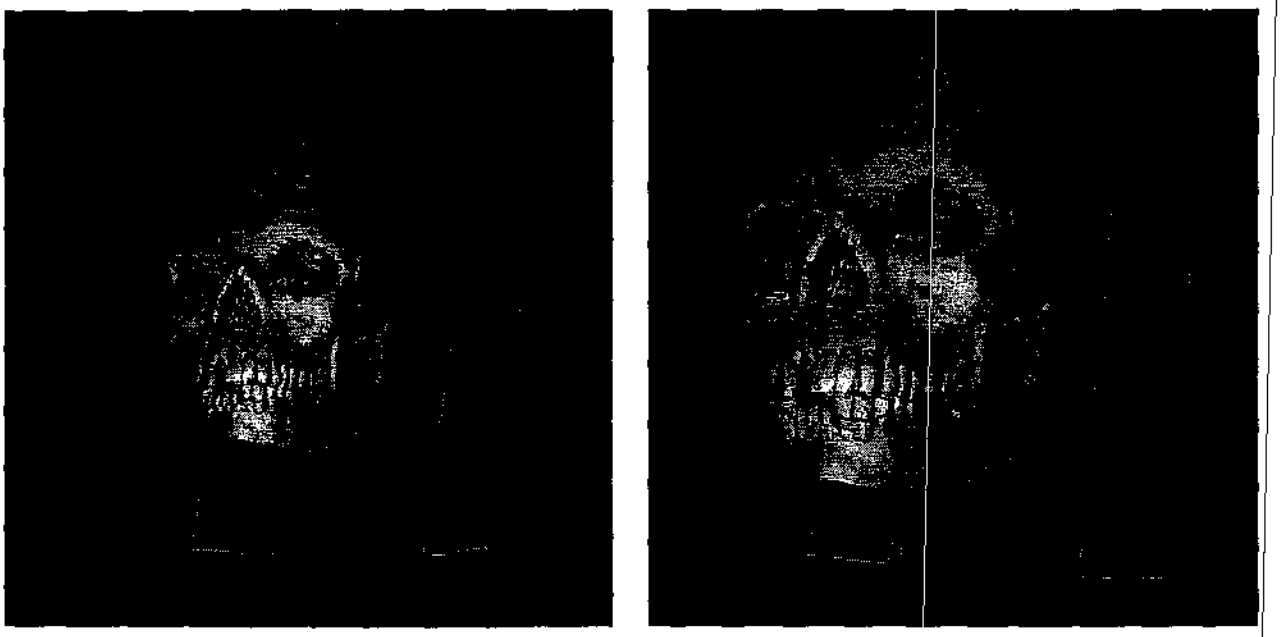
| Estruturas | Tamanho (voxels) | $RSPO$ (segundos) | $RSPD$ (segundos) |
|---|---|---|---|
| $S_1$ | 0.32M | 0.41 | 0.63 |
| $S_2$ | 0.40M | 0.51 | 0.86 |
| $S_3$ | 0.63M | 0.78 | 1.27 |

Tabela 5.1: O tamanho do *shell* e a média do tempo computacional dos métodos $RSPO$ e $RSPD$ para as estruturas $S_1$, $S_2$, e $S_3$.

Foram medidos, em um Pentium-II 400MHz, os tempos computacionais para *rendering* sob diferentes ângulos de *spin* e *tilt*, e também para o observador em diferentes posições. A Tabela 5.1 mostra os tempos computacionais médios (em segundos) de $RSPO$ e $RSPD$, para as estruturas $S_1$, $S_2$ e $S_3$, e a Tabela 5.2 mostra os tempos dos métodos $RVPO$ e $RVPD$, para as mesmas estruturas. Nota-se que a projeção em perspectiva digital é apenas 1.6 vezes mais lenta que a projeção ortográfica paralela. O *rendering* é feito em menos de 1.3 segundo, para *rendering* de superfícies, e em menos de 2 segundos, para *rendering* de volumes.

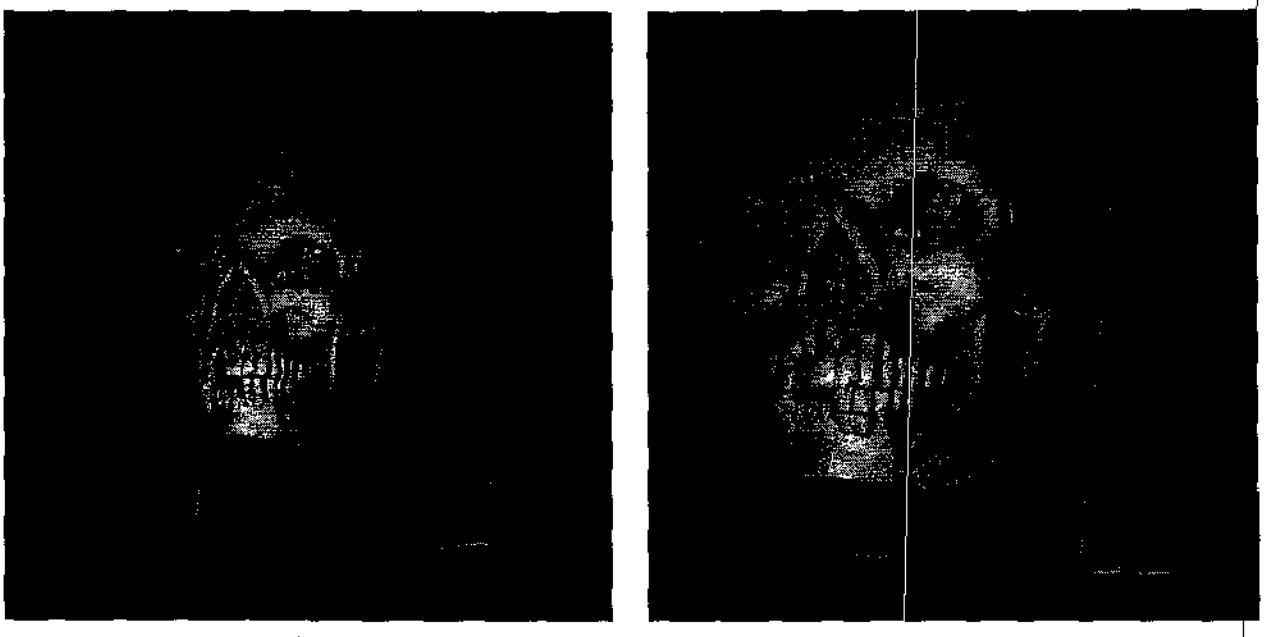
Esses resultados certamente são significativos, considerando que tanto a boa qualidade das imagens e tempos computacionais foram tomados em um PC típico, sem uso de qualquer *hardware* especializado.

Finalmente, foi comparado $RVDP$ com um método de *rendering* de volumes com perspectiva tradicional, usando em ambos os casos projeção de voxels como regiões retangulares no plano de visualização. Os métodos foram usados na composição de dois tecidos distintos em uma mesma imagem, e os resultados são mostrados na Figura 5.7.



(a) (b)

Figura 5.4: Um crânio seco reconstruido a partir de imagens de CT. (a) *Rendering* de superfícies com projeção ortogonal; (b) *Rendering* de superfícies com projeção perspectiva digital.



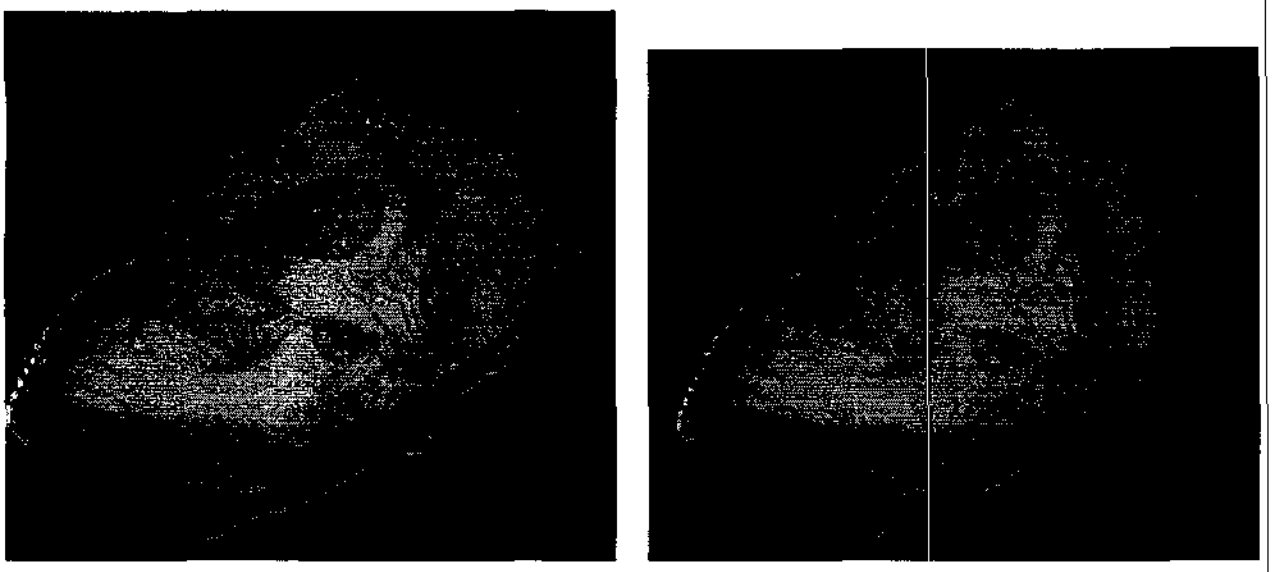
(a) (b)

Figura 5.5: Um crânio seco reconstruído a partir de imagens de CT. (a) *Rendering* de volumes com projeção ortogonal; (b) *Rendering* de volumes com projeção perspectiva digital.

| Estruturas | Tamanho (voxels) | $RVPO$ (segundos) | $RVDP$ (segundo |
|---|---|---|---|
| $S_1$ | 0.53M | 0.67 | 0.98 |
| $S_2$ | 0.70M | 0.91 | 1.36 |
| $S_3$ | 1.10M | 1.28 | 2.00 |

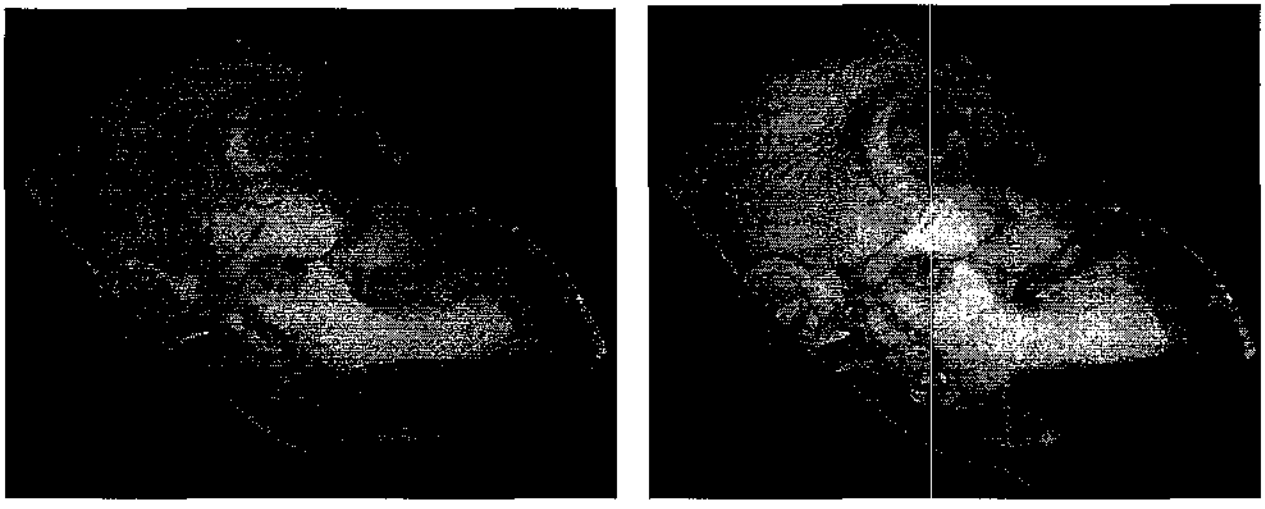
Tabela 5.2: O tamanho do *shell* e a média do tempo computacional dos métodos $RVPO$ e $RVPD$ para as estruturas $S_1$, $S_2$, e $S_3$.

Pode-se concluir que ambos os métodos produzem imagens de qualidades similares, e que o método baseado em perspectiva digital é em torno de 1.5 vezes mais rápido que o método de perspectiva tradicional. Esses resultados fundamentam a teoria de que o *rendering* de volumes em perspectiva digital é a melhor solução para visualização interativa.



(a) (b)

Figura 5.6: Crânio e face de um paciente real, reconstruídos de imagens de CT usando projeção perspectiva digital. (a) *Rendering* de superfícies; (b) *Rendering* de volumes.



(a) (b)

Figura 5.7: Crânio e face de um paciente real, reconstruídos de imagens de CT usando projeção perspectiva: (a) Digital (b) Tradicional.

# Capítulo 6

# Conclusão

Este trabalho apresentou uma extensão para a estrutura *shell*, originalmente proposta e desenvolvida por Udupa e Ohdner [29], um algoritmo para indexação FTB dos voxels em projeção perspectiva, que é independente da posição do observador, e um método para converter o valor discreto da distância, entre o observador e um voxel, na área ocupada por esse voxel sobre o plano de visualização.

O método aqui desenvolvido foi comparado ao *shell rendering* com projeção ortográfica paralela, sob dois aspectos: qualidade de imagem e tempo computacional para *rendering*. O *shell rendering* em perspectiva digital mostrou produzir a mesma qualidade de imagem que o *shell rendering* com projeção ortogonal. Através dos testes realizados em um PC Pentium II 400MHz, sem qualquer tipo de *hardware* especializado, foi possível mostrar que o *rendering* de superfícies e de volumes, em perspectiva digital, levam menos de 1.3 e 2.0 segundos, respectivamente, para completar o *rendering* de estruturas médicas típicas, e que o tempo requerido para o *rendering* em perspectiva é apenas por volta de 1.6 vezes mais lento que o requerido para *shell rendering* com projeção ortográfica paralela.

A extensão natural deste trabalho é posicionar o observador no interior da cena. No entanto, esta tarefa apresenta uma dificuldade adicional, pois dada uma coordenada $(x, y, z)$ no interior da cena, o acesso aos voxels mais próximos desta posição não é direto, considerando o uso da estrutura *shell*.

O estudo deste caso mostrou que a complexidade na manipulação do *shell* aumenta na seguinte ordem: *rendering* com projeção ortogonal, com projeção perspectiva e projeção perspectiva com o observador no interior da estrutura.

A dificuldade na transição do primeiro para o segundo método foi discutida em detalhes no Capítulo 4. Os problemas encontrados na implementação do terceiro método serão apresentados a seguir.

Quando o observador é colocado no interior da estrutura de interesse, o acesso FTB deve ser iniciado na posição do observador e seguir primeiro na direção de visualização.

Porém, como já foi mostrado no Capítulo 4, realizar o acesso ao *shell* numa direção arbitrária não é trivial, e implica em altos custos computacionais. Essa característica não representa nenhum tipo de limitação quando o observador está fora da cena 3D, mesmo no caso de projeção em perspectiva, pois o acesso pode sempre ser feito partindo de um determinado vértice, e seguir sobre os eixos $x, y, z$ em direção ao vértice oposto. Uma forma de tratar este problema é dividir a cena em setores (Figura 6.1) e realizar o acesso a cada setor na direção dos eixos principais originais (eixos $y'$ e $z'$).

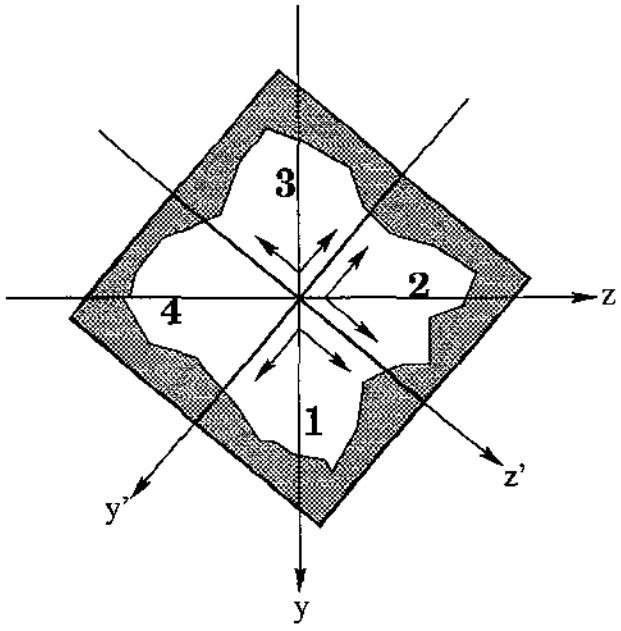



Figura 6.1: Problemas decorrentes do posicionamento do observador no interior da estrutura de interesse.

No exemplo apresentado na Figura 6.1, a direção de visualização é definida pelo eixo $z$, e o acesso é feito nas regiões 1, 2 e 3 de forma a se afastar do observador. Neste exemplo, a região 2 está totalmente inserida no campo de visão. Porém, as regiões 1 e 3 são vistas apenas parcialmente. Considerando o eixo $y$ como o delimitador do campo de visão do observador, torna-se necessário, nas regiões 1 e 3 verificar o ponto de parada da varredura. O teste de parada pode ser feito verificando se um voxel está do lado positivo ou negativo do eixo $y$. Antes disso, entretanto, é preciso determinar quais regiões são vistas apenas parcialmente, em função da rotação da cena.

Outro problema é identificar o ponto de partida da varredura no *shell* pois, ao contrário do que ocorre nos casos onde o observador está fora da cena, aqui a projeção não pode ser feita baseada apenas na direção de visualização. Ao posicionar o observador no interior da estrutura de interesse, o que geralmente se deseja é obter a visualização de espaços internos e cavidades desta estrutura. Portanto, o observador provavelmente estará numa

região vazia. Em outras palavras, o voxel sobre o qual o observador se encontra não pertence ao *shell*. Como o *shell* não mantém o posicionamento relativo entre os voxels da cena original, encontrar uma coordenada $(x, y, z)$ dentro do *shell* é uma tarefa que requer busca sequencial. No exemplo da região 2, a varredura na direção de $y'$ teria que ser iniciada na região 1, e a varredura na direção $z'$ iniciada na região 3. Não é possível acessar apenas os elementos de uma determinada região. O acesso realizado com essas restrições, impostas pelo *shell*, é muito ineficiente, pois cada região seria varrida mais de uma vez, e até mesmo a região 4, que não é vizível, seria acessada.

Fica claro que a estrutura *shell* apresenta algumas barreiras para sua utilização neste tipo de situação, onde o observador está posicionado no interior da cena. Em última análise, o *rendering* ortogonal usando a estrutura *shell* requer apenas o cuidado com a direção de acesso em cada eixo principal. No caso do *shell rendering* em perspectiva, a ordem de acesso aos eixos também importa. E no caso do observador no interior da estrutura, é necessário definir o acesso considerando a posição do observador. Portanto, a obtenção de uma solução eficiente para o caso do observador no interior da cena fica como uma sugestão de trabalho futuro.

# Referências Bibliográficas


[1] A. Kaufman, ed., A tutorial on volume visualization, *IEEE Computer Society Press*, Los Alamitos, CA, 1990.

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

[3] A.X. Falcão, Visualização de volumes aplicada à área médica, *FEEC-UNICAMP*, Feb, 1993, Tese de Mestrado.

[4] G.P. Carnielli, A.X. Falcão, and J.K. Udupa, Fast Digital Perspective Shell Rendering, *XII Brazilian Symposium on Computer Graphics and Image Processing*, sponsored by SBC, Campinas - SP, Oct 1999, **a ser publicado**.

[5] J.K. Udupa, G. Herman, eds., *3D imaging in medicine*, CRC Press, Boca Raton, FL, 1991.

[6] M.W. Vannier, J.L. Marsh and J.O. Warren, Three-dimensional computer graphics for craniofacial surgical planning and evaluation, *Computer Graphics*, vol. 17, Jul, 1983, 263–274.

[7] J.K. Udupa and B.E. Hirsch and S. Samarasekera and H. Hillstrom and G. Bauer and B. Kneeland, Analysis of in vivo 3D internal kinematics of the joints of the foot, *IEEE Transactions on Biomedical Engineering*, vol. 45, pp. 1387-1396, 1998.

[8] R.C. Rhoad and J.J. Klimkiewicz and G.R. Williams and S.B. Kesmodel and J.K. Udupa and B. Kneeland and J.P. Iannotti, A new in vivo technique for 3D shoulder kinematics analysis, *Skeletal Radiology*, vol. 27, pp. 92-97, 1998.

[9] M. Bentum, Interactive Visualization of Volume Data, *Dept. of Electrical Engeneering, Univ. of Twente*, Enschede, The Netherlands, Jun, 1996, *PhD Thesis.*

[10] M. Levoy, Volume rendering by adaptive refinement, *The Visual Computer*, **6**(1), pp. 2–7, 1990.

[11] R.A. Drebin, L. Carpenter and P. Hanrahan, Volume rendering, *Computer Graphics*, **22**(4), Aug, 1988, 65–74.

[12] M. Levoy, Display of surfaces from volume data, *IEEE Computer Graphics and Applications*, **5**(3), May, 1988, 29–37.

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

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

[15] J.K. Udupa, H. Hung and K. Chuang, Surface and volume rendering in 3D imaging: a comparison, *Journal of Digital Imaging*, vol. 4, 1991, 159–168.

[16] P. Lacroute and M. Levoy, Fast volume rendering using a shear-warp factorization of viewing transformation, *Computer Graphics*, **28**(4), pp. 451–458, 1994.

[17] R. Reynolds, D. Gordon and L. Chen, A dynamic screen technique for shaded graphics display of slice-represented objects, *Computer Vision, Graphics and Image Processsing*, vol. 38, 1987, 275–298.

[18] W.E.,Lorensen and H.E.,Cline, Marching cubes: A high resolution 3D surface construction algorithm, *Computer & Graphics*,**21**(4), pp. 163–169, Jul, 1987.

[19] D. Gordon and J.K. Udupa, Fast surface tracking in three-dimensional binary images, *Computer Vision, Graphics and Image Processing*, vol. 45, 1989, 196–214.

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

[21] A. Kalvin, Segmentation and surface-based modeling of objects in three-dimensional biomedical images, *Dept. of Computer Science, New York University*, Mar, 1991, *PhD Thesis*.

[22] P. Schröder and G. Stoll, Data parallel volume rendering as line drawing, *Proceedings of the 1992 Workshop on Volume Visualization*, pp. 25–32, Boston, Oct, 1992.

[23] G. Vézina, P.A. Fletcher and P.K. Robertson, Volume rendering on the MasPar MP-1, *Proceedings of the 1992 Workshop on Volume Visualization*, pp. 3–8, Boston, Oct, 1992.

[24] G.G. Cameron and P. E. Undrill, Rendering volumetric medical image data on a SIMD-architecture computer, *Proceedings of the Third Eurographics Workshop on Rendering*, pp. 135–145, Bristol, UK, May, 1992.

[25] D. Laur and P. Hanrahan, Hierarchical splatting: A progressive refinement algorithm for volume rendering, *Proceedings of SIGGRAPH'91. Computer Graphics*, **25**(4), pp. 285–288, July, 1991.

[26] J. Danskin and P. Hanrahan, Fast algorithms for volume ray tracing, *Proceedings of the 1992 Workshop on Volume Visualization*, pp. 91–98, Boston, Oct, 1992.

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

[28] K.J. Zuiderveld, A.H.J. Koning and M.A. Viergever, Acceleration of ray-casting using 3D distance transforms, *Proceedings of Visualization in Biomedical Computing 1992*, pp. 324–335, North Carolina, Oct, 1992.

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

[30] G.J. Grevera, J.K. Udupa, and D. Odhner, One order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware, *in Proceedings of SPIE on Medical Imaging'99*, San Diego, CA, **3658**, pp. 202–211, Feb 1999.

[31] C.F.X. de Mendonça, A.X. Falcão, A.C. Vannini, and J.J. Lunazzi. Fast Holographic Stereograms Display using Shell Rendering and a Holographic Screen. *In Proceedings of SPIE on Medical Imaging*, San Diego, CA, vol. 3658, pp. 484-492, Feb 1999.

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

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

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

[35] A. Goshtasby, D.A. Turner and L.V. Ackerman, Matching of tomographic slices for interpolation, *IEEE Transactions on Medical Imaging*, **11**(4):507–516, 1992.

[36] R.W. Parrott, M.R. Stytz, P. Amburn and D. Robinson, Statistically optimal interslice value interpolation in 3D medical imaging: Theory and implementation,*Proceedings of the Fifth IEEE Symposium on Computer-Based Medical Systems*, pp. 272–283, Jun, 1992.

[37] S.P. Raya and J.K. Udupa, Shape-based interpolation of multidimensional objects, *IEEE Transactions on Medical Imaging*, 9:32–42, 1990.

[38] G.T. Herman, J. Zheng and C.A. Bucholtz, Shape-based interpolation, *IEEE Computer Graphics and Applications*, **12**(3):69–79, 1992.

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

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

[41] R.A. Lotufo and A.X. Falcão, Shape-based interpolation methods applied to medical imaging, *In Proceedings of SIBGRAPI VI*, pp. 323–331, 1993.

[42] S. Chen, W. Lin, C. Liang and C. Chen, Improvement on dynamic elastic interpolation technique for reconstructing 3-D objects from serial cross sections, *IEEE Transacitons on Medical Imaging*, **9**(1), pp. 71–83, Mar, 1990.

[43] M. Levoy, P. Hanrahan, K.H. Hoehne, A. Kaufman and W. Lorensen, Course Notes: Volume Visualization Algorithms and Architectures, *SIGGRAPH 1990, 17th International Conference On Computer Granphics and Interactive Techniques*, Dallas, 1990.

[44] A.R. Smith, Volume Graphics and Volume Visualization: A Tutorial, *Pixar Inc.: Technical Memo 176*, California, 1987.

[45] H. Fuchs, Z.M. Kedam and S.P. Uselton, Optimal surface reconstruction from planar contours, *Communications of the ACM*, **20**(10), pp. 693–702, Oct, 1977.

[46] K.D. Toennies and U. Tronnier, 3D Modeling using an extended cell enumeration representation, *Computer Graphics*, **24**(5), pp. 13–20, Nov, 1990.

[47] P. Sabella, A rendering algorithm for visualizing 3D scalar fields, *Computer Graphics*, **22**(4), Aug, 1988, 51–58.

[48] C. Upson and M. Keeler, V-BUFFER: Visible volume rendering, *Computer Graphics*, vol. 22, Atlanta, 1988, 59–64.

[49] L. Westover, Footprint evaluation for volume rendering, *Computer Graphics*, vol. 24, Dallas, 1990, 367–376.

[50] G. Frieder, D. Gordon and R. A. Reynolds, Back-to-front display of voxels based objects, *IEEE Computer Graphics and Applications*, **5**(1), Jan, 1985, 52–60.

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

[52] H. Gouraud, Continous shading of curved surfaces, *IEEE Transaction of Computers*, **20**(6), 1971, 623–629.

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

[54] M. Vannier, C. Hildebolt, J. Marsh, T. Pilgram, W. McAlister, G. Shackelford, C. Offutt and R. Knapp, Craniosynostosis: diagnostic value of three-dimensional CT reconstruction, *Radiology*, vol. 173, pp. 669–673, 1989.