UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Conjunto de Operadores Básicos Para Visualização, Manipulação e Análise de Dados 3D

por Renato Marcos Silva Luppi

orientador Prof. Dr. Roberto de Alencar Lotufo

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

Outubro 94

Este exemplar corresponde à redação final da tese defendida por Renato Marcos Silva Luppi

Julgadora em 14 10 94

UNICAMP
BIBLIOTECA CENTRAL

Resumo

Esta dissertação apresenta um conjunto de operadores básicos desempenhando funções diversas em visualização, manipulação e análise de dados tridimensionais. Pelo fato de serem básicos, a combinação destes operadores permite a construção de funções mais complexas, onde a flexibilidade de se obter novos operadores, de níveis mais elevados, é resultado direto do número de operadores elementares disponíveis.

Para a implementação dos operadores foram usadas as ferramentas de desenvolvimento e manutenção de *softwares* do sistema Khoros, sendo cada operador tratado como uma rotina escrita em linguagem C. A combinação dos operadores foi feita com a utilização do ambiente de programação visual, também integrante do sistema Khoros, bem como através de *shell scripts*.

Diversos exemplos de aplicações práticas, inclusive interativas, são mostradas no final do trabalho, ilustrando a criação de processamentos poderosos construídos através da combinação dos operadores básicos.

Abstract

This work presents a set of basic operators performing a variety of functions in visualization, manipulation and analysis of tridimensional data. Because they are basic, the combination of these operators allows the construction of more complex functions, where the flexibility in obtaining new higher level operators is related directly to the number of elementary operators available.

In order to implement these operators, it was used the Khoros system software development tools, and each operator was treated as a routine written in C language. The combination of operators was done in the visual programming environment, which is also part of Khoros system, as well as using Unix shell language in shell scripts.

A lot of examples of practical aplications, as well as interactive aplications, are shown in the end of this work, to ilustrate the creation of powerful computations which were built by combining the basic operators.

Agradecimentos

Agradeço aos meus pais Warner e Maria Isaura e à minha irmã Valéria, pelo amor de sempre, sem o qual certamente a conquista de mais esta etapa seria impossível.

Agradeço à Mônica, minha grande companheira, pelo incentivo, apoio e carinho acima de tudo, essenciais para me manter firme e determinado durante todo este período.

Agradeço ao meu orientador Prof. Dr. Roberto de Alencar Lotufo, pelo qual tenho profundo respeito, por ter me assistido e por ter coordenado meus passos ao longo desta jornada.

Agradeço ao colegas Alexandre Xavier Falcão e Antônio José Berutti Vieira pelo "empurrãozinho" e por me ajudarem a organizar as idéias. Ao Jorge Diz, sobretudo pelo suporte técnico. Ao Daniel Márcio Kligerman e ao Ulisses de Mendonça Braga Neto pela colaboração no desenvolvimento de alguns programas.

Agradeço aos demais colegas, professores e amigos que de alguma forma contribuíram para a concretização deste trabalho.

Agradeço à CAPES e à FAPESP pelo apoio financeiro, sempre útil e benvindo.

Dedicado àqueles que acreditaram e me fizeram acreditar...

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Ao longo das duas últimas décadas têm crescido as pesquisas e os resultados obtidos com relação à visualização, manipulação e análise de dados multidimensionais, baseados em ferramentas gráficas computadorizadas. Visualização é o processo relacionado a como a informação estrutural 3D pode ser apresentada na tela de um computador para um observador humano. Manipulação está relacionada com como as estruturas podem ser manipuladas. Análise, por sua vez, se relaciona com a quantização das estruturas. A medicina é a área onde se tem notado uma maior demanda no desenvolvimento e utilização destas ferramentas [38][37], mas obviamente não é a única. Aplicações em microscopia, modelagem molecular, astrofísica, geofísica, química, engenharia e demais campos onde a necessidade de melhor entender estruturas com mais de duas dimensões são desafios constantes, também se apoiam cada vez mais na utilização destas ferramentas gráficas.

Existem muitos aspectos computacionais envolvidos no tratamento de dados tridimensionais. A princípio, é desejável que os dados 3D sejam processados e visualizados "instantaneamente", em resposta à interação do usuário, como por exemplo, quando um objeto é rotacionado na tela de um terminal conforme indicado por movimentos do *mouse*. Taxas de *display* de imagens desta forma são conhecidas por taxas de *display* em tempo real, definidas por serem maiores ou iguais à taxa de fusão de imagens do olho humano, que é aproximadamente 30 quadros (*frames*) por segundo. No caso particular de dados tridimensionais, o problema é agravado devido à grande quantidade de dados (tipicamente entre 2 e 35MB) a serem processados. Como conseqüência, normalmente existe um compro-

misso entre velocidade de *rendering* (termo de uso geral empregado para descrever a criação da representação de uma estrutura ou volume na tela de um computador, através de uma seqüência de operações) e nível de refinamento, que geralmente requer maior quantidade de memória. As primeiras tentativas de alcançar taxas de *display* interativas esbarraram em arquiteturas de multiprocessadores de uso específico bem como em implementações de algoritmos baseados em *hardware* [23]. Embora estas arquiteturas produzam imagens rapidamente, o fato dos algoritmos serem implementados em *hardware*, somado ao contínuo desenvolvimento do setor, fazem com que implementações mais flexíveis destes algoritmos se tornem desejáveis, uma vez que facilitam a incorporação das melhorias alcançadas para os algoritmos no processo de *rendering*. Com o avanço da tecnologia, processadores com alta capacidade de processamento e endereçamento de memória têm se tornado cada vez mais baratos e comuns, incentivando, desta maneira, o uso de algoritmos implementados em *software*.

Pacotes gráficos para atuar em dados tridimensionais constituem atualmente importantes ambientes de aplicações de uso específico. São exemplos destes *softwares*, ANALYZE da *CEMAX Inc.* [24], SOFTVU [22] e 3DVIEWNIX [33][35][34] da *University of Pennsylvania*, e tantos outros, como os relacionados em Herman [11]. A desvantagem apresentada por eles está na dificuldade de expansão, visando o tratamento de novas situações que possam emergir das pesquisas constantemente realizadas.

A mais nova geração de ambientes sofisticados para a visualização de dados científicos inclui uma linguagem visual baseada no fluxo de dados, que fornece uma interface gráfica na qual operadores pré-definidos são usados para importar, manipular e exibir os dados [39]. A grande vantagem destes ambientes está no elevado grau de interação e na possibilidade de expansão. As aplicações têm a forma de um diagrama de blocos executável, constituído por ícones que representam os operadores. A seleção e conexão destes operadores para a construção dos diagramas de blocos são feitas de forma interativa utilizando o *mouse*. Isto facilita muito no desenvolvimento de protótipos que solucionem problemas complexos de visualização, manipulação e análise de dados. Entre estes sistemas pode-se destacar o Khoros da *University of New Mexico* [20], AVS da *AVS Inc.* [36], Explorer da *Silicon Graphics* [2], apE da *Tara Visual* [3] e Visualization Data Explorer da *IBM* [16]. Destes, o único que pode ser classificado como um pacote aberto e de livre acesso é o Khoros. A sua infraestrutura conta com um ambiente de desenvolvimento de interface com o usuário, que

assiste na geração de código e no desenvolvimento e manutenção de *softwares*, facilitando a expansão do sistema de forma ordenada.

1.1 Objetivos

Estando na versão 1.05, o Khoros, embora possua formas de representar dados com mais de duas dimensões, apresenta poucos operadores para o tratamento destes dados . A proposta desta dissertação é apresentar um conjunto básico de operadores para estender o sistema Khoros, que possibilite a visualização, manipulação e análise de dados tridimensionais. Como característica comum destes novos operadores, eles devem ser capazes de permitir a construção de ferramentas e aplicações mais complexas, que sejam flexíveis o suficiente para que possam se adaptar a novas situações.

Este trabalho está inserido num contexto maior de Visualização, Modelagem, Manipulação e Análise de Dados Multidimensionais do Grupo de Computação de Imagens do Depto. de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da UNICAMP.

1.2 Materiais

Grande parte das discussões apresentadas aqui têm por base o artigo de Udupa e Gonçalves, intitulado *"Imaging Transforms for Visualization Surfaces and Volumes"* [32]. As implementações foram feitas em linguagem C, com o auxílio das ferramentas de desenvolvimento e manutenção de *softwares* do sistema Khoros, que também contribuiu com o ambiente de programação visual como plataforma de teste das rotinas individualmente e em conjunto com outras. Durante todo o processo foram utilizadas estações de trabalho Sun SPARCstation (1+, 2, classic e SparcServer 370) com o sistema operacional Sun OS 4.1.3_V1; IBM Risc System/6000 (Model 360 e Model 590) com o sistema operacional AIX 3.2.5. Os dados 3D utilizados foram obtidos nos tomógrafos de raios-x CT 9800 do Departamento de Radiologia da Faculdade de Ciências Médicas da UNICAMP e do *MIPG - Dept. of Radiology, University of Pennsylvania, Philadelphia.*

1.3 Estrutura do Trabalho

O trabalho está dividido da seguinte maneira: o capítulo 2 apresenta o sistema Khoros, descrevendo as partes que o compõem, bem como suas características, e introduzindo alguns conceitos e definições relacionados ao ambiente de programação visual utilizado. No capítulo 3 são descritas, uma a uma, as rotinas implementadas para o processamento de dados tridimensionais, e também a forma com que elas foram agrupadas e adicionadas ao sistema Khoros para ampliá-lo. O capítulo 4 mostra alguns experimentos e seus resultados, obtidos através da combinação dos operadores desenvolvidos e dos já existentes. Ainda neste capítulo, são apresentados dois exemplos de rotinas gráficas de alto nível, uma concebida a partir de um operador básico, e outra que faz uso dos dados processados por outros operadores, também básicos, e que desempenha funções de aplicação mais especializada, ilustrando a praticidade em utilizá-las em áreas de interesses particulares. Sugestões de implementações e trabalhos futuros são colocados no capítulo 5. A dissertação termina com um apêndice, onde as interfaces gráficas dos operadores com o usuário são mostradas, seguido por último das referências bibliográficas utilizadas.

Capítulo 2

O Sistema Khoros

2.1 Introdução

Embora tenha sido originalmente criado para pesquisa em processamento de imagens [21], o Khoros é usado atualmente como uma ferramenta de desenvolvimento e pesquisa em diversos campos de atuação, como por exemplo em aplicações incluindo visualização, manipulação e análise de imagens médicas (*medical imaging*), análise de ecossistemas, sensoriamento remoto, sistemas de informações geográficas, gerenciamento de banco de dados e sistemas de controle. Esta extensa faixa de atuação se deve à disponibilidade de acesso às bibliotecas que incluem álgebra de matrizes, processamento de imagens, reconhecimento de padrões, processamento de sinais e processamento de superfícies, e ainda os programas interativos de aplicações para visualizar e editar dados uni-dimensionais, bi-dimensionais e tri-dimensionais, embora o Khoros não tenha ferramentas para lidar com volumes de forma mais abrangente, pois os dados tri-dimensionais são tratados como um conjunto de *arrays* bi-dimensionais, sem qualquer relação entre eles.

O sistema, que conta com padrões existentes X Windows e UNIX, tem uma infraestrutura que consiste de várias camadas de sub-sistemas que se interagem. Ele incorpora um sistema de desenvolvimento de interface com o usuário (UIDS - *User Interface Development System*) composto por geradores de código e ferramentas interativas de desenvolvimento e manutenção de *softwares*, muito úteis na criação de novas rotinas, e uma interface de acesso às bibliotecas da linguagem de programação visual chamado **Cantata**

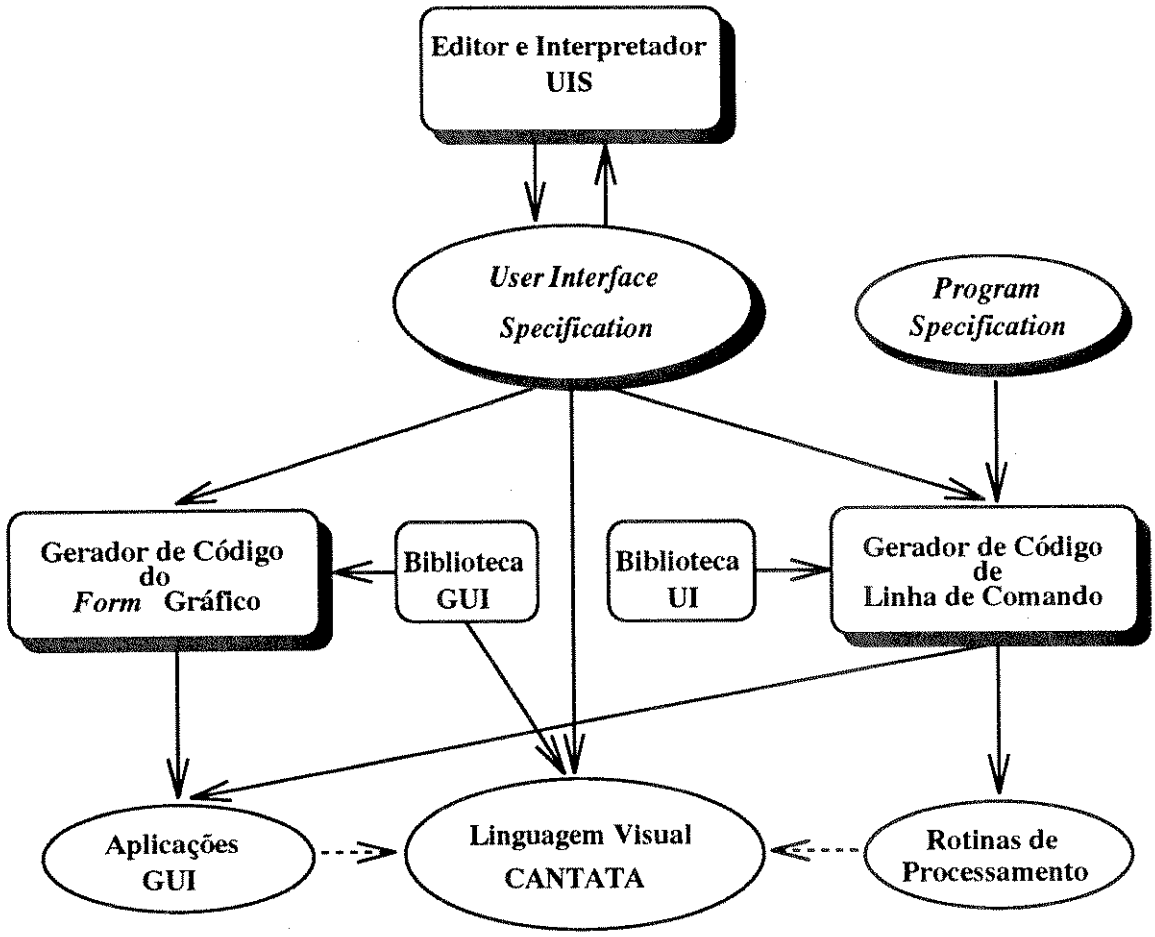


Figura 2.1: estrutura dos softwares do sistema Khoros

[19][20]. A figura 2.1 ilustra a relação entre estes elementos. Nela, os retângulos sombreados representam ferramentas específicas do UIDS; retângulos normais são as bibliotecas de funções gerais e de desenvolvimento; formas ovaladas sombreadas indicam entradas para as ferramentas do Khoros e formas ovaladas normais identificam os programas gerados. As linhas pontilhadas significam que as aplicações desenvolvidas usando o Khoros podem ser referenciadas pelo **Cantata**. As aplicações são rotinas de processamento específico acessadas por uma interface gerada automaticamente baseada numa especificação de interface com o usuário (UIS - *User Interface Specification*). Três tipos de interface são possíveis: gráfica, textual e através da linguagem visual do **Cantata**, comumente mais adotada. A interface gráfica é baseada em *forms* e menus, orientada a *mouse* e roda em *workstations* sob o sistema X Windows. A interface textual é baseada em linha de comando e orientada a teclado e pode ser executada em computadores usando o sistema operacional UNIX. A linguagem visual permite acesso aos programas que usam ambos os tipos de interface, seja ela gráfica ou textual.

A ênfase do Khoros é dada na formação de um ambiente de processamento de imagens, fornecendo o suporte necessário para estender a biblioteca de rotinas conforme a necessidade. Atualmente existem mais de 200 rotinas com funções de operações aritméticas, manipulações geométricas, conversões, transformações, filtros, classificação, etc. A familiarização com o conjunto de operadores do sistema, permite que novos programas compatíveis com a interface e com o formato da estrutura de dados do Khoros (VIFF - *Visualization Image File Format*) possam ser facilmente gerados e incorporados ao sistema. A forma tradicional com a qual isto é feito é através da criação de *toolboxes*. Entre os formatos de arquivos suportados pelas rotinas de conversão do Khoros, tanto para importação quanto para exportação, se encontram a maioria dos formatos padrões existentes: raster, X11 bitmap, TGA, FITS, PBM, TIFF e matriz.

A seção 2.2 apresenta o mais poderoso componente do Khoros, o ambiente de programação visual **Cantata**, e na seqüência a seção 2.3 dá destaque ao ambiente de desenvolvimento e manutenção de *softwares*. A seção 2.4 encerra o capítulo com uma visão geral do sistema Khoros. Maiores informações sobre o sistema podem ser adquiridas no manual no Khoros [14].

2.2 Ambiente de Programação Visual Cantata

Cantata é uma linguagem que emprega um modelo de programação orientada a fluxo de dados, graficamente expressa, e que provê um ambiente de programação visual dentro do sistema Khoros. Fluxo de dados é uma abordagem naturalmente visível, na qual um programa é descrito como um grafo direcionado, onde cada nó representa um operador (ou função) e cada arco direcionado representa um caminho sobre o qual os dados fluem. A execução de cada nó é habilitada quando todos os dados estiverem disponíveis nos seus arcos de entrada. Após processado, o dado resultante é colocado no arco de saída e flui para os próximos nós a ele conectados, dando seqüência ao processamento. Desta maneira, as execuções vão acontecendo conforme os dados se tornem disponíveis a operadores sucessivos. Este modelo de execução utilizado pelo **Cantata** é chamado de modelo orientado a dados (*data-driven*), em oposição à outra forma conhecida por *demand-driven*, ou orientado a demanda, onde a ordem de execução é determinada pela necessidade de dados, ao invés da disponibilidade.

Os elementos gráficos da linguagem visual consistem de uma *workspace*, *forms*, *glyphs* e conexões. Estruturas de controle (*if-then-else*, *while-loop*, *count-loop*, etc.), um interpretador de expressões e a possibilidade de criação de procedimentos visuais estendem a funcionalidade da metodologia de fluxo de dados. A *workspace* do **Cantata** consiste de um grande canvas sobre o qual o programa de aplicação é montado. Com exceção de especificações de alguns parâmetros para as rotinas, toda interação com o **Cantata** é feita através de movimentos do *mouse* e acionamentos de botões. Pode-se salvar uma *workspace* juntamente com uma seção de programação no **Cantata** para posterior restauração. A *master form* do **Cantata** (localizada acima e a esquerda da *workspace*) é usada para iniciar a interação de nível mais alto do **Cantata** com o usuário. Botões de interação geral são encontrados ao longo do lado esquerdo da *master form*. As rotinas estão agrupadas, segundo suas funções, em categorias (*subforms*) representadas por um menu que pode ser acessado selecionando o botão correspondente no topo da *master form*. A escolha de uma destas *subforms* provê o usuário com uma lista das rotinas disponíveis naquela categoria. Ao ser selecionada uma rotina, é mostrado no lado direito da *subform*, um painel onde os parâmetros da rotina podem ser fornecidos. O botão GLYPH, quando acionado, substitui a *subform* por uma representação mais compacta da rotina, chamada de *glyph*. O *glyph* é

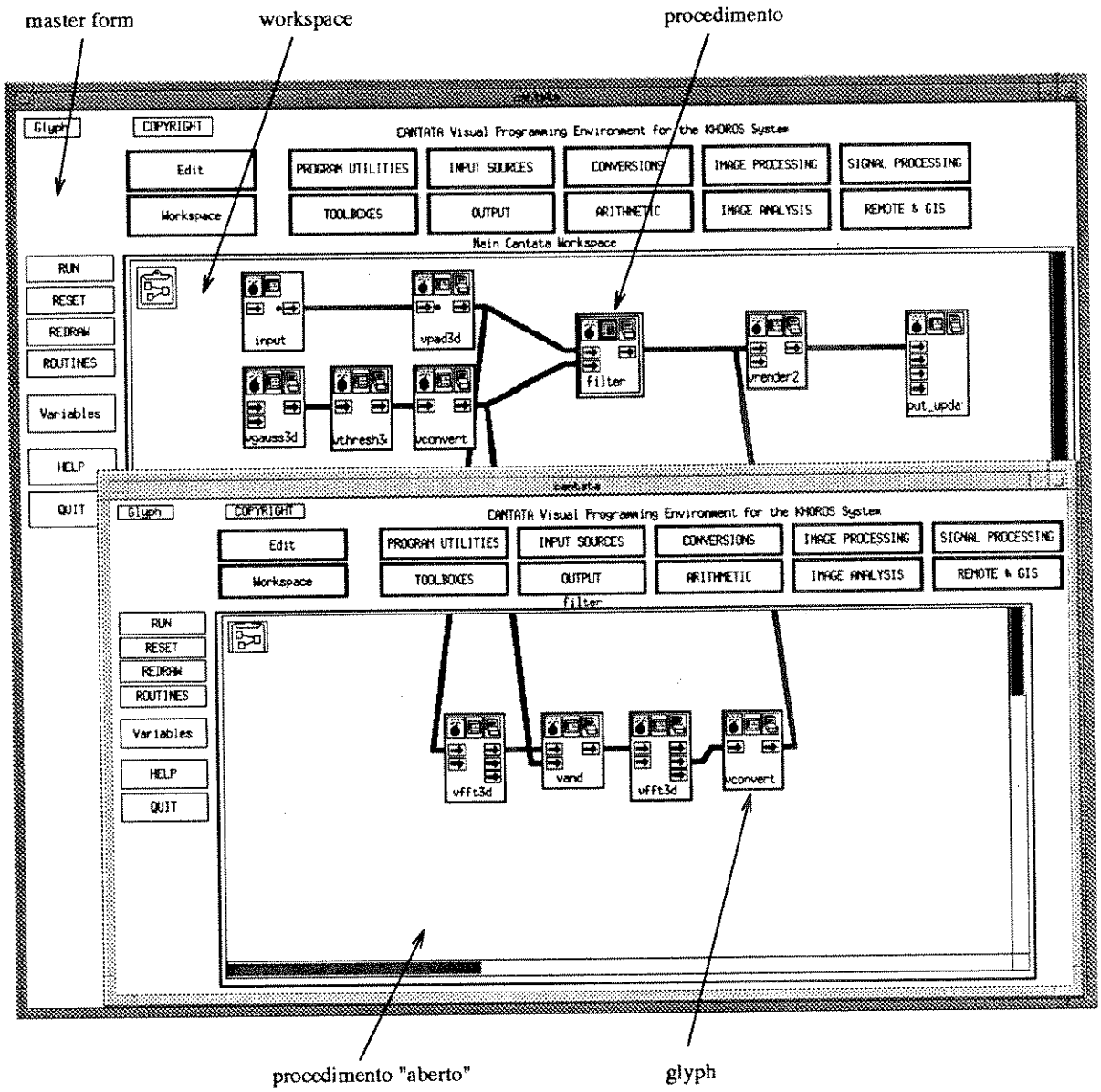


Figura 2.2: agrupamento hierárquico de workspaces

um pequeno *icon* que pode ocupar qualquer posição dentro da *workspace*. As entradas e saídas de cada *glyph* são representadas por botões dispostos dos lados esquerdo e direito, respectivamente. Os três botões no topo do *glyph*, da esquerda para a direita, são usados para destruir o *glyph*, reabrir a *subform* e executar a rotina correspondente.

O programa de aplicação é construído na forma de um grafo direcionado de fluxo de dados, onde os *glyphs* estão associados aos nós e os arcos correspondem às conexões feitas entre as entradas e saídas dos *glyphs* através do *mouse*. O **Cantata** associa automaticamente um nome para cada conexão. Estes nomes podem ser implementados via arquivos em disco ou via sistema de arquivos de memória compartilhada do Khoros. O monitoramento visual do processo também é possível, e é feito através da mudança de cor dos *glyphs* conforme os operadores a eles associados estejam em execução. O motivo do monitoramento é auxiliar o programador no acompanhamento do estado da execução do processo em andamento. Agrupamentos hierárquicos de *workspaces* podem ser usados para gerenciar a complexidade visual, criando procedimentos visuais, também chamados de macros, com a finalidade de adicionar estrutura à linguagem, da mesma maneira que subrotinas e procedimentos são usados em linguagens convencionais (veja figura 2.2). O *glyph* de um procedimento tem bordas duplas e, quando aberto, se apresenta na forma de uma *workspace* de subrotinas, contendo os tradicionais *glyphs*.

O interpretador de expressões é usado para definir as variáveis da *workspace* e avaliar o resultado das expressões. Expressões podem ser usadas em qualquer parâmetro de entrada nos painéis das rotinas. O interpretador pode ser acessado pelo botão rotulado VARIABLES no lado esquerdo da *master form*. O escopo das variáveis é definido de acordo com o local da declaração, dentro da hierarquia de *workspaces*. O interpretador de expressões suporta um amplo conjunto de bibliotecas matemáticas da linguagem C e UNIX, incluindo operadores lógicos e aritméticos, constantes e funções matemáticas.

Na posição superior da *master form* do **Cantata** existe um botão rotulado TOOLBOXES que dá acesso a uma categoria especial de *subforms*. A particularidade destas *toolboxes* deve-se ao fato de que novas rotinas são normalmente incorporadas ao ambiente do **Cantata** através das *toolboxes*.

2.3 Ambiente de Desenvolvimento e Manutenção de Softwares

O UIDS é um conjunto de ferramentas do sistema Khoros que oferece meios para definir a interface com o usuário e gerar o código referente a esta interface para o programa de aplicação. O desenvolvimento e manutenção dos programas criados para integrar o sistema Khoros pode ser visto sob dois aspectos diferentes, porém interligados. O primeiro deles diz respeito à interface com o usuário e o outro à aplicação propriamente dita. Embora existam três tipos de interface disponíveis (gráfica, textual e através de linguagem visual), aplicações usando as duas primeiras são programas completos, cujos códigos, com exceção da parte funcional da aplicação, foram totalmente gerados pelo UIDS. Ambas as interfaces são completamente descritas pelo arquivo de especificação de interface com o usuário (UIS - *User Interface Specification*). No caso do interfaceamento gráfico, existe ainda uma biblioteca do Khoros, a GUI (*Graphical User Interface*), que fornece os meios para este tipo de interface. A biblioteca GUI pode ser também usada para interpretar um arquivo UIS para criar uma interface gráfica com o usuário em tempo de execução. O **Cantata** faz uso deste modo interpretativo. Se a interface textual foi utilizada para gerar uma rotina, o **Cantata** pode interpretar o mesmo arquivo UIS e apresentar uma interface gráfica na forma de um painel ou um *glyph*. A execução da rotina pelo **Cantata** é feita através da geração da linha de comando correspondente, de acordo com os parâmetros do painel, seguida da criação de um processo filho fazendo um *fork* do comando.

A especificação contida no arquivo UIS contém informação relacionada à aparência física e à funcionalidade da interface da aplicação. A especificação é um arquivo ASCII composto por uma seqüência de linhas, seguindo uma estrutura sintática, cada qual descrevendo uma parte da interface com o usuário. Agindo como uma descrição de alto nível, a especificação é usada para gerar o código para as interfaces gráficas e de linha de comando. Além disso, a UIS representa uma especificação formal no diálogo entre o usuário final e a aplicação. A elaboração de um arquivo UIS pode ser feita de duas maneiras. Na primeira delas, um editor gráfico de UIS oferecido pelo sistema Khoros pode ser usado para escolher interativamente as opções apropriadas para a desejada interface. Conforme as escolhas são feitas, elas vão sendo automaticamente traduzidas para o formato do arquivo UIS e em seguida interpretadas pela GUI que mostra os resultados na tela. Uma outra

maneira é usar um editor de textos para gerar o arquivo UIS diretamente.

Os geradores de código servem para auxiliar o programador, oferecendo ajuda nas tarefas que são invariantes entre as aplicações. Isto inclui geração de código para extração de dados das interfaces com o usuário, criação de estruturas em linguagem C para intermediar o programa de aplicação e a interface, e geração do código utilizado para iniciar estas estruturas apropriadamente. Também são geradas as páginas de manual devidamente formatadas.

Além do UIS, existe um outro arquivo chamado PS (*Program Specification*), que serve de entrada ao gerador de código de linha de comando para que ele possa gerar o programa completo e a rotina de biblioteca. Neste arquivo, o programador da aplicação provê informações adicionais, que não podem ser incluídas no arquivo UIS, como por exemplo, sintaxe para chamadas à rotina de biblioteca, segmentos de código que devem aparecer no programa de aplicação, documentação do programa e da rotina de biblioteca, exemplos, etc..

2.4 Conclusão

O sistema Khoros, composto por diversos sub-sistemas, tem uma área de atuação bastante geral e abrangente. Para suprir necessidades específicas, o sistema oferece meios de desenvolvimento e manutenção de novos operadores, para que possam ser incorporados a ele, através da utilização de arquivos de especificação de interface com o usuário em alto nível e arquivos de especificação de programa propriamente dito. A confecção destes arquivos pode ser assistida por editores especiais, possibilitando uma maior rapidez e organização no desenvolvimento do novo *software*, de forma que os padrões utilizados pelo Khoros sejam respeitados, mantendo o sistema íntegro. O uso das ferramentas de desenvolvimento e manutenção de *softwares* garantem também o uso da nova rotina pelo ambiente de programação visual **Cantata**, um dos componentes principais do sistema Khoros. Neste ambiente, programas de aplicação podem ser montados, utilizando os diversos elementos da linguagem visual. Para criar uma aplicação no **Cantata**, o usuário seleciona as rotinas desejadas e estruturas de controle, conforme a necessidade, posiciona os correspondentes *glyphs* na *workspace* e os conecta para indicar o fluxo do processamento de rotina para ro-

tina. Novas rotinas podem ser organizadas em grupos, de acordo com sua funcionalidade, e constituírem uma *toolbox*. As *toolboxes* permitem acesso a estas rotinas por um menu especial no ambiente do **Cantata**. A possibilidade de interfaceamento por linha de comando permite que sejam escritas *shell scripts*, concentrando *workspaces* completas ou parte delas, em unidades de processamento mais especializados.

Capítulo 3

Toolbox V3DTOOLS

3.1 Introdução

Como uma *toolbox* para o sistema Khoros, V3DTOOLS teve origem com a tese de Mestrado "Visualização de Volumes Aplicada à Área Médica", por Alexandre Xavier Falcão, em 1993 [4]. Seu trabalho apresenta um tutorial sobre os principais métodos utilizados na visualização de dados biomédicos 3D, bem como algumas implementações relacionadas com a visualização volumétrica. Inicialmente com apenas algumas rotinas, mais precisamente quatro, a *toolbox* evoluiu desde então, até se constituir no que é hoje. Esta evolução diz respeito não somente ao crescimento no número de rotinas que a compõem, mas principalmente na sua filosofia. Contando com 24 rotinas, classificadas em 7 grupos de acordo com as funções desempenhadas, a *toolbox* V3DTOOLS é, além da extensão de 2D para 3D de algumas rotinas integrantes do sistema Khoros, um conjunto de operadores básicos para visualização, manipulação e análise de dados tridimensionais.

O destaque da V3DTOOLS vem da concepção de operações básicas para posteriormente serem combinadas e compor operadores mais complexos, bem como novas metodologias. Geralmente, operações funcionalmente independentes são integradas entre si ou embutidas em algum método, como por exemplo no *rendering* em visualização biomédica, por questões de eficiência computacional. Porém, quebrando funções e métodos em pedaços menores, de forma a vê-los como um conjunto com propósito mais generalizado, pode-se chegar a resultados funcionalmente poderosos, através de combinações destas partes, antes

não percebidas por estarem justamente encapsuladas. Udupa e Gonçalves [32] tratam de maneira bem detalhada esta abordagem, chegando a definir operadores que se relacionam com as diversas transformações básicas, normalmente usadas para processar dados 3D, e criando uma linguagem de caráter matemático para compor operadores mais complexos. Obviamente a riqueza dos resultados conseguidos, que serão apresentados posteriormente no capítulo 4, depende da identificação e combinação apropriadas dos operadores básicos.

Para melhor entender o contexto no qual a V3DTOOLS está inserida, é necessário se ter uma visão geral dos diversos tipos de representação de dados 3D conhecidos na literatura e suas implicações. Entre elas pode-se citar os métodos de descrição por contorno, superfície e volume. Modelagens por contorno e superfície [1][13] proporcionam *renderings* rápidos através da representação das superfícies dos objetos 3D por primitivas de dimensões menores (1D e 2D). Desta forma, a quantidade de memória necessária para armazenar tal informação é muito reduzida, mas o preço a ser pago é o de permitir somente um tipo de *rendering*, chamado *rendering* de superfície. Além disso, há a necessidade de processar o volume para extração das superfícies a cada vez que, de alguma forma, ocorre mudança no objeto, como cortes por exemplo. Os contornos são obtidos individualmente de cada fatia do volume de maneira que a borda em cada fatia seja representada por um conjunto de pontos conectados por uma linha direcionada [30]. As primitivas 2D para a representação por superfícies usam geralmente triângulos ou outros polígonos. Existem duas classes principais de algoritmos para modelar a superfície de um objeto: *tiling* [6] e *surface tracking* [29]. Os métodos de *tiling* determinam a superfície do objeto de interesse extraindo os contornos das fatias e unindo-os com um conjunto de primitivas 2D. Os métodos de *surface tracking* usam as faces dos *voxels* como primitivas 2D. A superfície é determinada pelo conjunto das faces dos *voxels* conectados que façam parte da superfície do objeto de interesse. Representações por volume usam todo o conjunto de dados 3D diretamente, sem representações intermediárias, por isto suportam *renderings* de superfície e de volume, capazes de gerar imagens de volumes com vários objetos (pele, osso e gordura, por exemplo), geralmente usando efeitos de semi-transparência e cor. Estes métodos sofrem a desvantagem de terem que manipular grandes quantidades de dados. As primitivas 3D usadas normalmente são o *voxel* ou a *octree* [18][17]. *Voxel*, ou *volume element*, é a unidade 3D análoga a um *pixel*. *Voxels* são igualmente dimensionados nas três direções e tem a forma de pequenos paralelepípedos originados pela divisão do espaço por um conjunto de planos paralelos a cada eixo

do espaço. Desta forma o *voxel* passa a ser a unidade de representação mais imediata, pois o espaço torna-se um *array* tridimensional composto por *voxels*, cujos índices correspondem às suas coordenadas e cujos valores correspondem às densidades dos mesmos. A estrutura de dados *octree* é uma estrutura em árvore, onde o nó raiz representa o volume como um todo, e cada um dos seus oito filhos são formados por sub-divisões recursivas do volume do nó pai em octantes. Os sub-volumes dos octantes vão sendo divididos até que um critério de parada seja satisfeito (geralmente complexidade do volume representado pelo nó ou medida do volume representado pelo nó).

Originalmente as rotinas da *toolbox* V3DTOOLS foram desenvolvidas tendo em vista uma aplicação voltada à área médica, porém não estão restritas a esta única aplicação, pois são capazes de tratar conjuntos de dados 3D representados pelo método de volume usando o *voxel* como primitiva. A representação baseada em volumes tem crescido em popularidade devido à sua habilidade de fornecer meios para computar os *renderings* de volume e superfície, sem haver necessidade de realizar um pré-processamento de extração da superfície. O formato de arquivos utilizado pelo Khoros, chamado VIFF, possui várias maneiras de representação do *array* de *voxels*. A forma adotada associa a direção x às colunas, a direção y às linhas e a direção z às bandas (fatias). Por simplificação, o conjunto de dados 3D é chamado de volume ou cena neste trabalho.

Neste capítulo é dada uma visão geral das rotinas que compõem a versão atual da *toolbox* V3DTOOLS (seção 3.2), seguida de uma breve descrição das diferenças entre a versão atual e a primeira versão (seção 3.3).

3.2 Versão Atual

Utilizando técnicas baseadas em *voxel*, as 24 rotinas existentes na atual *toolbox* V3DTOOLS estão classificadas e agrupadas dentro de 7 grupos funcionais e são apresentadas a seguir:

1. rotinas de transformações geométricas
   - vinter
   - vexpand3d

2. rotinas de rendering
   - vzbuff
   - visnorm
   - vvsnorm
   - vvoxext
   - vtextu
   - vshad
3. rotinas de processamento puntual do voxel
   - vthresh3d
4. rotinas de processamento espacial
   - vqmed3d
   - vconv3d
5. rotinas de transformada digital de Fourier
   - vfft3d
   - vmpp3d
6. rotinas de manipulação de regiões
   - vinsert3d
   - vpad3d
   - vaband
   - vdband
   - vmband
   - vmontage
   - vplanexyz
7. rotinas de geração de volumes sintéticos
   - vframe

- vgauss3d
- vimpul3d
- vorthog

Os mesmos 7 grupos funcionais podem ser notados pela distribuição física das rotinas no menu da *subform* da *toolbox* V3DTOOLS, que dá acesso às rotinas que a compõem. A *subform* pode ser vista na figura 3.1, tendo em particular, a rotina `vgauss3d` selecionada.

A seguir são apresentadas descrições mais detalhadas de cada uma das rotinas citadas acima, acompanhadas de figuras ilustrando suas entradas e saídas. Muitas destas figuras foram obtidas através da utilização de rotinas da própria V3DTOOLS. Os *forms* das rotinas com os parâmetros *default* estão no apêndice A, ao final da dissertação.

3.2.1 Rotinas de Transformações Geométricas

3.2.1.1 vinter

A rotina `vinter`, aplicada ainda na fase de pré-processamento, converte um volume 3D em outro volume 3D com *voxels* cúbicos, através da interpolação das fatias intermediárias do volume. A interpolação, neste caso, é feita apenas na direção z, ou seja, entre fatias, devido às características dos dados adquiridos por equipamentos médicos, como por exemplo, ressonância magnética (*magnetic ressonance imaging* - MRI), tomografia por emissão de pósitron (*positron emission tomography* - PET), tomografia computadorizada de raios-x (*x-ray computed tomography* - CT), tomografia computadorizada por emissão de fóton (*single photon emission computed tomography* - SPECT), etc.. Este tipo de dado se apresenta na forma de um conjunto de fatias 2D, divididas em *pixels* de iguais dimensões em x e y, cujos valores representam alguma característica física de um pequeno espaço tridimensional em torno de seus centros. A necessidade da interpolação destes dados ocorre porque a distância entre as fatias são geralmente maiores, quando comparadas com as dimensões dos pixels. Como opções do programa (veja *form* na figura A.1), `vinter` tem implementações das técnicas de interpolação apresentadas em Lotufo et al. [15], ou seja, interpolação de níveis de cinza (*gray level interpolation*), também conhecida por interpolação clássica, interpolação euclideana baseada na forma (*euclidean shape-based interpolation*) e

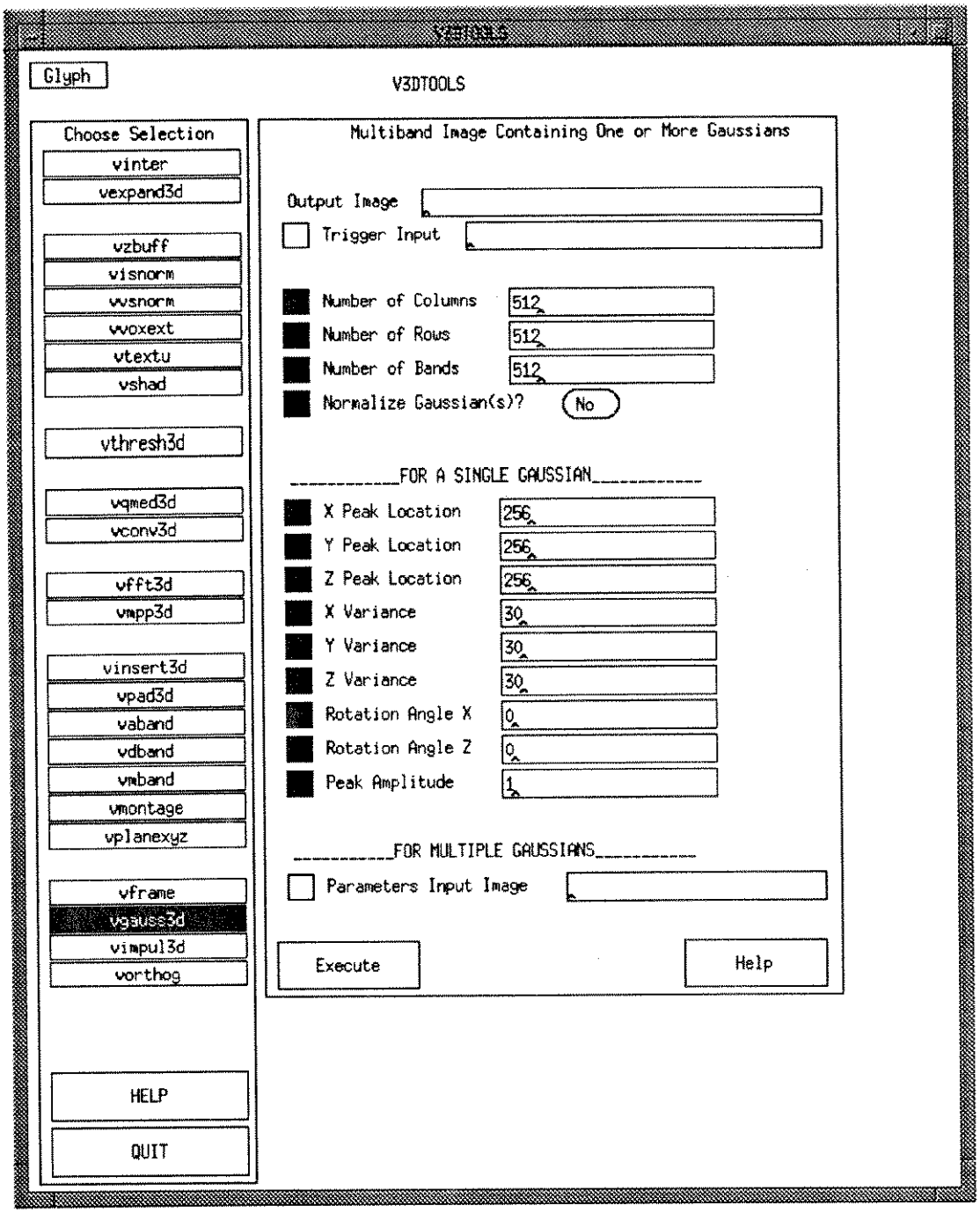


Figura 3.1: subform V3DTOOLS

combinação das duas últimas, bem como opções de escolha entre interpolação de ordem zero (replicação de fatias), interpolação linear e interpolação *spline* cúbica modificada. A interpolação de níveis de cinza é o meio clássico de se estimar as fatias intermediárias. A interpolação euclideana baseada na forma se aplica a dados segmentados, onde a informação de distância obtida para cada fatia é usada na estimativa das fatias intermediárias. Já a interpolação combinada, usa os dois métodos anteriores, tendo o gradiente local como o fator de normalização para poder combiná-los. Se o volume de entrada for em níveis de cinza, a rotina oferece meios para decidir o tipo de saída desejada (binária ou cinza). Se a escolha for uma saída binária, um processo interno de segmentação por *threshold* é realizado, sendo para isto necessário o fornecimento do valor de corte. *Threshold* é uma técnica de segmentação, ou seja, de extração de um objeto ou uma superfície de uma cena, que localiza os *voxels* que tenham propriedades comuns. Um critério bastante usado para determinar estes *voxels* com características comuns é o valor de corte ou janela de corte. Um *voxel* cuja densidade caia dentro dos valores da janela, ou acima do valor de corte é dito pertencer ao objeto, caso contrário é classificado como *background*. De acordo com o tipo dos dados de entrada e saída, a utilização de uma ou outra técnica é limitada segundo a tabela 3.1, apresentada a seguir.

		saída	
		binária	cinza
entrada	binária	euclideana baseada na forma	—
	cinza	clássica euclideana baseada na forma combinada	clássica

Tabela 3.1: técnicas de interpolação segundo o tipo dos dados de entrada e saída

Uma outra opção oferecida é a de poder escolher explicitamente o número de

fatias finais resultantes da interpolação. Caso este parâmetro seja omitido, a rotina é capaz de calcular o número de fatias finais a fim de gerar um espaço isotrópico (mesma resolução nas três dimensões), com base nas informações de dimensão dos *pixels* e de espaçamento entre as fatias encontradas no cabeçalho do arquivo de entrada.

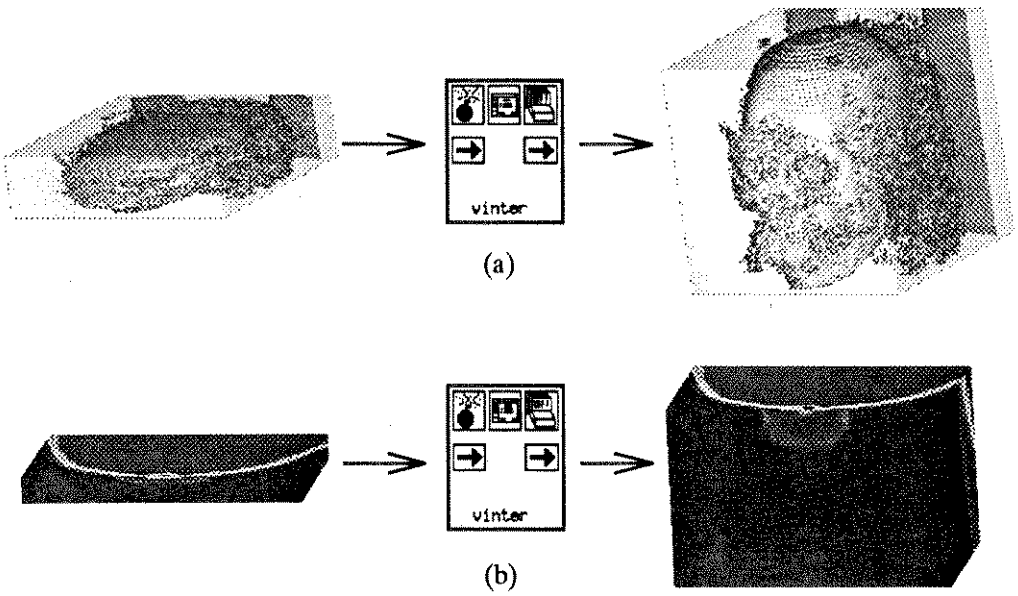


Figura 3.2: rotina vinter: (a) interpolação de crânio seco e (b) crânio de paciente

A figura 3.2 ilustra a interpolação de dois volumes amplamente utilizados na descrição da *toolbox* V3DTOOLS. Eles foram obtidos por tomografia computadorizada de raios-x e possuem características diferentes. O volume em 3.2a é um crânio inteiro e seco de dimensões 256x526x68, com *voxels* de 0.8x0.8x3.0 milímetros na entrada e 256x256x248 com *voxels* de 0.8x0.8x0.81 milímetros na saída; e o em 3.2b é parte de um crânio de um paciente cujas dimensões na entrada são 300x180x28 com *voxels* de 0.4x0.4x3.0 milímetros e 300x180x196 com *voxels* de 0.4x0.4x0.41 milímetros na saída. Os ruídos foram deixados propositalmente, para mostrar que a interpolação é aplicada ao volume por completo.

3.2.1.2 vexpand3d

`vexpand3d` age replicando os *voxels* de um volume, sendo portanto, uma interpolação de ordem zero. A diferença entre a interpolação da rotina `vexpand3d` e a da `vinter`

é que na primeira delas a replicação acontece não somente entre fatias (direção z), mas também entre colunas e linhas (direções x e y). A rotina oferece ainda, a possibilidade de escolha de fatores de expansão independentes para cada uma das direções, porém devem ser inteiros positivos e não nulos. A figura 3.3 mostra o resultado da expansão dos dados originais do crânio seco, com fatores 1 em x, 2 em y e 3 em z. Seu *form* se encontra na figura A.2.

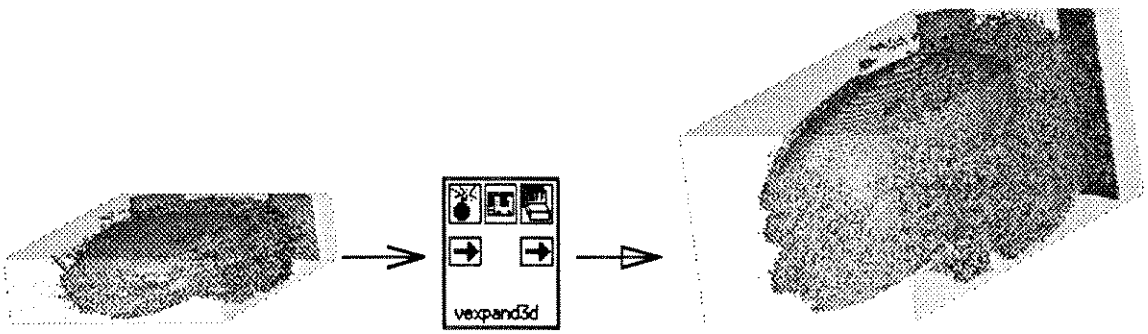


Figura 3.3: rotina vexpand3d

3.2.2 Rotinas de Rendering

3.2.2.1 vzbuff

A rotina `zbuffer` é responsável pela etapa de projeção e remoção de superfícies escondidas (*hidden surface removal*) no processo de *rendering*. Projeção é uma operação que mapeia pontos de um sistema de coordenadas n-dimensional para um outro sistema de coordenadas de dimensão menor que n, enquanto que remoção de superfícies escondidas se preocupa em fazer com que somente a parte do objeto que está visível, segundo uma posição de observação, seja efetivamente visualizada. Os algoritmos mais comuns para remoção de superfícies escondidas são *depth sort*, *z-buffer* [25, pág. 265–272], *back-to-front* [5] e *front-to-back* [9].

A `vzbuff` gera na saída três arquivos opcionais (veja figura 3.4). O primeiro deles é o arquivo contendo o *z-buffer*, ou seja, a distância de cada ponto (u, v) do plano de projeção aos *voxels* visíveis da superfície do objeto. A segunda saída consta de um arquivo com 3 bandas, representando as coordenadas tridimensionais dos pontos onde a superfície

foi detectada, tendo como referencial a própria origem do volume. A terceira e última saída possível é um arquivo contendo os valores (densidades) dos *voxels* nestes mesmos pontos. A utilidade de cada uma destas saídas será vista mais adiante.

Internamente, um algoritmo de *ray casting* [28] lança raios de cada ponto (*pixel*) do plano de projeção em direção ao objeto, para realizar a projeção e calcular qualquer uma das saídas desejadas. O método de *ray casting* tem intrínseco o algoritmo de remoção de superfícies escondidas. *Ray casting* é um caso particular da técnica *ray tracing* [25, pág. 296–305], onde somente o raio primário lançado de cada *pixel* do plano de projeção é levado em consideração, desprezando, portanto, os raios originados por reflexão, refração e transmissão nos pontos de intersecção com o objeto. Estes raios partem perpendicularmente ao plano e caminham sob um passo de valor estipulado por um parâmetro, até encontrar os *voxels* pertencentes à superfície do objeto. Na maioria dos casos, a superfície pode ser definida pelo conjunto de *voxels* maiores que um valor de limiar (*threshold*) fornecido como parâmetro. Neste caso a superfície encontrada não pode ser classificada como uma iso-superfície, pois não há garantia de que todos os *voxels* pertencentes à superfície possuam a mesma densidade. Para possibilitar diversas vistas, o plano de projeção pode ocupar qualquer posição em torno do objeto (dado por ângulos de rotação α e β, em torno dos eixos z e x respectivamente), bem como pode ser deslocado em direção ao objeto (parâmetro opcional estipulando a distância de corte) e entrar dentro dele, conseguindo com isto o efeito de cortes seqüenciais em direções diferentes das fatias originais (veja *form* na figura A.3). Quando um raio de luz atravessa todo o volume sem que o objeto tenha sido atingido, é atribuído um valor negativo ao ponto (u, v) correspondente, no caso do arquivo do *z-buffer* e do arquivo das coordenadas, e zero para o arquivo de densidade dos *voxels*. Além do passo, que indica o quanto se avança o raio de luz em direção ao objeto a cada interação, o que se traduz num compromisso de escolha entre rapidez e precisão, a rotina `vzbuff` possibilita também uma sub-amostragem no lançamento dos raios de luz ao partir do plano de projeção, oferecendo um processamento mais rápido, porém de qualidade inferior, desejado em casos onde a necessidade da visualização seja somente para revelar, grosseiramente, o conteúdo desconhecido de um volume. Este tipo de visualização é normalmente denominado *preview*.

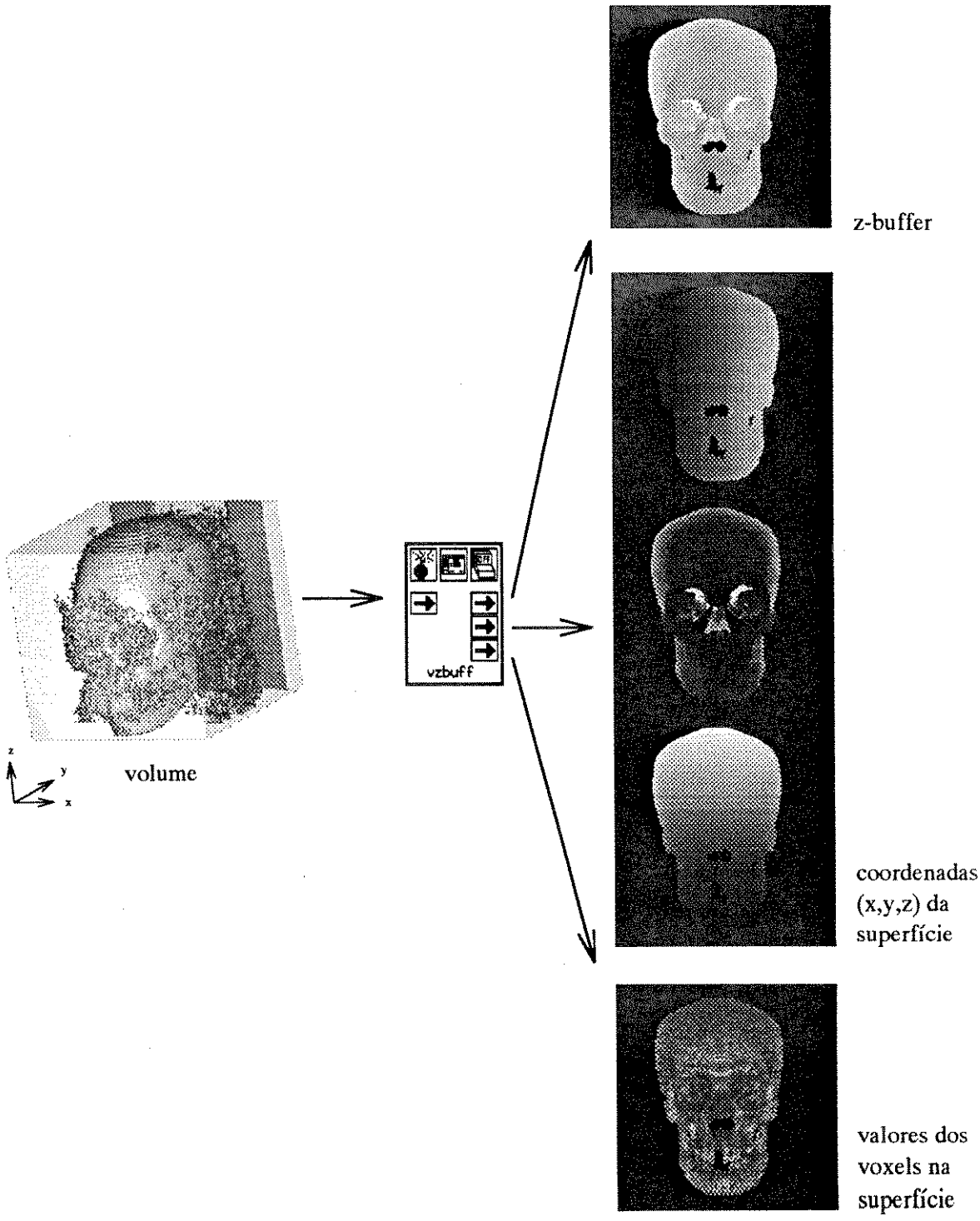


Figura 3.4: rotina vzbuff

3.2.2.2 visnorm

A rotina `visnorm` (veja figura 3.5), que era um processamento interno a `zbuffer` na versão anterior da *toolbox*, estima os vetores normais a cada ponto visível da superfície, segundo as informações de distância contidas no *z-buffer* (saída de `vzbuff`), através do método das diferenças centrais [7], que se constitui em uma aproximação dos vetores normais pelo gradiente das distâncias entre o *pixel* e seus vizinhos, conforme formulado na equação 3.1.

Método das diferenças centrais para estimativa das normais no espaço imagem:

$$\vec{N} = (N_u, 1, N_v) \qquad \begin{aligned} N_u &= \frac{d(u+1,v)-d(u-1,v)}{2} \\ N_v &= \frac{d(u,v-1)-d(u,v+1)}{2} \end{aligned} \tag{3.1}$$

onde:
$d(u,v)$ é a distância do *pixel* (u,v) à superfície do objeto (valor do *z-buffer* na posição (u,v))

Os vetores normais estimados desta maneira são chamados de vetores normais no espaço imagem (*image/view space normal vectors*) [9] e esta estimativa pressupõe a existência de continuidade espacial entre *pixels* vizinhos, o que quase sempre se verifica, com exceção das bordas e cumes, onde algumas anomalias podem ocorrer [31]. O espaço imagem é a área retangular contida no plano uv, proveniente da projeção do objeto. Ele está dividido em *pixels* quadrados e é particularmente representado pelo arquivo de *z-buffer* gerado em `vzbuff`. A posterior utilização das normais estimadas no espaço imagem vão caracterizar a tonalização no espaço imagem (*z-buffer gradient shading*) [27]. Como a rotina responsável pela tonalização requer vetores no referencial do objeto, os mesmos ângulos de rotação utilizados para a geração do *z-buffer* são necessários para que internamente seja feita a correção (veja o *form* na figura A.4). Ao final, os vetores são normalizados e cada uma das 3 componentes ocupa uma banda na saída. Para pontos não pertencentes ao objeto (valor do *z-buffer* negativo) é gerado o vetor nulo.

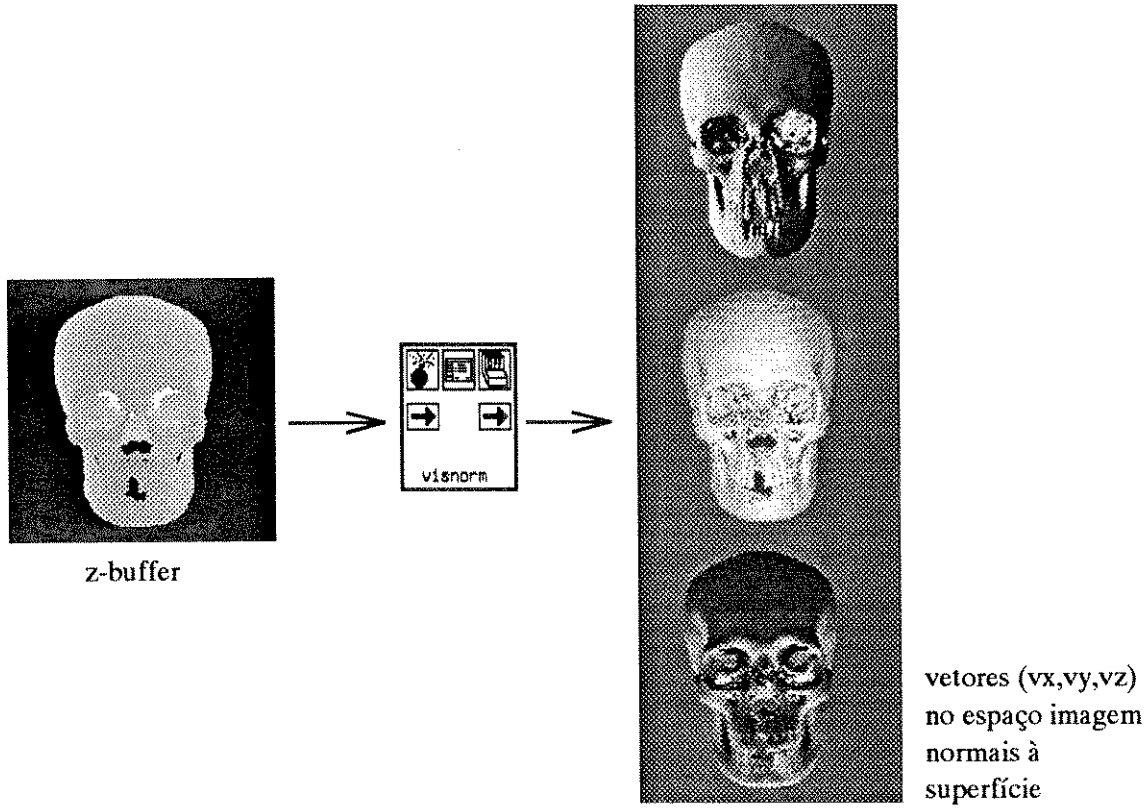


Figura 3.5: rotina visnorm

3.2.2.3 vvsnorm

Assim como `visnorm`, a rotina `vvsnorm` fazia parte de `zbuffer`. Para volume de dados em tons de cinza, ela estima os vetores normais no espaço *voxel* (*voxel/scene space normal vectors*), para todos os pontos visíveis da superfície do objeto [12], levando em consideração o gradiente das densidades dos *voxels*. O espaço *voxel* é definido como o cubóide que contém os *voxels* em tons de cinza, resultante do empilhamento das fatias coletadas por um dos métodos de aquisição citados. O arquivo de coordenadas gerado pela rotina `vzbuff` fornece as coordenadas dos pontos da superfície do objeto e pelo método das diferenças centrais [7] são calculadas as três componentes (direções x, y e z) dos vetores normalizados, de acordo com a densidade dos *voxels* na vizinhança do ponto no volume de entrada (veja equação 3.2).

Método das diferenças centrais para estimativa das normais no espaço *voxel*/objeto:

$$\vec{N} = (N_x, N_y, N_z) \qquad \begin{aligned} N_x &= \frac{f(x+1,y,z) - f(x-1,y,z)}{2} \\ N_y &= \frac{f(x,y-1,z) - f(x,y+1,z)}{2} \\ N_z &= \frac{f(x,y,z-1) - f(x,y,z+1)}{2} \end{aligned} \tag{3.2}$$

onde:
$f(x, y, z)$ é a densidade do *voxel* nas coordenadas (x, y, z)

Para *voxels* ocupando posições de borda, onde um ou mais vizinhos não existem, as componentes dos vetores são calculadas segundo o método das diferenças retrógradas (*backward difference*) ou diferenças progressivas (*forward difference*) [7], como mostrado

nas equações 3.3 e 3.4 respectivamente, para o caso particular da componente z.

Diferença retrógrada em z: quando o *voxel* está na última fatia (não existe coordenada $z+1$):

$$N_z = f(x,y,z) - f(x,y,z-1) \tag{3.3}$$

Diferença progressiva em z: quando o *voxel* está na primeira fatia (não existe coordenada $z-1$):

$$N_z = f(x,y,z+1) - f(x,y,z) \tag{3.4}$$

A tonalização usando normais estimadas no espaço *voxel* vão caracterizar o *gray level gradient shading* [27]. Para volumes binários, o resultado obtido é equivalente ao da estimativa no espaço objeto (*object space normal vectors*) [10] e é feito automaticamente (na primeira versão da *toolbox*, a escolha entre normais no espaço *voxel* ou normais no espaço objeto é feita de forma explícita). O espaço objeto é semelhante ao espaço *voxel*, porém seus *voxels* são binários, indicando que o objeto de interesse já tenha sido segmentado, o que explica o termo "espaço objeto". A estimativa do vetor normal no espaço objeto calcula a normal para um *voxel* visível usando a geometria dos voxels vizinhos ou suas faces. No caso de `vvsnorm` o cálculo é feito baseado na geometria dos *voxels* vizinhos, aproveitando as equações 3.2 a 3.4. Como em `visnorm`, a saída é composta por três bandas e a pontos não pertencentes ao objeto é atribuído o vetor nulo. As entradas e saídas da rotina `vvsnorm` podem ser vistas na figura 3.6, e seu *form* na figura A.5.

3.2.2.4 vvoxext

`vvoxext` extrai o valor dos *voxels* de um volume nas coordenadas dadas. Estas coordenadas são normalmente geradas pela rotina `vzbuff` (veja figura 3.7). Utilizando o arquivo de coordenadas da superfície de um volume para obter os *voxels* do mesmo volume, terá como resultado a saída de valor dos *voxels* que `vzbuff` geraria. A vantagem de `vvoxext`

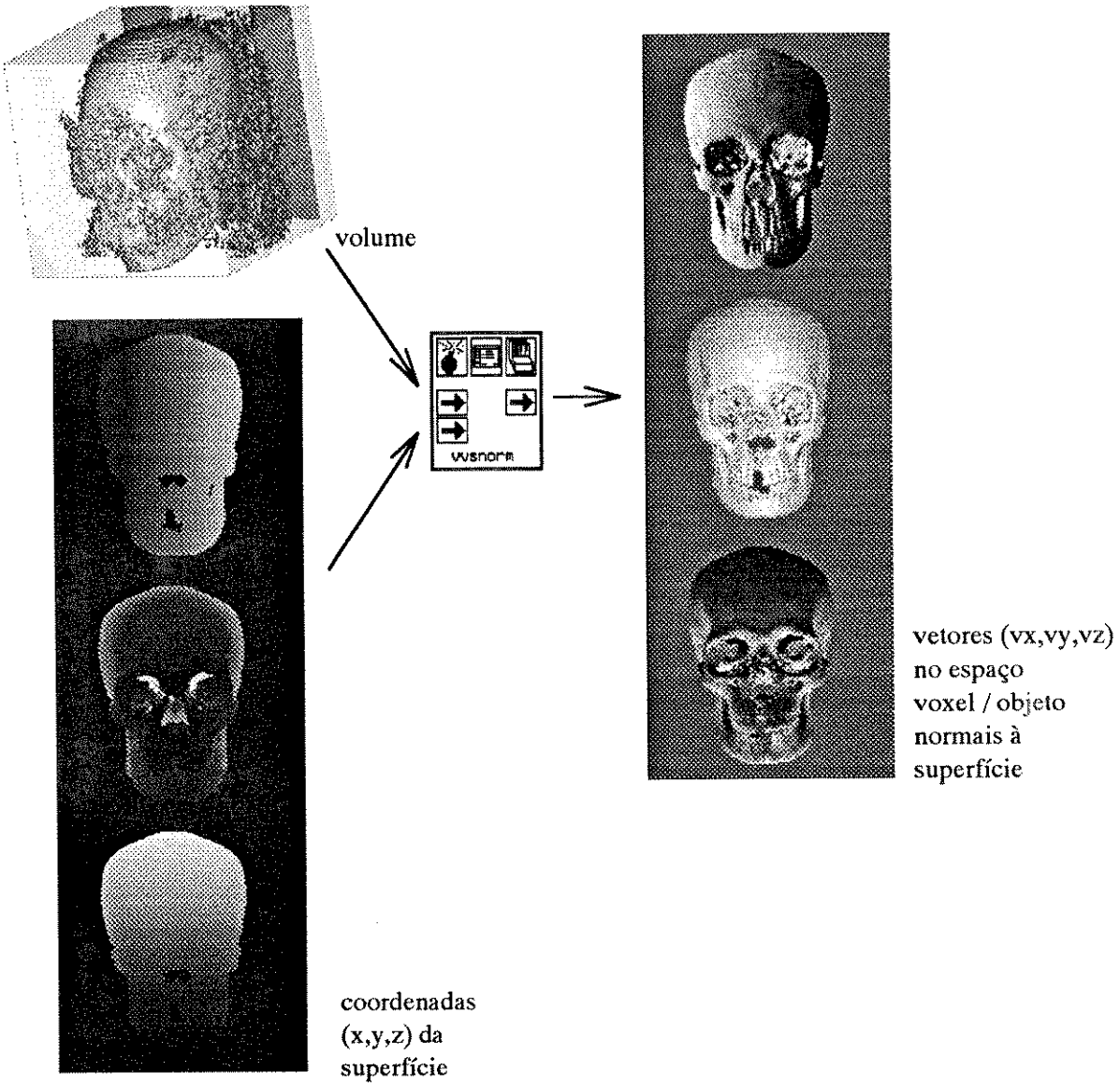


Figura 3.6: rotina vvsnorm

é poder combinar o arquivo de coordenadas de um volume para obter os *voxels* de um outro volume, como se fosse uma interseção. Coordenadas negativas correspondem a *background* e nestes casos são gerados *voxels* zero na saída. Exemplos da utilidade deste operador serão apresentados no capítulo 4. Seu *form* se encontra na figura A.6.

3.2.2.5 vtextu

Esta rotina gera informações adicionais sobre os dados na vizinhança da superfície, para posteriormente ser usada na tonalização do objeto pela rotina `vshad`. Partindo de cada ponto da superfície do objeto, que é dado pelo arquivo de coordenadas (obtido em `vzbuff`), a "textura", assim chamada para simplificação, é obtida caminhando-se pelo objeto na direção do observador (dada pelos mesmos ângulos α e β usados em `vzbuff`) ou caminhando-se pelo objeto na direção do vetor normal à superfície no mesmo ponto (normais podem ser obtidas em `visnorm` ou `vvsnorm`). Veja na figura 3.8 a ilustração destas duas possibilidades. O valor da "textura" pode ser obtido tomando-se o valor médio, valor ponderado, valor máximo ou valor mínimo dos *voxels* visitados durante o percurso e, nestes casos, deve ser estipulado o quanto caminhar para frente e para trás, a partir da superfície, para que a "textura" seja calculada. Opcionalmente a caminhada pode ser interrompida caso um voxel abaixo do valor de *threshold* seja encontrado. Uma outra maneira de se obter a "textura" é caminhar o máximo possível na direção dada, enquanto o *threshold* permitir. Este último caracteriza a espessura. A figura 3.9 ilustra a rotina `vtextu` e seu *form* está na figura A.7.

3.2.2.6 vshad

A rotina `vshad` é a última no processo de visualização e faz a tonalização (*shading*) da imagem. *Shading* é um processo computacional que melhora o efeito 3D de uma imagem fornecendo uma ilusão de profundidade, geralmente alcançada por um modelo de iluminação. O tipo de tonalização desejado (*depth shading* ou *gradient shading*) pode ser escolhido como parâmetro (veja *form* na figura A.8). A tonalização mais simples, a *depth shading*, necessita apenas da informação de distância (veja equação 3.5), dada pelo arquivo de *z-buffer* gerado em `vzbuff`. Já na *gradient shading*, termo que implica no uso das normais

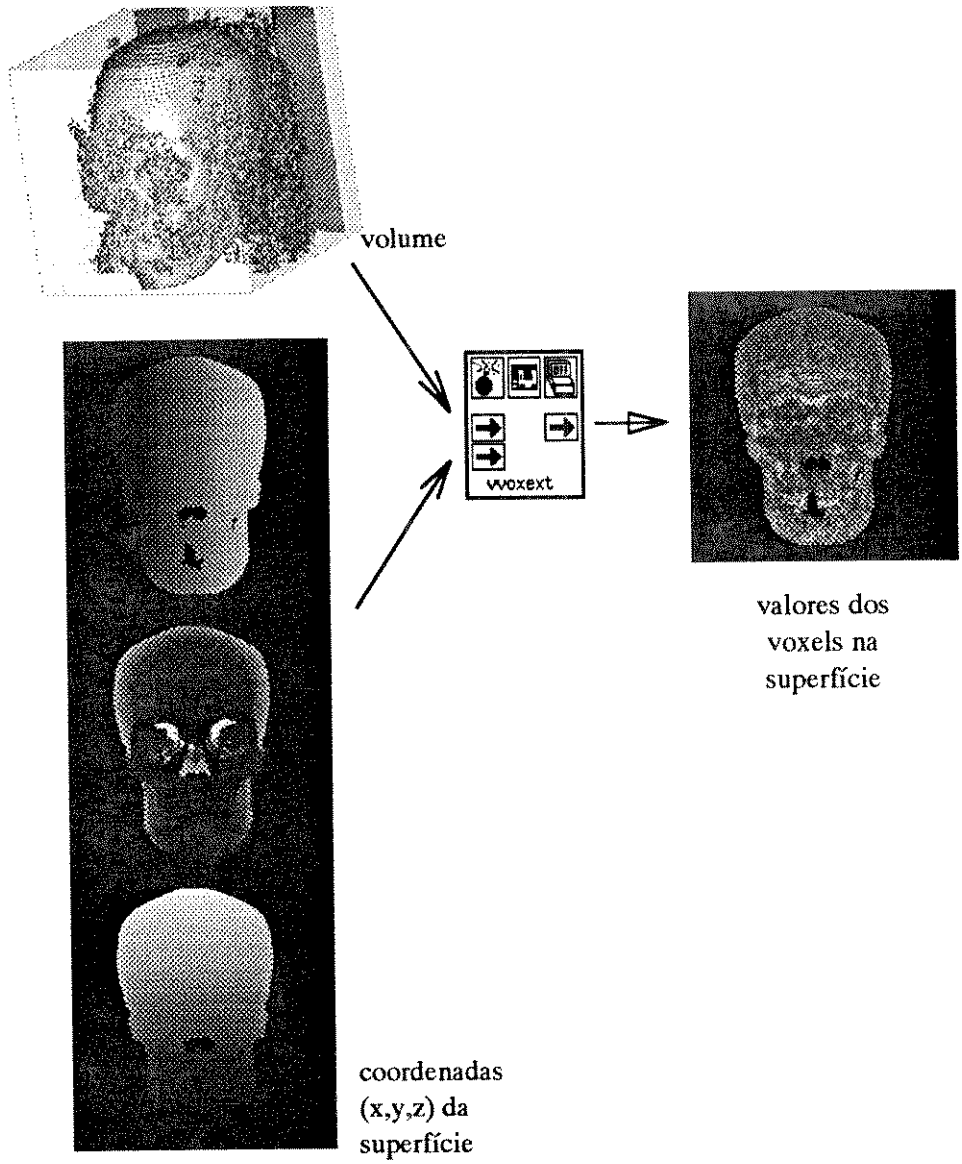


Figura 3.7: rotina vvoxext

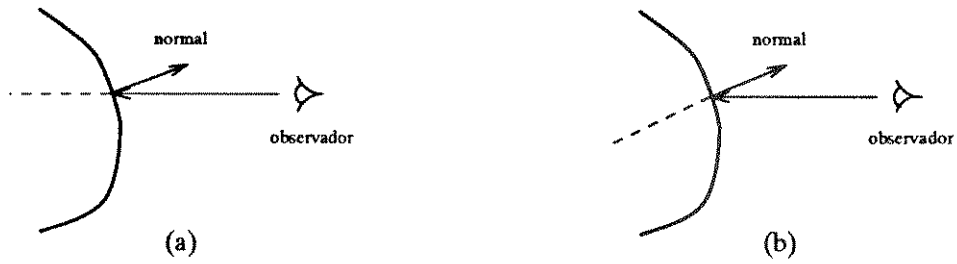


Figura 3.8: direções de obtenção da "textura": (a) direção do observador; (b) direção das normais

estimadas no espaço voxel, objeto ou imagem, além das distâncias, são também necessários os valores dos ângulos formados pelos raios de luz incidente e as normais à superfície (calculadas por `visnorm` ou `vvsnorm`). O arquivo das normais, juntamente com a informação dos ângulos de rotação utilizados na rotina `vzbuff`, são suficientes para poder calcular os ângulos necessários no modelo de Phong [25, pág. 311-317] para simular o comportamento físico da luz ao atingir a superfície do objeto, segundo a equação 3.6. A rotina também permite estipular todos os coeficientes envolvidos no modelo, bem como valores de intensidade máxima e mínima e valor de *background*.

Tonalização por distância:

$$I_{dist}(u,v) = \frac{I_{max} - I_{min}}{d_{min} - d_{max}}[d(u,v) - d_{max}] + I_{min} \tag{3.5}$$

Tonalização usando o modelo de Phong:

$$I(u,v) = I_{max}K_a + I_{dist}(u,v)[K_d \cos\theta(u,v) + K_s \cos^n 2\theta(u,v)] \tag{3.6}$$

onde:

I_{max} é a intensidade máxima

I_{min} é a intensidade mínima

d_{max} é a maior distância da superfície do objeto ao plano de visão (maior valor do *z-buffer*)

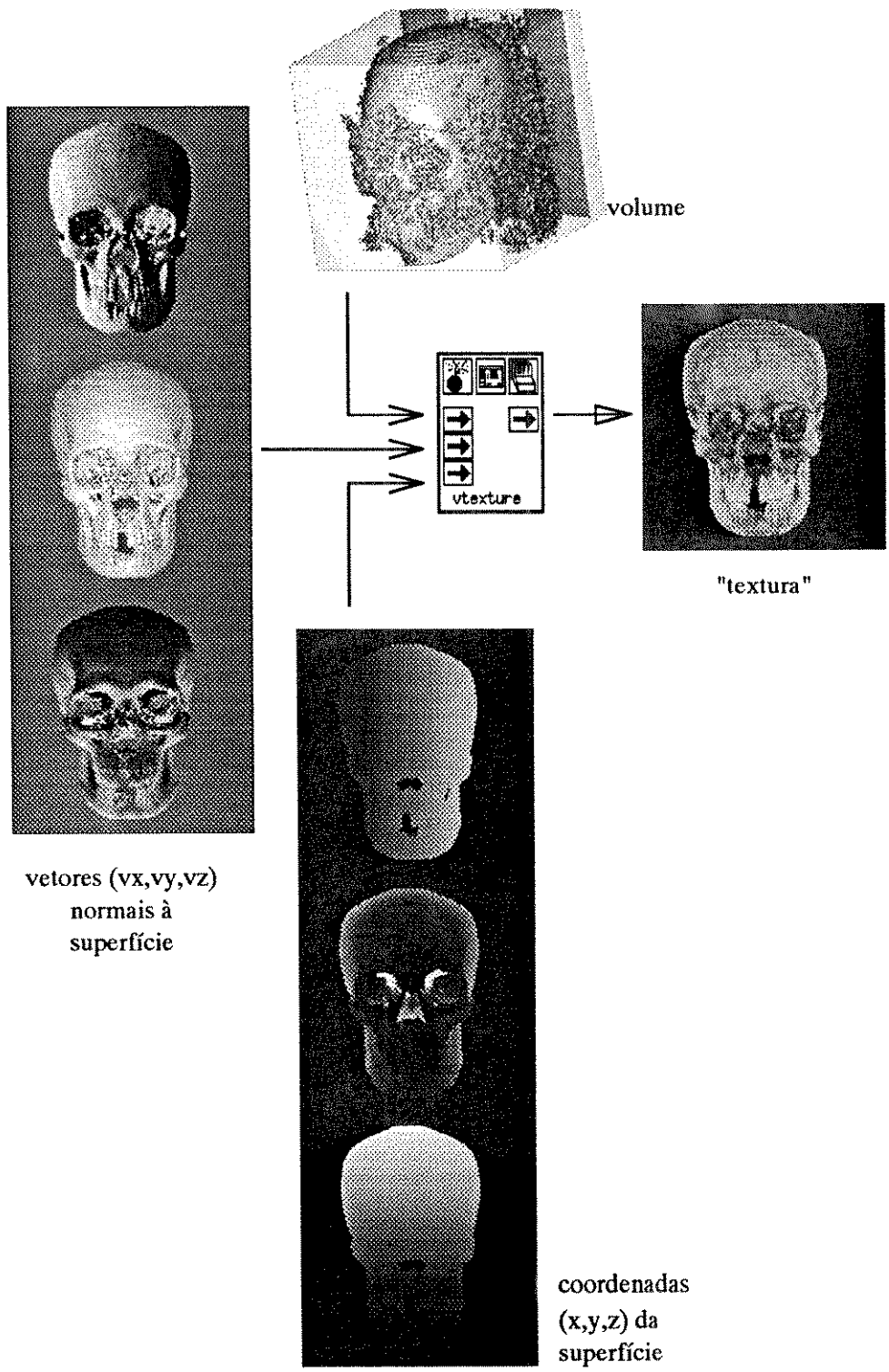


Figura 3.9: rotina vtextu

d_{min} é a menor distância da superfície do objeto ao plano de visão (menor valor do *z-buffer*)
$d(u,v)$ é a distância do *pixel* (u,v) à superfície do objeto (valor do *z-buffer* na posição (u,v))
$I(u,v)$ é o nível de cinza associado ao *pixel* (u,v)
K_a é o coeficiente de reflexão para a luz ambiente
K_d é o coeficiente de reflexão difusa
K_s é o coeficiente de reflexão especular
$\theta(u,v)$ é o ângulo entre o vetor normal à superfície do objeto e o raio de incidência lançado do *pixel* (u,v) do plano de projeção

Opcionalmente pode-se usar a informação de "textura" gerada pela rotina `vtextu`, de forma a ser mais uma parcela somada na equação do modelo de Phong, ponderada por um coeficiente de "textura" (K_t) fornecido (veja equação 3.7). A figura 3.10 mostra as entradas e saídas de `vshad`.

Tonalização usando o modelo de Phong adaptado para "textura":

$$I(u,v) = I_{max}K_a + I_{dist}(u,v)[K_d \cos\theta(u,v) + K_s \cos^n 2\theta(u,v) + K_t T(u,v)] \qquad (3.7)$$

onde:
K_t é o coeficiente de "textura"
$T(u,v)$ é a informação de "textura" associado ao *pixel* (u,v)

3.2.3 Rotinas de Processamento Puntual do Voxel

3.2.3.1 vthresh3d

A rotina `vthresh3d` gera uma imagem com apenas dois níveis de cinza que podem ser escolhidos pelo usuário, através de um valor de corte (*threshold*) aplicado em todo o volume (veja *form* na figura A.9). Como dois valores de *threshold* (um inferior e outro superior) podem ser estipulados, a rotina pode agir como uma função do tipo *windowing*,

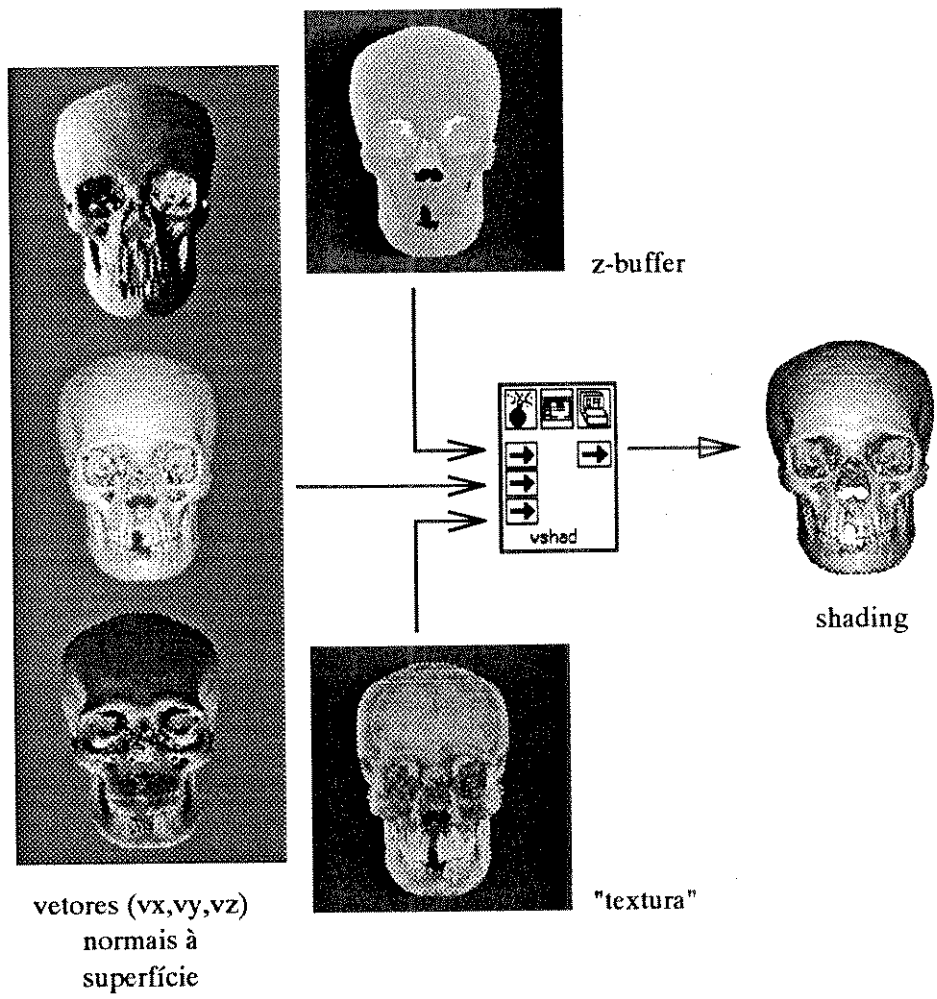


Figura 3.10: rotina vshad

onde *voxels* que possuam densidades que estejam entre os valores da janela são mapeados para um valor e os demais para outro. A figura 3.11 mostra o resultado da aplicação da rotina em um volume com 9 fatias. A técnica utilizada para visualizar as fatias na forma apresentada será vista no capítulo 4.

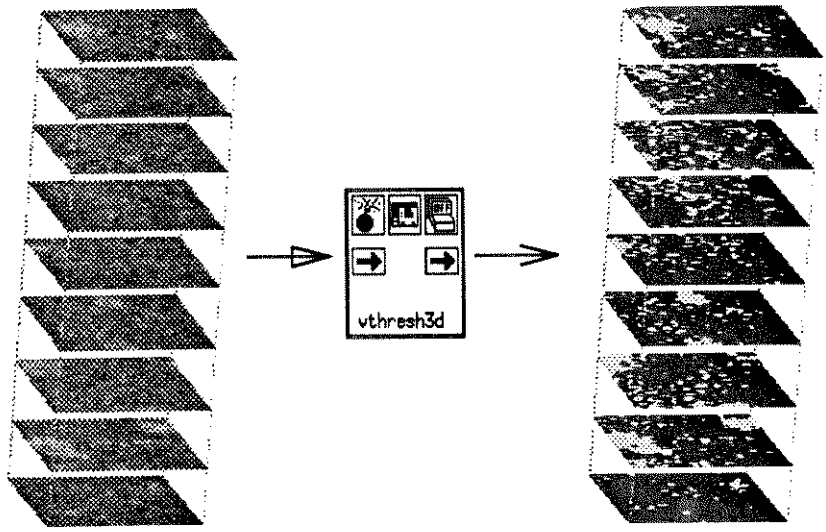


Figura 3.11: rotina vthresh3d

3.2.4 Rotinas de Processamento Espacial

3.2.4.1 vqmed3d

A rotina **vqmed3d**, cujo *form* está na figura A.10, realiza uma suavização em três dimensões do volume através de um processo de filtragem por mediana, onde um paralelepípedo (núcleo) de dimensões w x h x d definidas pelo usuário, varre todo o volume trocando o valor do *voxel* central pela mediana dos seus vizinhos que estejam internos ao paralelepípedo. O tratamento das bordas pode ser feito através de *overlapping* do núcleo ou não. Neste último caso o volume resultante terá as bordas com *voxels* zero. A mediana é o elemento central do vetor ordenado que contém os *voxels* interiores ao núcleo e a ordenação é feita através do algoritmo *quick sort*. A figura 3.12 mostra um volume corrompido por ruídos e o resultado após processado por **vqmed3d**, ilustrando uma de suas aplicações.

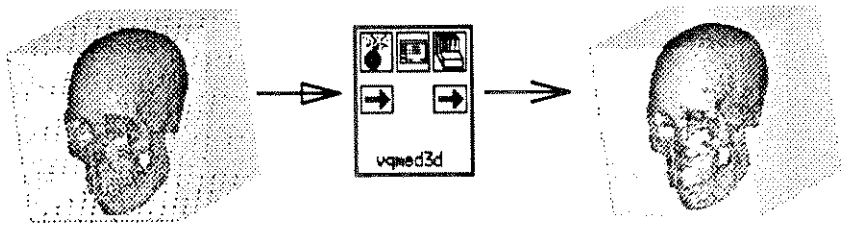


Figura 3.12: rotina vqmed3d

3.2.4.2 vconv3d

A rotina **vconv3d** é uma extensão de 2D para 3D da rotina **vconvolve** do Khoros. Ela realiza uma convolução em três dimensões de um volume, segundo o núcleo escolhido (veja *form* na figura A.11). Maiores detalhes sobre a teoria da convolução podem ser encontrados em Gonzalez et al. [8, pág. 81-92]. A figura 3.13 mostra as entradas e saídas de **vconv3d** numa aplicação particular para remover os ruídos de um volume através de um núcleo de média na vizinhança 3x3x3.

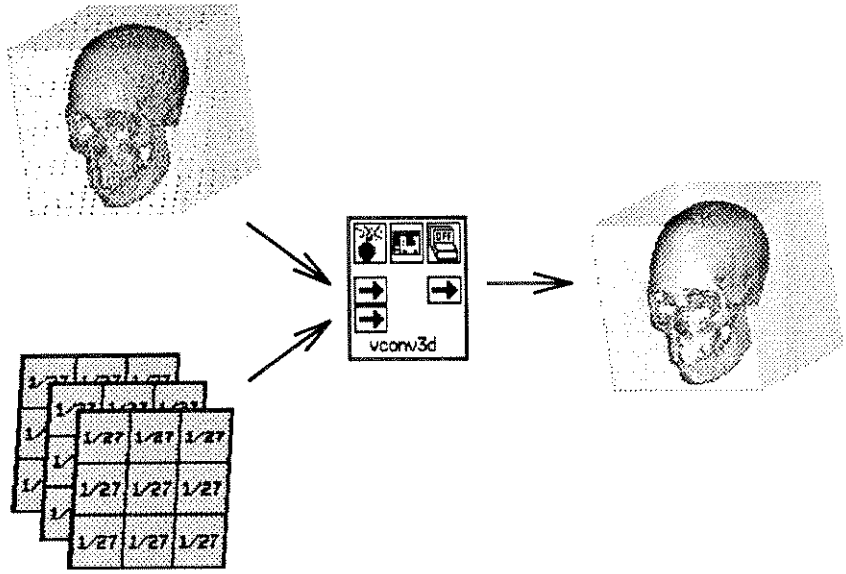


Figura 3.13: rotina vconv3d

3.2.5 Rotinas de Transformada Digital de Fourier

3.2.5.1 vfft3d

A rotina **vfft3d** computa a FFT (*Fast Fourier Transform*) direta ou inversa de um volume [8, pág. 61-81]. A entrada deve ser um volume cujas dimensões sejam potência de 2 e pode ser um único volume de dados complexos, um único volume contendo somente a parte real ou dois volumes, sendo um com a parte real e outro com a parte imaginária. Como saída existem três opções: volume complexo (parte real e imaginária), somente a parte real ou somente a parte imaginária, conforme mostrado no seu *form* (figura A.12). A figura 3.14 ilustra a atuação de **vfft3d**.

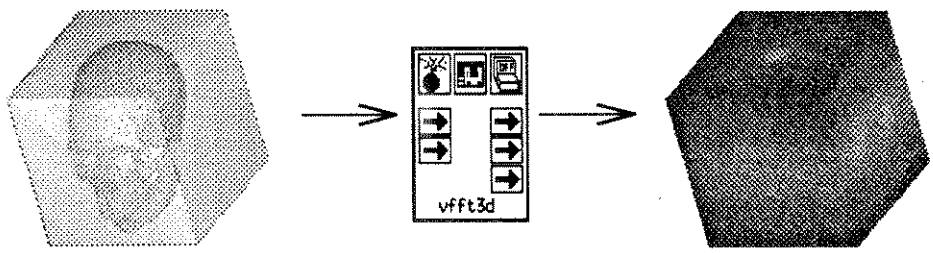


Figura 3.14: rotina vfft3d

3.2.5.2 vmpp3d

vmpp3d é capaz de computar várias representações de um volume de dados do tipo complexo, entre elas: magnitude, log(magnitude+1) , potência, log(potência+1) e fase, utilizados para analisar o espectro de Fourier (veja *form* na figura A.13). Muitos dos espectros decrescem rapidamente em função do aumento da freqüência, ocasionando, portanto, dificuldade na visualização das componentes de alta freqüência. A representação do espectro na forma log(magnitude+1) é freqüentemente usada para compensar estas situações [8, pág. 72-74]. Para facilitar a compreensão do processamento feito por **vmpp3d**, a figura 3.15 mostra o resultado obtido em uma fatia central do espectro de uma esfera de raio 8 inserida num volume de 64x64x64. **vmpp3d** também foi utilizada para gerar a imagem mostrada na saída da figura 3.14, para tornar visível o volume espectral calculado por **vfft3d**.

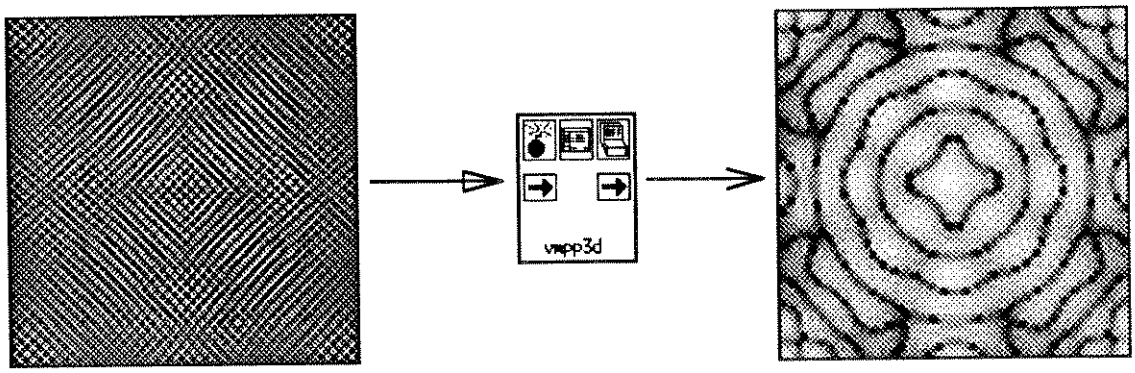


Figura 3.15: rotina vmpp3d

3.2.6 Rotinas de Manipulação de Regiões

3.2.6.1 vinsert3d

`vinsert3d` insere um sub-volume em outro, na posição especificada pelos *offsets* em x, y e z. Caso o sub-volume não se encaixe perfeitamente dentro do outro, ocorrerá o preenchimento (*padding*) de forma a englobar ambos, como se fosse um *bounding box*. O valor destes *voxels* de preenchimento (tanto a parte real quanto a imaginária, se os dados forem do tipo complexo) podem ser especificados pelo usuário (veja *form* na figura A.14). Devido ao preenchimento, não há restrições quanto ao tamanho dos volumes, nem à posição de inserção, a não ser que esta deve ser um ponto de coordenadas maiores ou iguais a zero. A figura 3.16 mostra o resultado da inserção do cubo no paralelepípedo. A região mais clara no volume da saída corresponde à área preenchida automaticamente.

3.2.6.2 vpad3d

`vpad3d` (figura 3.17) é uma rotina semelhante a `vinsert3d`. Na `vpad3d`, apenas o sub-volume é fornecido e o outro é criado internamente de acordo com as dimensões e valores de *voxel* constantes fornecidos (veja *form* na figura A.15).

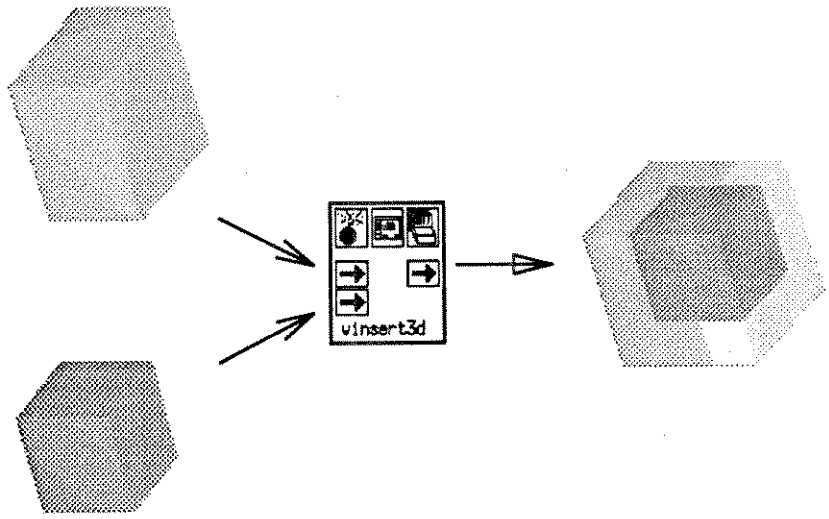


Figura 3.16: rotina vinsert3d

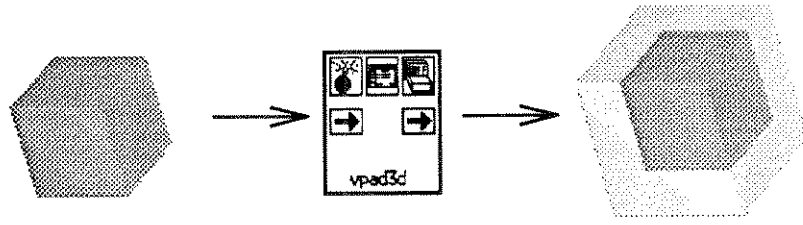


Figura 3.17: rotina vpad3d

3.2.6.3 vaband

A rotina **vaband** (*add bands*), cujo *form* se encontra na figura A.16, realiza uma adição de fatias a um volume, permitindo que se escolha uma faixa de fatias do volume de entrada a ser copiada para uma determinada posição de um segundo volume. Este segundo volume é opcional e, quando não fornecido, a faixa de fatias selecionada é copiada diretamente para a posição escolhida no arquivo de saída. Neste último caso o arquivo de saída funciona também como arquivo de entrada, o que não é uma situação muito comum envolvendo as rotinas do Khoros. A implementação foi feita desta forma para possibilitar que bandas geradas por um *loop* numa *workspace* possam ser concatenadas a um arquivo inicialmente vazio, de modo transparente ao usuário. Neste caso deve-se ter atenção especial, pois a reexecução da *workspace* não elimina as bandas já concatenadas em execuções anteriores. Um exemplo prático pode ser visto na figura 4.5 no capítulo seguinte. A figura 3.18 exemplifica a atuação de **vaband**.

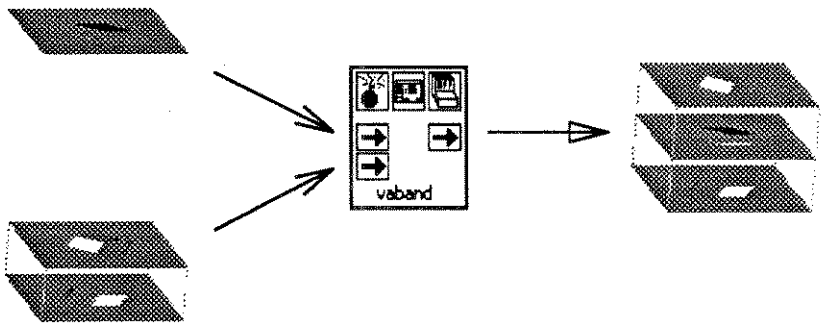


Figura 3.18: rotina vaband

3.2.6.4 vdband

vdband (*delete bands*) é usada para remover uma faixa de fatias de um arquivo. O arquivo de saída é uma cópia do arquivo de entrada, porém sem as fatias selecionadas (veja figura 3.19 e seu *form* na figura A.17).

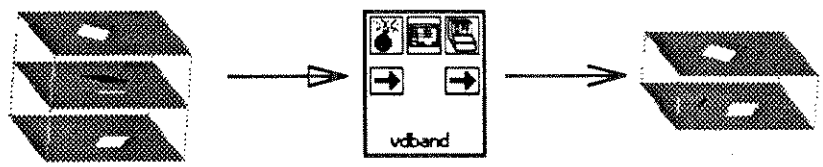


Figura 3.19: rotina vdband

3.2.6.5 vmband

`vmband` (*move bands*) permite que uma faixa de fatias de uma imagem seja movida para uma outra posição dentro da mesma imagem. (veja figura 3.20). Não há aumento do número de fatias, apenas uma mudança de ordem. A posição destino geralmente determina a posição da primeira fatia da faixa selecionada, porém em uma situação particular esta posição pode corresponder à posição da última fatia da faixa. A condição em que isto ocorre é quando a posição destino somada ao número de fatias contidas na faixa a ser movida ultrapassar o número de fatias da imagem. Seu *form* está na figura A.18.

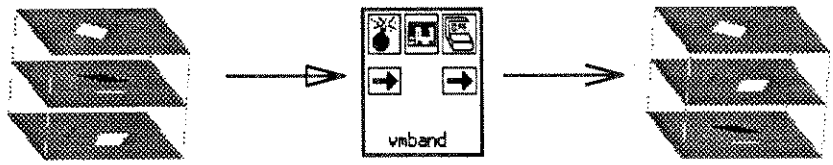


Figura 3.20: rotina vmband

3.2.6.6 vmontage

A rotina `vmontage` cria uma imagem 2D a partir de uma imagem 3D, colocando as fatias do volume lado-a-lado, dispostas em linhas e colunas, permitindo a visualização de todas as fatias de uma só vez. A figura 3.21 ilustra algumas das possibilidades de `vmontage`, cujo *form* está na figura A.19.

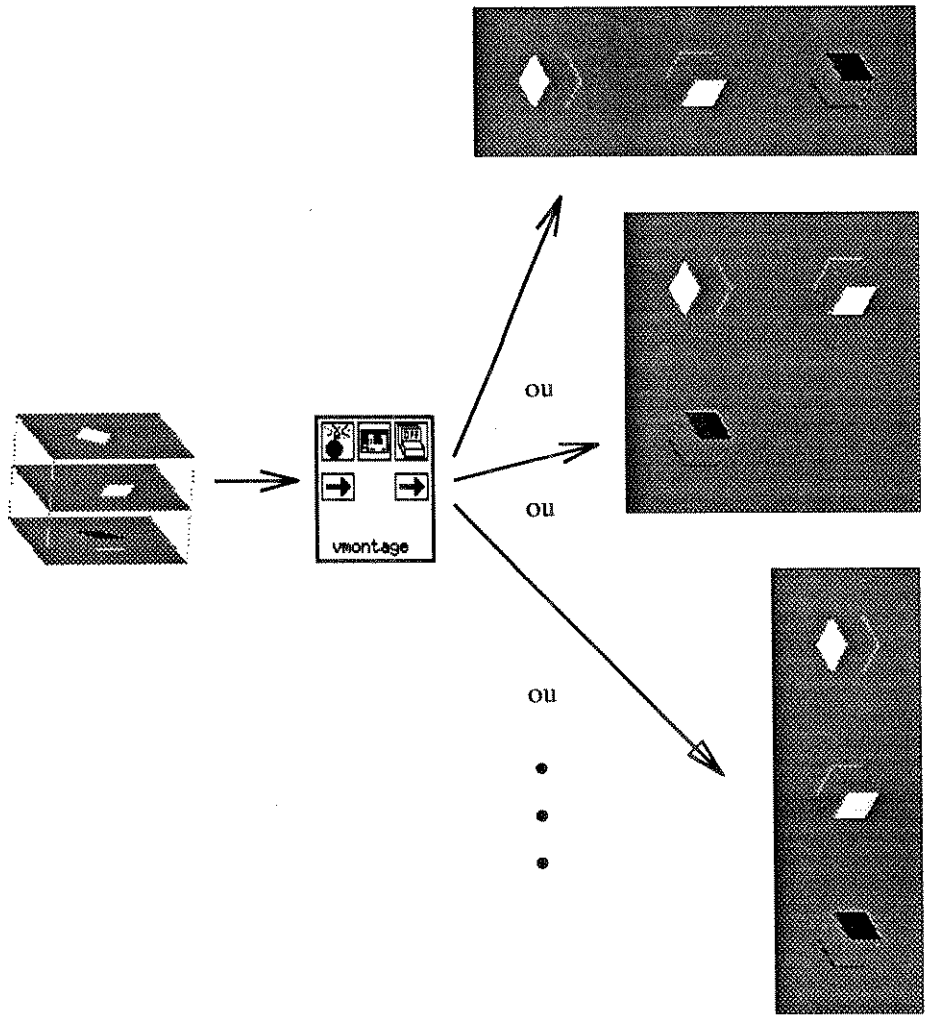


Figura 3.21: rotina vmontage

3.2.6.7 vplanexyz

vplanexyz (*form* na figura A.20) gera, a partir de um volume, até três imagens que contêm informações sobre o valor dos *voxels* nos planos de coordenadas x, y e z constantes e que são fornecidos opcionalmente, se o plano correspondente for desejado na saída. Veja a figura 3.22 que ilustra a rotina **vplanexyz** quando aplicada ao volume do crânio de um paciente.

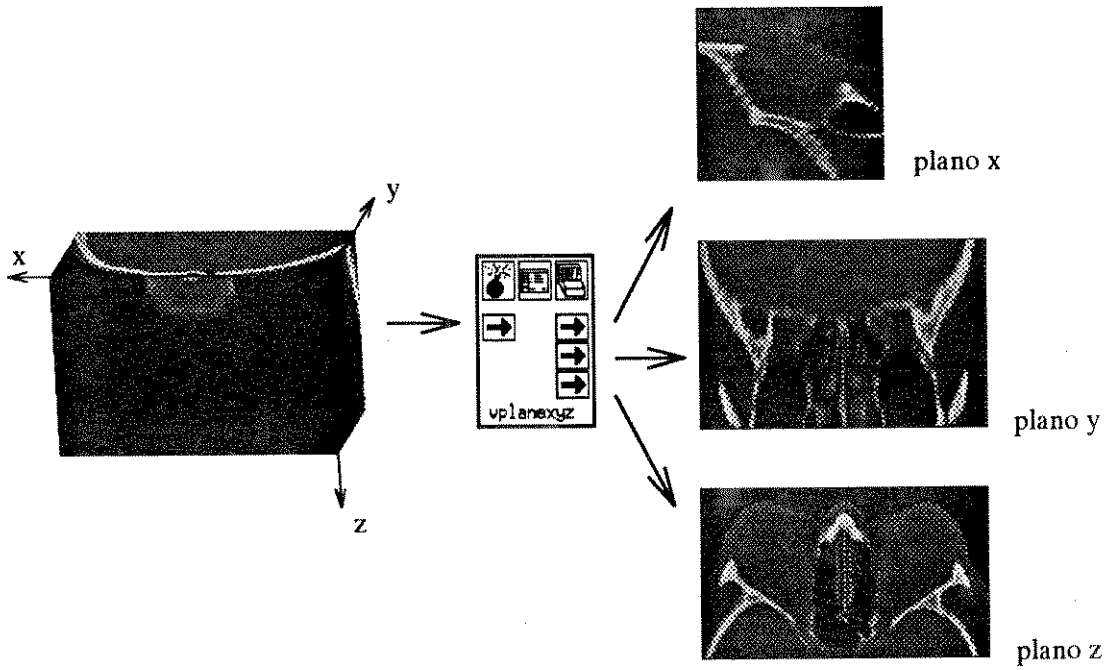


Figura 3.22: rotina vplanexyz

3.2.7 Rotinas de geração de Volumes Sintéticos

3.2.7.1 vframe

vframe gera um volume de dimensões escolhidas pelo usuário, onde todo o volume é preenchido pelo valor de *background*, com exceção das arestas que recebem o valor de *foreground*. A espessura das arestas é variável, podendo ser estipulada por parâmetro, conforme mostrado no seu *form* na figura A.21. A figura 3.23 exemplifica o uso de **vframe**.

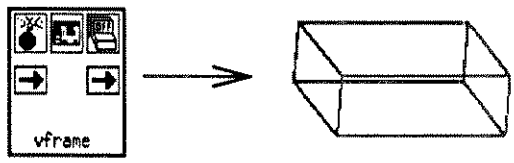


Figura 3.23: rotina vframe

3.2.7.2 vgauss3d

`vgauss3d` gera um volume de dimensões escolhidas pelo usuário, contendo uma ou mais distribuições gaussianas. Segundo pode ser visto no seu *form* (figura A.22), a média e a variância (nas três direções independentemente) podem ser especificadas, bem como a amplitude (normalizada ou não) e a correlação (expressa através dos ângulos de rotação em torno dos eixos x e z). No caso de mais de uma distribuição gaussiana por volume, os parâmetros que definem cada uma delas são fornecidos seqüencialmente em um arquivo com 9 bandas, ou seja, uma para cada parâmetro. A figura 3.24 mostra a rotina `vgauss3d` e o volume contendo uma única distribuição criado por ela.

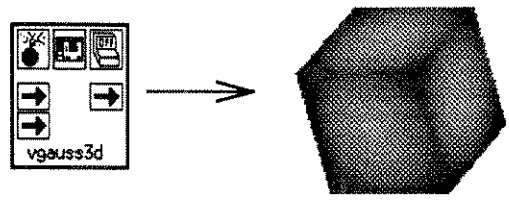


Figura 3.24: rotina vgauss3d

3.2.7.3 vimpul3d

`vimpul3d` gera um volume de dimensões escolhidas pelo usuário, contendo pulsos unitários (pontos de densidade 1), distribuídos segundo *offset* inicial e espaçamento entre eles nas três direções. A figura 3.25 mostra um exemplo de volume contendo pulsos gerados por `vimpul3d`. Seu *form* está na figura A.23.

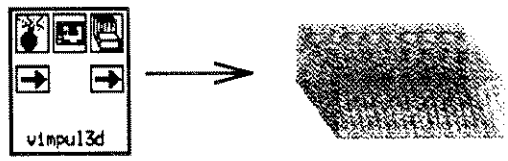


Figura 3.25: rotina vimpul3d

3.2.7.4 vorthog

`vorthog` (*form* na figura A.24) gera um volume de dimensões escolhidas pelo usuário, contendo planos ortogonais entre si. Cada um dos três planos é definido por apresentar pontos de coordenada constante em uma das direções, tendo portanto equações do tipo $x = k1$, $y = k2$ e $z = k3$. Valores de *voxel* para caracterizar o fundo, os planos, suas arestas e suas interseções são parâmetros da rotina e, sendo segmentadas por *threshold* num processamento posterior, podem ser facilmente identificadas. A figura 3.26 mostra um volume criado por `vorthog` contendo 3 planos ortogonais passando pelo centro do volume.

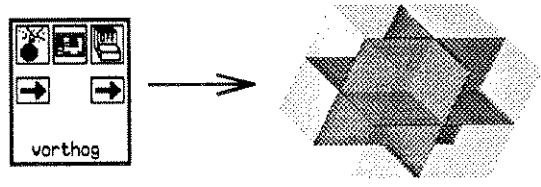


Figura 3.26: rotina vorthog

3.3 Diferenças Entre a Primeira Versão e a Versão Atual

As quatro rotinas que fizeram parte da primeira versão da *toolbox* V3DTOOLS (`soft2viff`, `vinter`, `zbuffer` e `shading`) são implementações de algumas técnicas de visualização apresentadas no capítulo 3 da tese de Mestrado citada anteriormente [4]. Estas rotinas possibilitam a formação de um pequeno sistema de visualização volumétrica, indo do pré-processamento até o *rendering* final do volume. De uma forma geral, todas elas sofreram melhorias em aspectos implementacionais, bem como oferecendo um maior número de opções, como é o caso de `vzbuff` e `vinter`. Como um primeiro passo no desenvolvimento da atual *toolbox* V3DTOOLS, as rotinas já existentes foram revisadas e modificadas para

atender ao requisito principal da *toolbox*, ou seja, ser composta por rotinas com funções operacionais básicas.

Nesta reestruturação, a rotina `soft2viff`, responsável pela conversão dos dados do formato SOFTVU para o formato VIFF, foi simplesmente transferida para outra *toolbox*, por não se tratar essencialmente de uma operação básica para dados 3D. Esta conversão se fazia necessária porque anteriormente à aquisição do Khoros só dispúnhamos do sistema SOFTVU de visualização e os dados 3D disponíveis estavam formatados segundo este padrão. Além disso, existia um contrato com a GE, onde era possível converter os dados de fita do tomógrafo GE9800 do Depto. de Radiologia do Hospital das Clínicas da Unicamp para o formato SOFTVU.

A antiga rotina `vinter` não possui opção para interpolação de ordem zero, incorporada à versão atual de `vinter`.

A rotina `zbuffer` passa a se chamar `vzbuff` na nova versão, atendendo a algumas regras de nomenclatura do sistema Khoros. A antiga `zbuffer` gera na saída um arquivo composto por uma, ou opcionalmente duas bandas, onde a primeira delas contém o *z-buffer*, e a segunda contém o ângulo entre o vetor normal à superfície e a direção da luz incidente, que são informações necessárias para o próximo passo na visualização, o *shading*. A existência ou não da banda adicional contendo os ângulos na saída de `zbuffer`, depende exclusivamente do tipo de *shading* desejado, o que é decidido por parâmetros já em `zbuffer`, através da escolha entre *gradient shading* (com ângulos) ou *depth shading* (sem ângulos). Uma vez escolhido *gradient shading*, as normais são calculadas internamente em `zbuffer`, segundo um parâmetro optando entre estimativa das normais no espaço imagem, no espaço objeto ou no espaço *voxel*. Por se tratar de um processamento interno, as informações a respeito das normais são dados intermediários e não são fornecidos na saída da rotina. A nova `vzbuff` não faz o cálculo das normais, deixando a tarefa para as rotinas `visnorm` e `vvsnorm`. Além do arquivo de *z-buffer*, ela também gera na saída os arquivos de coordenadas e densidade dos *voxels* da superfície. Os ângulos α e β que determinam a posição do plano de projeção ao redor do objeto não são limitados a uma faixa de valores que vai de -180 a +180 graus, como na antiga `zbuffer`. Isto facilita o uso da rotina em laços de programas que determinam os ângulos por variáveis a cada iteração. A opção de *preview* é característica somente da nova `vzbuff`.

Assim como a **vzbuff**, a antiga **shading** passa a ser chamada de **vshad** na nova versão. A nova **vshad**, adaptada para atender às mudanças em **zbuffer**, ao invés de ter como única entrada o arquivo vindo da saída de **zbuffer**, passa a ter três entradas. A primeira delas contém o arquivo de *z-buffer* e a segunda recebe o arquivo com as normais. Observe que os ângulos formados pelo raio de luz incidente e as normais à superfície, necessários na equação 3.6 não são mais fornecidos, tendo, portanto, que serem calculados internamente em **vshad**. Este cálculo só é possível se, além das normais, for conhecido também a direção do raio incidente, o que é dado pelos ângulos α e β utilizados para rotacionar o plano de projeção em **vzbuff**, que passam a ser também parâmetros da nova **vshad**. O arquivo contendo o *z-buffer* é uma entrada obrigatória e o que contém as normais é opcional, pois no *depth shading*, a informação do ângulo entre as normais e o raio incidente não é necessário. A terceira entrada, também opcional, se relaciona com a utilização ou não da informação de "textura", gerada pela rotina **vtextu**, antes não implementado em **shading**. A informação extra de "textura" é processada incorporando-a ao modelo de iluminação de Phong, como mostrado na equação 3.7.

3.4 Conclusão

A extensão e melhoria da versão original da *toolbox* V3DTOOLS resultou em um conjunto de operadores desempenhando funções consideradas básicas para serem usados num processamento de nível mais alto, através da combinação dos resultados obtidos individualmente com cada rotina. Esta combinação é facilmente implementada no ambiente de programação visual do Khoros, o **Cantata**, ou utilizando-se a linguagem *shell* do Unix. No próximo capítulo são abordados exemplos práticos de combinação dos diversos operadores da *toolbox* V3DTOOLS na solução de problemas de visualização, manipulação e análise de dados tridimensionais.

Capítulo 4

Experimentos e Resultados

Existem três maneiras de se combinar os operadores básicos que compõem a *toolbox* V3DTOOLS, descrita no capítulo 3. A forma mais imediata é através da criação de *workspaces*. Numa abordagem não tão direta, a combinação pode ser feita também através da escrita de programas que desempenhem funções mais complexas, fazendo para isto, chamadas às bibliotecas das rotinas básicas. Finalmente, o agrupamento de rotinas pode ser feito, de modo similar ao anterior mas sem a necessidade da criação de código executável, por *shell scripts*, que fazem chamadas aos próprios executáveis das rotinas. É claro que esta solução, pelo fato da comunicação entre as rotinas ser feita através da escrita e leitura dos dados em disco, é mais lenta que se fosse feita diretamente na memória, porém mesmo assim é mais rápida e compacta que uma *workspace*, sem contar o fato que os arquivos temporários que não forem sendo mais usados podem ser apagados com o decorrer do processamento, otimizando a utilização do espaço em disco. Desta forma, são apresentadas neste capítulo algumas *workspaces* e *shell scripts*, bem como seus resultados, mostrando a flexibilidade conseguida com a filosofia de adotar operadores básicos na confecção de processamentos mais complexos. Assim como as rotinas que representam os operadores básicos, as *shell scripts* também possuem seus próprios *forms* para entrada de dados e parâmetros, e estão no apêndice A no final do trabalho.

4.1 Rendering Simples

No capítulo 3, muitas das ilustrações foram obtidas pela utilização de diversas rotinas combinadas entre si, buscando novas maneiras de visualização. Praticamente todas elas se envolveram com algum tipo de *rendering*, e para melhor entendê-las, é apresentada na figura 4.1 uma *workspace* simples, com os elementos básicos para compor a visualização de um volume, desde os dados originais, até o *display* final.

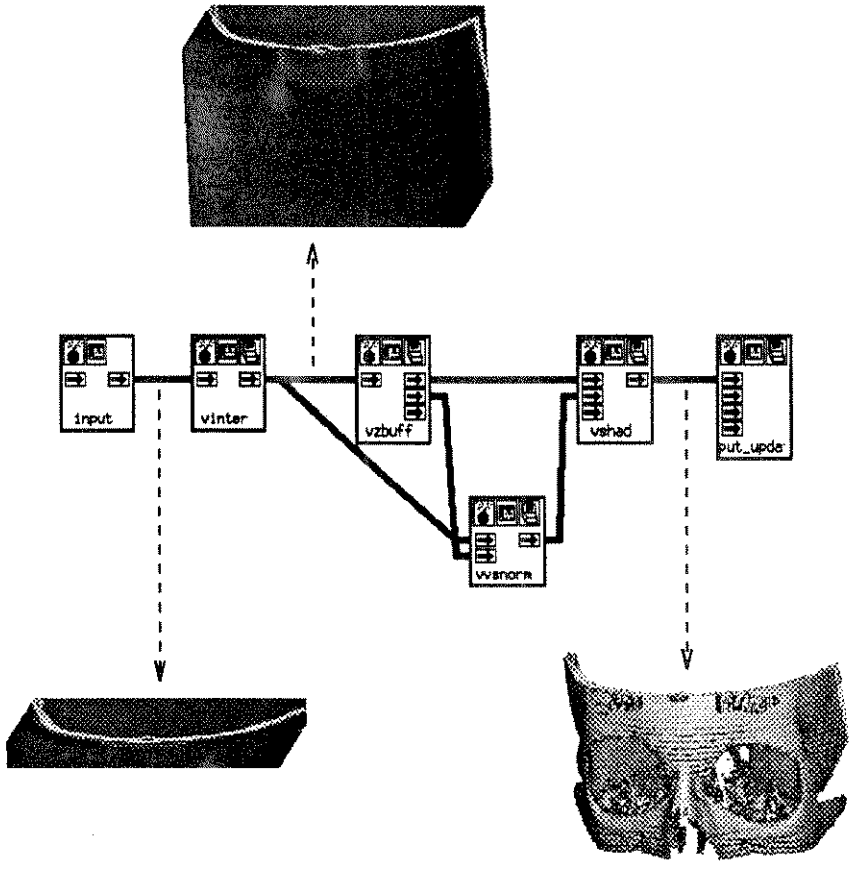


Figura 4.1: rendering simples

Com um valor de *threshold* apropriado na rotina `vzbuff`, é possível detetar a superfície da porção óssea do volume de dados. Nos *renderings* da figura 3.2 foram utilizados valores de *threshold* iguais a 1 para que todo o conteúdo dos volumes pudessem ser vistos. Voltando à figura 3.2, observe a diferença entre os dois volumes apresentados. Enquanto

o volume contendo a região orbital de um paciente em 3.2b está totalmente preenchido, o crânio seco em 3.2a está em um volume com bastante espaço vazio, ou seja, densidade de *voxel* igual a zero. A figura 4.2 mostra o resultado de uma animação obtida com a variação do valor de *threshold* no volume de dados do paciente. Esta variação permite a "remoção" gradual dos tecidos de densidade mais baixa até atingir a parte óssea.

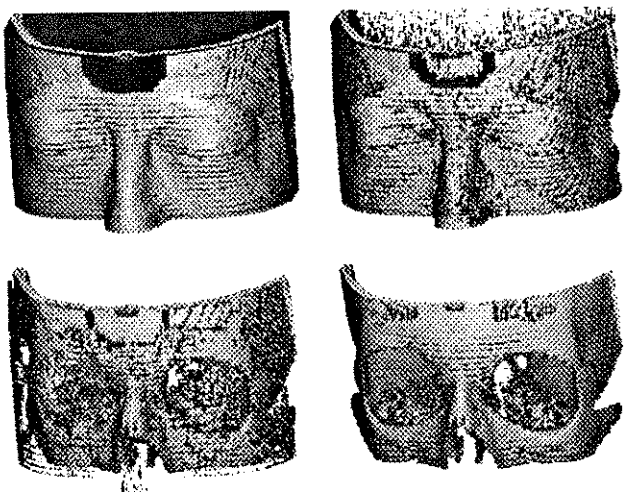
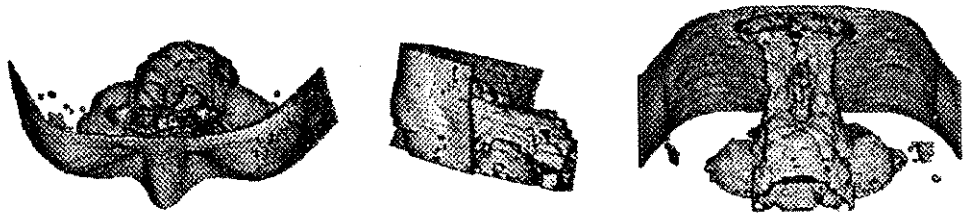


Figura 4.2: animação variando threshold

A segmentação por *threshold* pode ser feita por `vthresh3d` imediatamente antes da aplicação da rotina `vzbuff`. A vantagem existe em alguns casos, onde se deseja, por exemplo, visualizar somente certos tecidos que ocupam uma faixa de densidade capaz de ser segmentada por *threshold*. A extração de um objeto de um volume de dados médicos usando a técnica de *threshold* é uma tarefa nem sempre possível, devido ao fato de que mais de um material pode estar presente dentro do volume de cada *voxel*, causando uma falha na classificação. Observe que em `vzbuff` não é possível definir a janela de *threshold* mas somente um valor de corte, que corresponde ao limite inferior da janela. Embora o volume tenha sido segmentado, ainda existe a possibilidade de estimativa das normais no espaço *voxel*, bastando que o volume de entrada para `vvsnorm` seja o volume em níveis de cinza (obtido pela filtragem do volume segmentado, como será visto na seção 4.9.4 ainda neste capítulo) e não o binário (depois da segmentação, sem nenhum processamento). Na figura 4.3 é mostrado o mesmo volume da figura 4.2 sob vários ângulos, onde somente a pele e um tecido mais central não foram removidos pela segmentação.



(a) (b) (c)

Figura 4.3: segmentação por faixa de threshold sob várias vistas: (a) superior; (b) lateral; (c) traseira

Outro parâmetro bastante manipulado em `vzbuff` é o ângulo de rotação do objeto em torno dos eixos x e z. A figura 4.4 mostra uma animação deste tipo, ou seja, o mesmo volume contendo o crânio seco visto por diversos ângulos, com variações de 30 em 30 graus.

Como uma primeira aplicação prática envolvendo uma *workspace*, pode-se utilizar a opção de *preview* da rotina `vzbuff` para se ter uma idéia do conteúdo desconhecido de um determinado volume. O processamento em modo *preview* não é rico em detalhes, mas é bastante rápido, pois opera somente em uma fração dos dados, de acordo com a taxa de sub-amostragem desejada. Se a relação entre tempo de processamento e taxa de sub-amostragem gerar uma imagem final muito pequena, o Khoros oferece rotinas de *zoom* para ampliação, ou pode-se usar a rotina `vexpand3d`, que embora tenha sido desenvolvida para atuar em dados tridimensionais, funciona perfeitamente nos casos bidimensionais, como as demais rotinas da *toolbox* V3DTOOLS. Por se tratar de um *preview*, o fato dos *pixels* serem replicados ao invés de interpolados não é tão relevante. A figura 4.5 mostra tanto a *workspace* quanto o resultado gerado por ela, tendo o volume do crânio seco como entrada. A *workspace* possui dois *loops*, o superior responsável pela rotação em torno do eixo z e o inferior responsável pela rotação em torno do eixo x. As rotinas `vaband` no final de cada *loop* servem para concatenar as imagens intermediárias geradas pelos *loops*, e a `vaband` central une o resultado final de cada um deles, que posteriormente é processado por `vmontage` e feito o *display* do volume nas seis vistas principais: frontal, lateral esquerda, traseira, lateral direita, superior e inferior. Observe que o volume de entrada utilizado foi o interpolado, e não o original, como será feito de agora em diante, salvo indicação contrária.

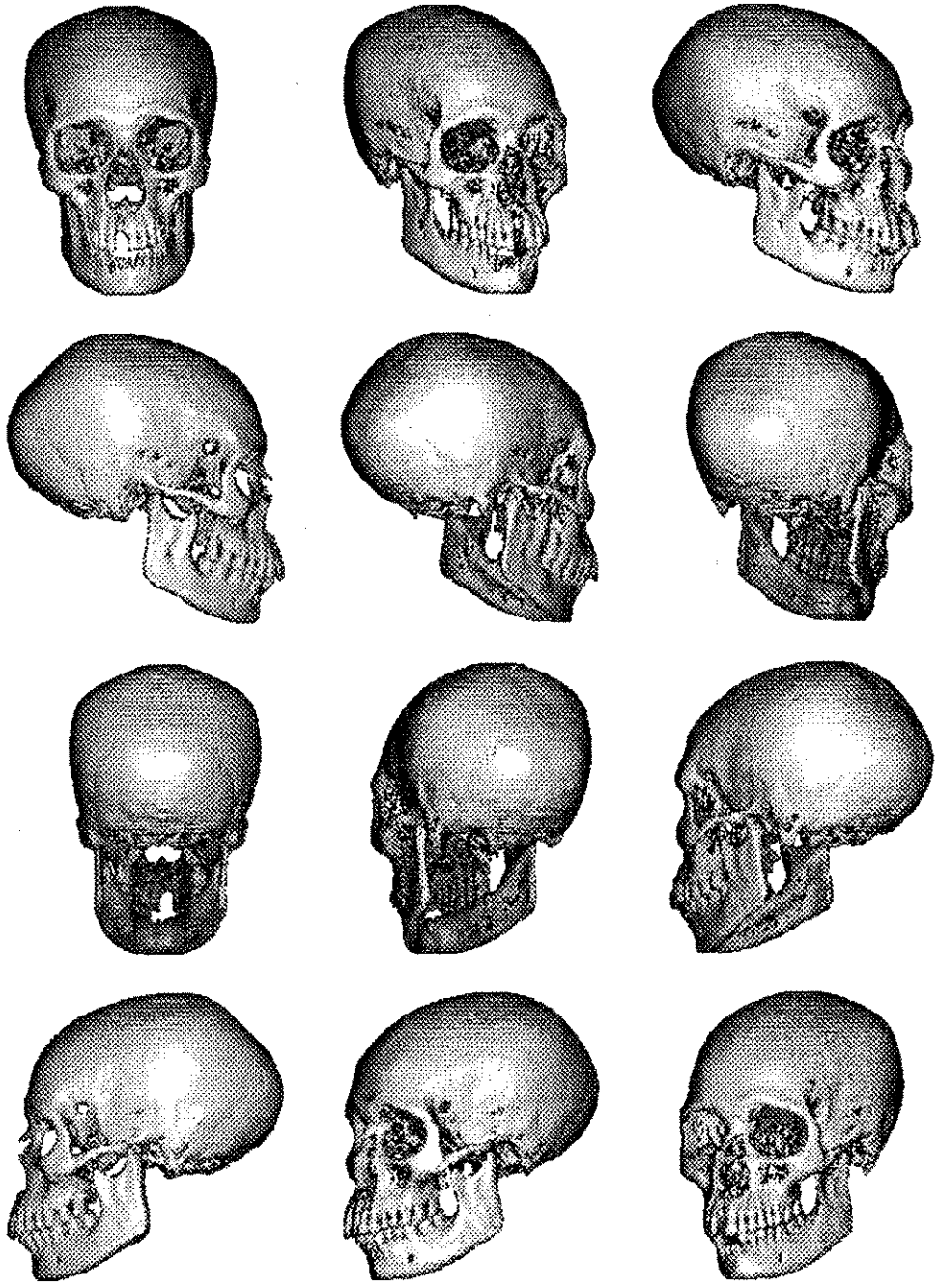
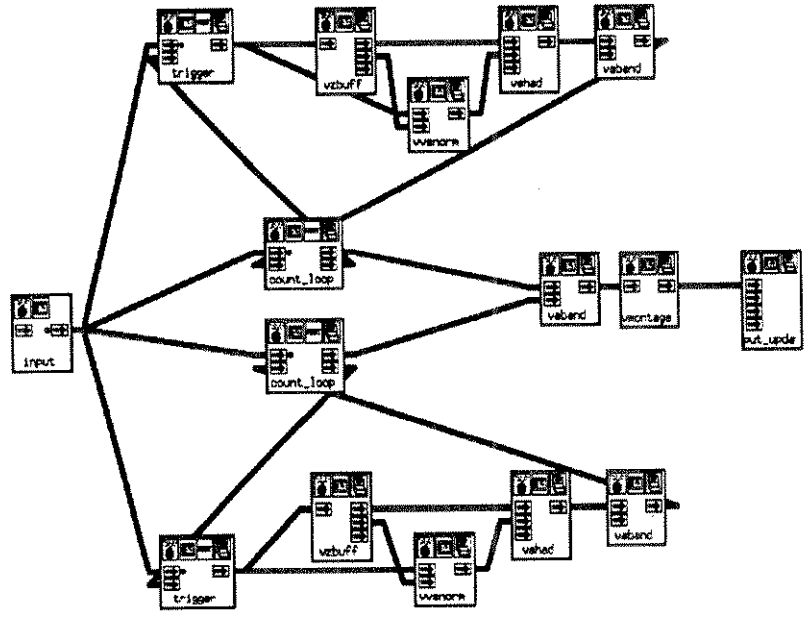
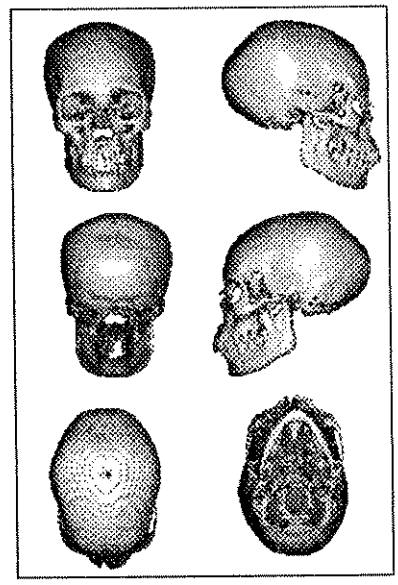


Figura 4.4: animação variando ângulos de rotação

UNICAMP
BIBLIOTECA CENTRAL



(a)



(b)

Figura 4.5: combinação de rotinas para aplicação em preview: (a) workspace; (b) resultado

O operador **vzbuff** tem ainda um outro parâmetro que estabelece a distância do plano de projeção em relação ao objeto, possibilitando até mesmo que o plano esteja dentro do objeto, o que pode resultar em efeitos de cortes. A figura 4.6 ilustra o uso deste parâmetro.

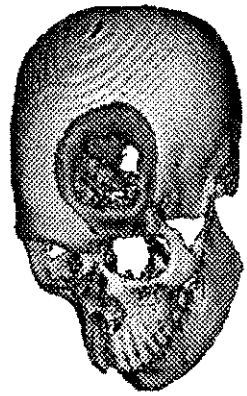


Figura 4.6: efeito da distância do plano de projeção em vzbuff

Geralmente os valores dos ângulos α e β utilizados pela rotina **vshad** são os mesmos usados em **vzbuff**, porém, para volumes contendo um único objeto, cuja superfície visível seja convexa, pode-se usar outros valores de ângulos, simulando a mudança de posição da fonte de luz. A figura 4.7 é um exemplo do efeito causado pela mudança destes parâmetros numa esfera. As restrições impostas ao objeto deve-se ao fato de que a rotina **vshad** não realiza um algoritmo de *ray casting* como é feito em **vzbuff**, e portanto efeitos de sombra, decorrentes da interferência entre objetos que ocupam um mesmo volume ou da interferência do objeto com ele próprio, quando se trata de objetos côncavos, não são tratados de forma correta.

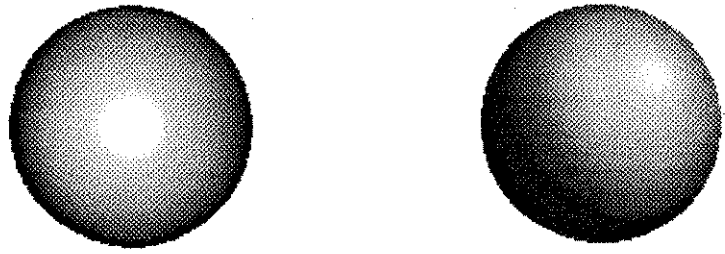
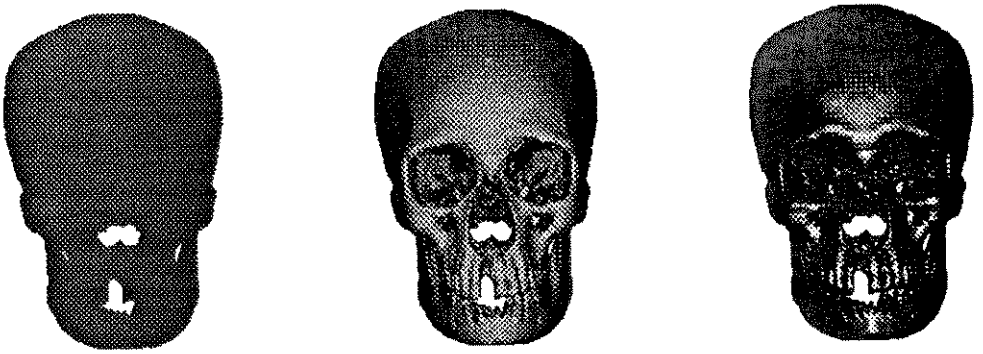


Figura 4.7: efeito dos ângulos de rotação em vshad

Ainda em **vshad**, os parâmetros que controlam os efeitos da iluminação sobre o objeto visualizado ficam praticamente inalterados uma vez que um resultado considerado satisfatório seja conseguido. Normalmente tem-se observado que contribuições de 20% para a luz ambiente, 80% para a componente difusa e 20% para a especular, representados pelos coeficientes K_a, K_d e K_s na equação 3.6, oferecem resultados bastante aceitáveis, quando não é usada a textura. A figura 4.8 contém, respectivamente, as contribuições isoladas de cada uma das parcelas citadas.



(a) (b) (c)

Figura 4.8: contribuições da luz (a) ambiente, (b) difusa e (c) especular no shading

Os vetores normais utilizados na figura 4.1 foram calculados pela rotina `vvsnorm`. Como o volume de entrada era em níveis de cinza, as normais se caracterizaram por ser do tipo *voxel space normal vectors*. Se a entrada fosse binária, então teríamos os *object space normal vectors*. Existem ainda as normais *image space normal vectors*, estimadas por `visnorm` com base na informação de distância contida no *z-buffer*. Como dito anteriormente, esta última estimativa pode oferecer resultados indesejáveis nos pontos de descontinuidade espacial, como bordas, por exemplo, que se apresentam em tons mais escuros. As tonalizações que usam as normais são referenciadas pelo termo genérico *gradient shading*. Existe ainda o *depth shading*, que utiliza somente a informação de distância contida no *z-buffer*. Os *renderings* conseguidos com cada uma das técnicas podem ser vistos na figura 4.9 e a *workspace* correspondente na figura 4.10.

O uso de uma ou outra técnica de *shading* proporciona diferentes resultados de acordo com o volume a ser visualizado, mais precisamente de acordo com o método de aquisição de dados utilizado. Além disso, dependendo do pré-processamento realizado,

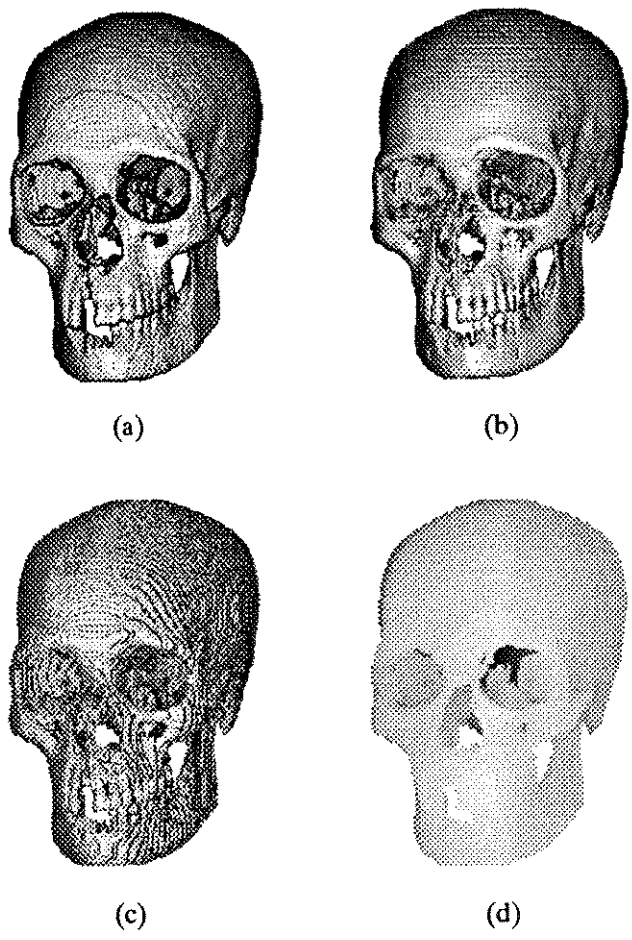


Figura 4.9: renderings obtidos segundo o shading usado: (a) image space; (b) voxel space; (c) object space; (d) depth shading

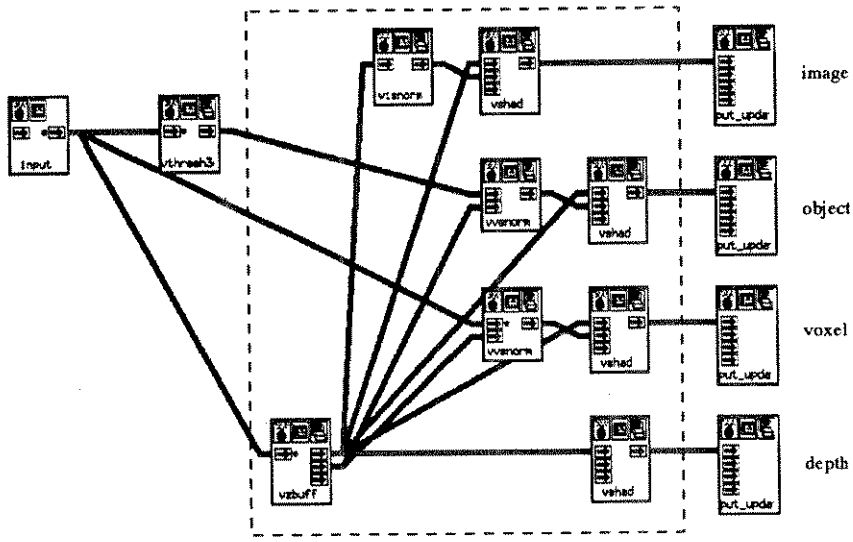
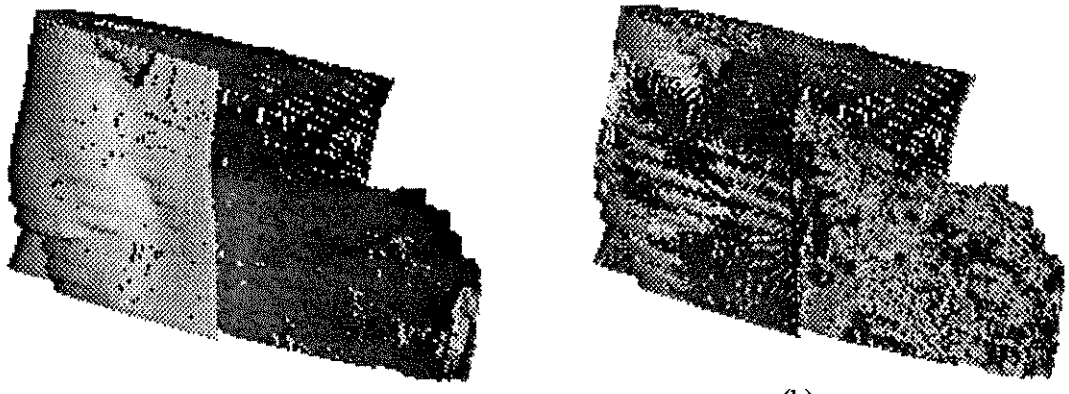


Figura 4.10: workspace implementando renderings segundo os 4 tipos de shadings

como *threshold* por exemplo, as técnicas mais adequadas para um mesmo volume de dados pode mudar de acordo com a situação. Na figura 4.2, a visualização do volume contendo o crânio de um paciente foi feita usando *shading* no espaço *voxel*. Na figura 4.3, o mesmo volume foi visualizado com *shading* no espaço imagem, por ser mais adequado neste caso. Como ficaria a vista lateral da figura 4.3b sob as mesmas condições de *rendering*, apenas mudando o *shading* para o espaço *voxel* pode ser visto na figura 4.11a. Nela, o arquivo de coordenadas foi usado para calcular as normais no espaço *voxel* no volume em níveis de cinza (antes da segmentação), e não no volume segmentado por faixa de *threshold*. Isto explica as partes escuras no *shading*, correspondendo a normais direcionadas para o interior da superfície visível, provavelmente porque a densidade dos *voxels* que ocupam o interior do crânio e que foram removidos sejam maiores que a densidade dos *voxels* na superfície visível, gerando normais com sentidos negativos. Isto fica mais claro se comparada a pele da parte externa, onde não houve problemas no *shading*, com a pele da parte interna (mais ao fundo). Aparentemente este problema deixaria de existir se, ao invés do volume em níveis de cinza, fosse usado o volume binário, onde os *voxels* fora da faixa de *threshold* foram realmente removidos, para o cálculo das normais, caracterizando o *shading* no espaço objeto. O problema surgido é visto na figura 4.11b. Neste caso, devido à faixa estreita de *threshold*, as superfícies são muito finas, possivelmente compostas por um único *voxel*, e pelo método das diferenças centrais (equação 3.2) o vetor normal resultante nestes pontos

é zero, gerando as partes escuras. Esta mesma anomalia também acontece na figura 4.11a.



(a) (b)

Figura 4.11: exemplo de dependência do tipo de shading adequado em relação ao volume de dados: (a) espaço voxel; (b) espaço objeto

4.2 Adicionando Informação de "Textura"

Informações sobre "textura" podem ser incorporadas ao *rendering* de um volume, pela utilização da rotina `vtextu`. A figura 4.12 é composta por 4 "texturas" do crânio seco obtidas de diferentes maneiras, combinando "texturas" pela média e valor máximo com deslocamento na direção das normais e na direção do observador, mas sempre caminhando 20 unidades para o interior do objeto. A primeira delas foi conseguida caminhando-se pelas normais à superfície e obtendo o valor médio; a segunda também usou o valor médio, porém na direção do observador; a terceira "textura" voltou a utilizar a direção das normais para registrar o valor máximo encontrado e a última delas resultou do valor máximo na direção do observador.

O efeito conseguido com a combinação de "texturas" no *shading* é mostrado na figura 4.13. Nela pode-se ver o *rendering* normal, sem o uso de "texturas", seguido de outros dois *renderings* que empregaram as "texturas" apresentadas à esquerda. Ambas as "texturas" foram obtidas caminhando-se 15 unidades pelas normais em direção ao interior do objeto. Na Primeira foi registrado o valor médio e na segunda o valor máximo. Observe as diferenças, principalmente na região dos dentes, onde a contribuição da "textura" é mais

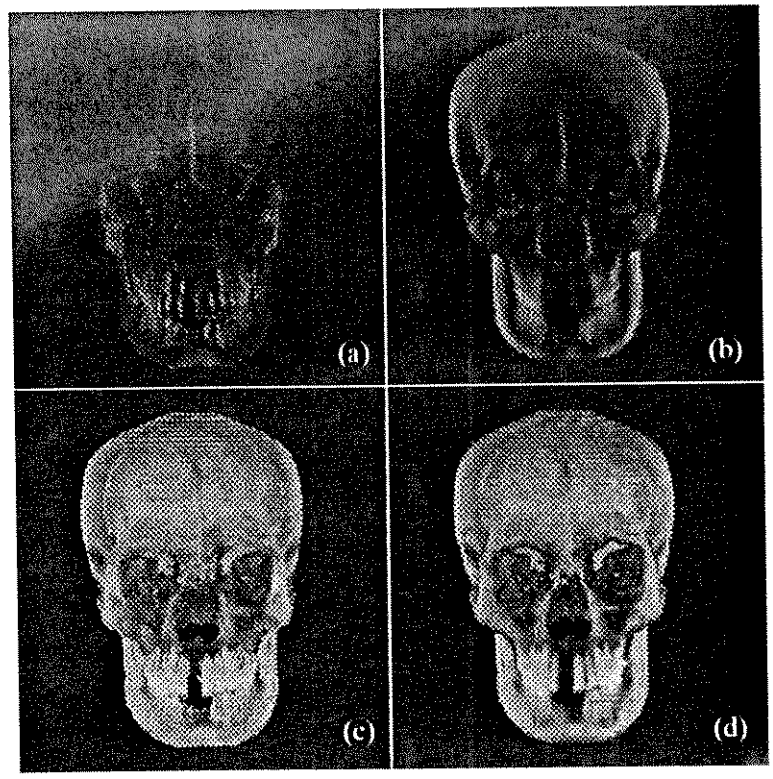


Figura 4.12: exemplo de "texturas": (a) e (b) valor médio; (c) e (d) valor máximo; (a) e (c) caminhando pelas normais; (b) e (d) caminhando pelos raios de observação

acentuada.

Pela forma com a qual é calculada, a "textura" por si só pode trazer algumas informações úteis, como por exemplo as regiões de maior e menor densidade do objeto estudado, dando à **vtextu** uma característica voltada também para a análise. Além disso, utilizando a rotina **vtextu** com parâmetros adequados, obtém-se algumas técnicas de reprojeção, conhecidas por reprojeção aditiva e reprojeção de máxima intensidade (figura 4.14). Técnicas de reprojeção são assim chamadas por simularem o processo de formação de fatias tomográficas, lançando raios do observador para o objeto, que o atravessam e registram o valor médio encontrado durante a trajetória, no caso de reprojeção aditiva, ou o valor máximo, no caso de reprojeção de máxima intensidade. Os ângulos α e β, que fazem parte do conjunto de parâmetros de **vtextu**, são usados para definir a posição do observador, quando a trajetória do raio incidente for a escolhida para a obtenção da informação de "textura". Por este motivo, recebem geralmente os mesmos valores dos ângulos usados em **vzbuff**, porém, nada impede que valores diferentes sejam usados, embora fique vago, à primeira vista, o significado desta ação.

4.3 Combinando Shading e Valores dos Voxels

A necessidade de se ter o *rendering* de volumes com bastante freqüência, levou ao desenvolvimento de uma *shell script* chamada **vrender**, que simula a região tracejada na figura 4.10, através de chamadas às rotinas envolvidas, na forma textual. Para ser usada no **Cantata**, foi definida uma interface com o usuário (veja *form* da figura A.25), segundo o formato especificado pelo Khoros. Além de tornar *workspaces* mais simples, e portanto mais claras, as *shell scripts* tornam o processamento um pouco mais veloz pois o monitoramento que o **Cantata** exerceria sobre os *glyphs* fica resumido a apenas um. Embora a comunicação entre as rotinas continuar a ser feita através da escrita e leitura dos dados em disco, as *shell scripts* permitem uma economia na utilização do espaço em disco, pois os arquivos temporários que não forem sendo mais usados, são apagados com o decorrer do processamento. Os parâmetros que estipulam os valores de *threshold* e dos ângulos de rotação e o tipo de *shading* desejado são variáveis que, na prática, acabam decidindo o resultado do *rendering* final. Esta observação conduz à definição dos parâmetros necessários

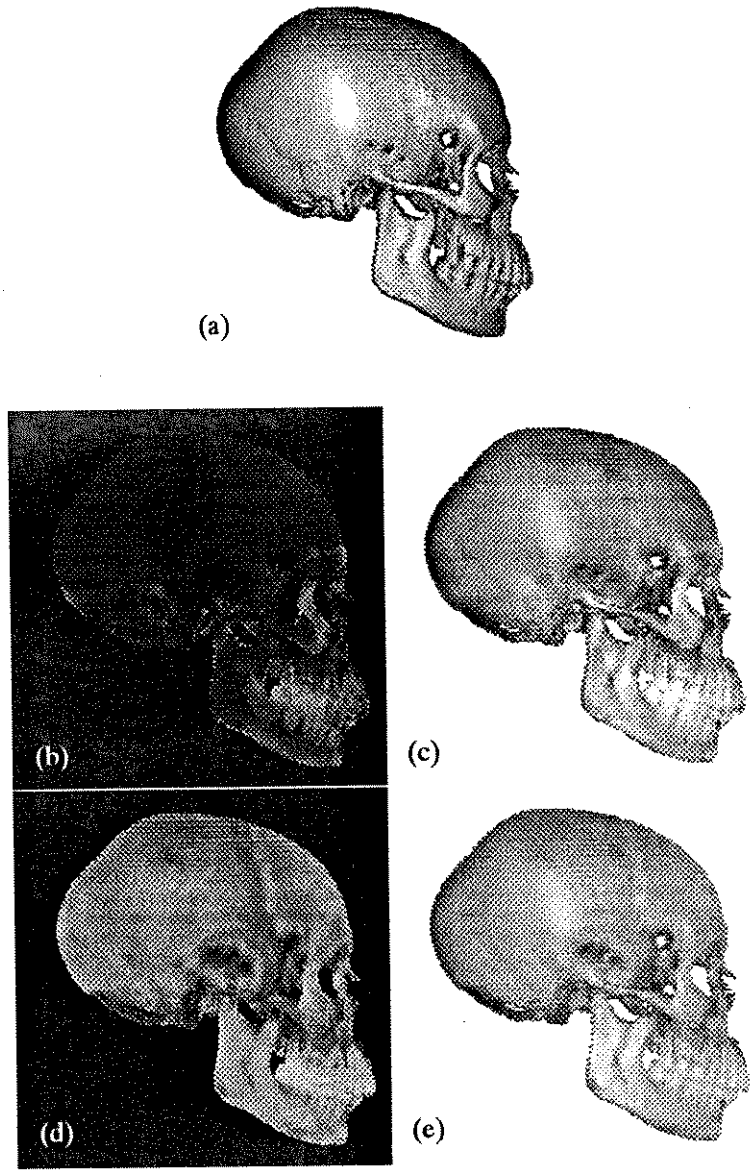
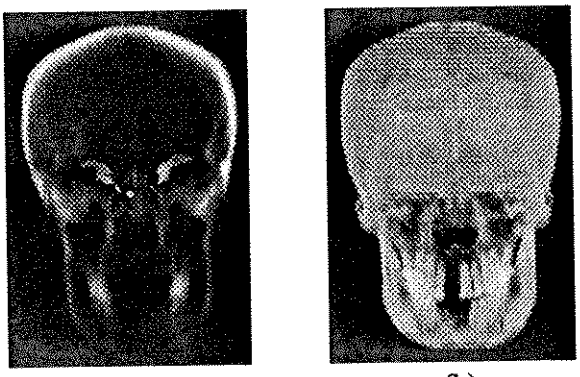


Figura 4.13: efeito da textura no rendering final: (a) rendering sem textura; (b) textura de média pelas normais; (c) rendering usando (b); (d) textura de máximo pelas normais; (e) rendering usando (d)



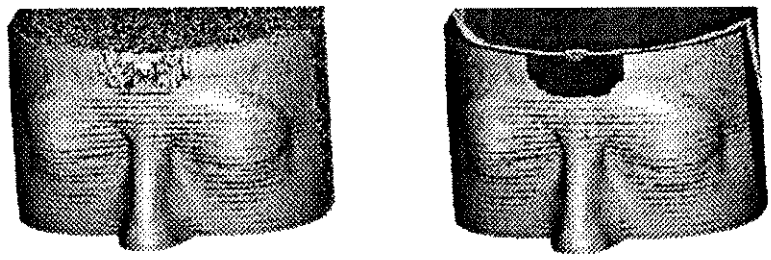
(a) (b)

Figura 4.14: técnicas de reprojeção através de vtextu: (a) aditiva; (b) máxima intensidade

à nova rotina **vrender**, que é utilizada em algumas *workspaces* mais adiante. Observe que, ao contrário da *workspace* da figura 4.10, somente um tipo de *shading* é produzido a cada vez, embora todos eles estejam implementados em **vrender**.

Na verdade, a figura 4.10 representa uma relação simplificada do processamento realizado por **vrender**. A figura 4.15 mostra *renderings* de um volume usando *voxel space shading*, onde a figura 4.15b foi resultado de **vrender** e a figura 4.15a não. A diferença é que **vrender** mostra o valor dos *voxels* nas faces de corte, ou seja, nas faces do volume/paralelepípedo que contém o objeto, caso o objeto toque nas faces do volume, indicando interseção entre eles. Note que a decisão para considerar um *voxel* como pertencente ou não ao objeto é feito de acordo com o valor de *threshold* fornecido. Como efeito, pode-se por exemplo identificar o osso (parte mais clara) na face superior e lateral direita da figura 4.15b, o que não ocorre na figura 4.15a, onde o *shading* foi feito mesmo nas regiões de corte, dando um aspecto ruidoso devido à descontinuidade na variação da direção das normais nestes pontos. Os *renderings* do volume do paciente da figura 3.2b e outras foram obtidos com esta técnica.

Para se ter uma idéia do processamento realizado por **vrender**, a figura 4.16 ilustra o caso particular, onde a normal é estimada no espaço *voxel*. O fluxo inferior é responsável pela determinação dos locais onde a densidade original do *voxel* deve ser mostrada, ao invés do valor calculado pelo algoritmo de tonalização. Isto é feito da seguinte forma: o **vzbuff** com *threshold* igual a zero gera a saída contendo o valor dos *voxels* pertencentes às faces visíveis do volume e o **vthresh3d** gera uma máscara identificando quais destes *voxels*



(a) (b)

Figura 4.15: diferença entre *renderings*: (a) normal; (b) implementado em vrender

fazem parte do objeto, usando um *threshold* igual ao usado em `vzbuff` do fluxo superior. No final os valores de tonalização calculados por `vshad` são trocados pelos valores dos *voxels* onde a máscara permitir, pela rotina `vreplace`, integrante do sistema Khoros. Este processamento fornece informações de conteúdo, ao invés de informações anatômicas, e é indicado para volumes em níveis de cinza, principalmente onde foi feito algum tipo de corte.

4.4 VOI - Visualização/Extração

Muito semelhante à *shell script* `vrender`, a `vrender2` faz exatamente o mesmo que sua antecessora, quando o segundo arquivo de entrada, que é opcional, não for fornecido (veja *form* na figura A.26). A entrada opcional, se usada, deve ser um volume de dimensões iguais às do volume principal, e sua função é definir um volume de interesse, sendo que somente a porção dos dados que estiverem dentro da região indicada por *voxels* de densidade maior que zero são levados em consideração para o *rendering*, mantendo as características apresentadas em `vrender` quanto ao *display* dos valores dos *voxels* que estiverem sobre a superfície do volume de interesse. O caso particular do processamento de `vrender2` para um *rendering* usando normais estimadas no espaço *voxel* é mostrado na figura 4.17. As coordenadas do volume de interesse foram obtidas pela rotina `vzbuff` do fluxo inferior com *threshold* igual a 1. Perceba a necessidade do uso da rotina `vvoxext` para obter a densidade dos *voxels* no volume de dados segundo a superfície do volume de interesse, definido na entrada opcional. A figura 4.18 é uma animação conseguida com o uso de volumes de interesse cada vez menores.

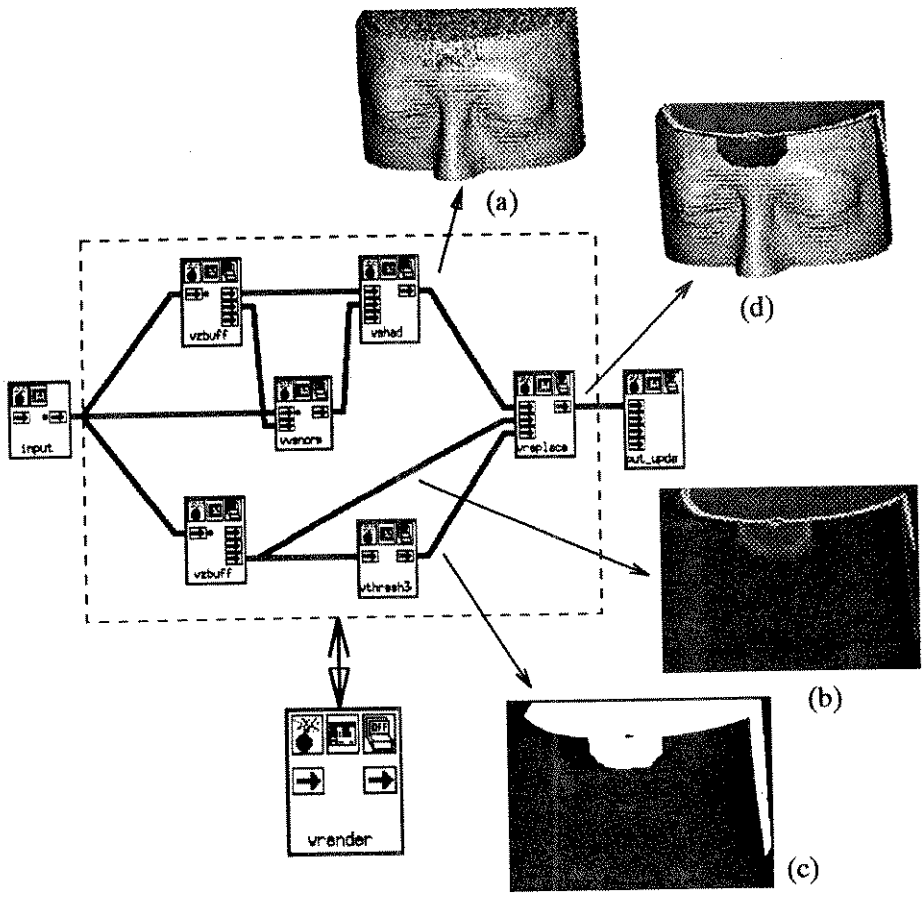


Figura 4.16: exemplo de processamento efetuado por vrender: (a) shading tradicional; (b) voxels na fronteira do volume; (c) máscara dos voxels acima do threshold; (d) resultado final

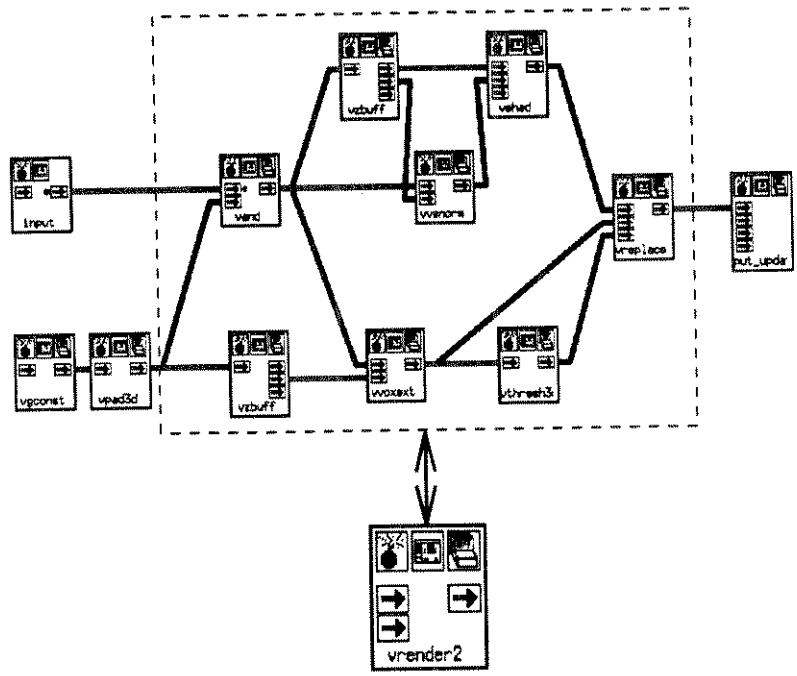


Figura 4.17: exemplo de processamento efetuado por vrender2

Volumes de interesse podem sem efetivamente extraídos do volume original, bastando que para isto sejam fornecidas as coordenadas opostas em relação à diagonal principal do paralelepípedo desejado. A vantagem é que ocorre redução no volume de dados a ser processado para a realização do *rendering*, o que torna o processo mais rápido, além de não haver necessidade da criação de um volume de interesse de mesmas proporções que o volume original. A implementação é feita através da combinação das rotinas `vextract` do Khoros e `vaband` da *toolbox*. A primeira delas reduz o volume nas direções x e y e a segunda reduz o volume na direção z. A figura 4.19 mostra uma aplicação prática, onde o maxilar inferior foi isolado do volume original contendo o crânio seco, para depois ser feito o *rendering*. A definição dos pontos para delimitar o volume de interesse pode ser feita utilizando o recurso de declaração de variáveis do Khoros, que se encarrega de passar os valores adequados como parâmetros para as rotinas envolvidas.

Explorando um pouco mais a combinação de *shadings* e densidade dos *voxels* em volumes limitados por algum tipo de volume de interesse, não é difícil de imaginar que pequenas modificações em `vrender2` são capazes de gerar uma infinidade de possibilidades,

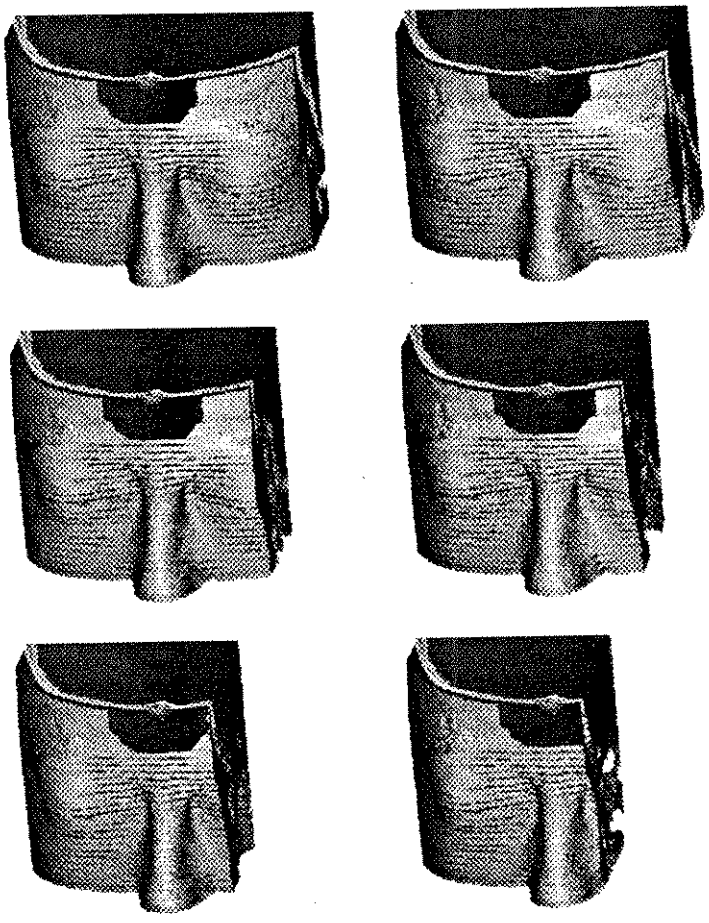


Figura 4.18: animação variando o volume de interesse em vrender2

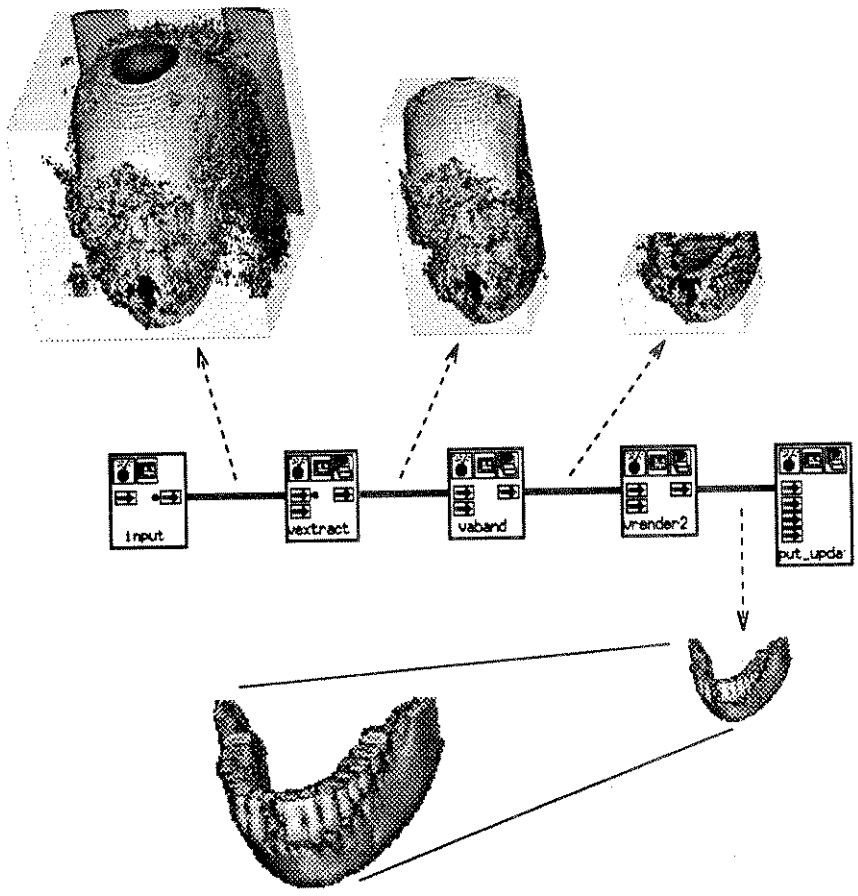


Figura 4.19: extração de volume de interesse isolando maxilar inferior

dependendo das combinações que são feitas, como mostra a figura 4.20. Nela podem ser vistos os efeitos conseguidos com uma esfera de densidade interior que segue a distribuição gaussiana e um cubo com densidade constante. As três primeiras figuras são os *renderings* dos objetos e da união dos dois. As quatro figuras intermediárias consideram a esfera como objeto e o cubo como volume de interesse. Da esquerda para a direita, são mostrados o *rendering* do objeto com seus *voxels* na área "cortada", o *rendering* do volume de interesse com os *voxels* do objeto na área "cortada", o *rendering* do objeto com os *voxels* do volume de interesse na área "cortada" e por último o *rendering* do volume de interesse com seus *voxels* na área "cortada". As quatro últimas figuras seguem a mesma idéia da seqüência anterior, porém o cubo é considerado como o objeto, enquanto que a esfera faz as vezes do volume de interesse.

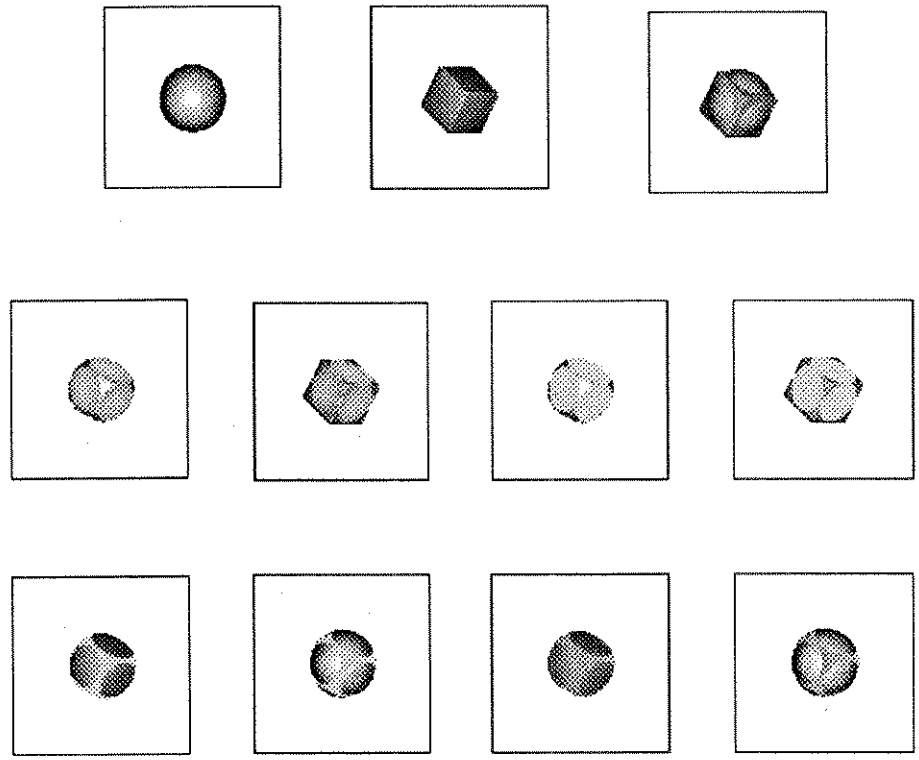


Figura 4.20: efeitos da combinação de densidade de voxels e shadings em renderings de volumes

Obviamente, os resultados que podem ser alcançados com o uso de volumes de interesse dependem da facilidade de se criar volumes sintéticos, o que será visto mais

adiante, na seção 4.8.

4.5 Visualizando Planos Ortogonais

O uso de `vvoxext` em `vrender2` para obter o valor dos *voxels* de um objeto segundo a superfície de um outro objeto se constitui, por si só, uma fonte extensa de métodos de visualização. Por exemplo, se combinada com a rotina `vorthog` e `vzbuff`, como mostrado na figura 4.21, é possível visualizar planos ortogonais de um volume sob uma posição de observação no espaço tridimensional. Os mesmos planos podem ser obtidos e vistos por inteiro e de frente pela rotina `vplanexyz`. Neste último caso não ocorrem interferências causadas pela perspectiva, mas há perda na noção da relação espacial que existe entre os planos.

Ainda com relação ao resultado da *workspace* da figura 4.21, é possível, fazendo uso de processamentos extras bem simples, modificar a apresentação da imagem final, como mostra a figura 4.22. A possibilidade de escolha de diferentes níveis de cinza para as arestas dos planos ortogonais criados por `vorthog`, por exemplo, permite realçar as bordas dos planos, o que é conseguido com um *threshold* adequado, aplicado sobre o arquivo de densidade de *voxels* gerado por `vzbuff`, e depois seguido da rotina `vreplace`, responsável pela troca dos valores dos *pixels* da borda na imagem final, de forma análoga à que é feita nas *workspaces* das figuras 4.16 e 4.17. De maneira semelhante pode-se mudar a cor de fundo, facilmente detectável por apresentar valores negativos no arquivo de *z-buffer* gerado por `vzbuff`, ou até mesmo realçar a tridimensionalidade dos planos, combinando o *rendering* do volume gerado por `vorthog` com a densidade dos *voxels* através de uma outra rotina do Khoros, chamada `vblend`, que faz uma espécie de fusão de duas imagens, de acordo com a porcentagem de contribuição atribuída a cada uma delas, como equacionado em 4.1.

Equação de fusão de imagens implementada em `vblend`:

$$Img(u,v) = \alpha\ Img_1(u,v) + (1-\alpha)\ Img_2(u,v) \tag{4.1}$$

onde:
$Img(u,v)$ é o valor do *pixel* (u,v) na imagem resultante

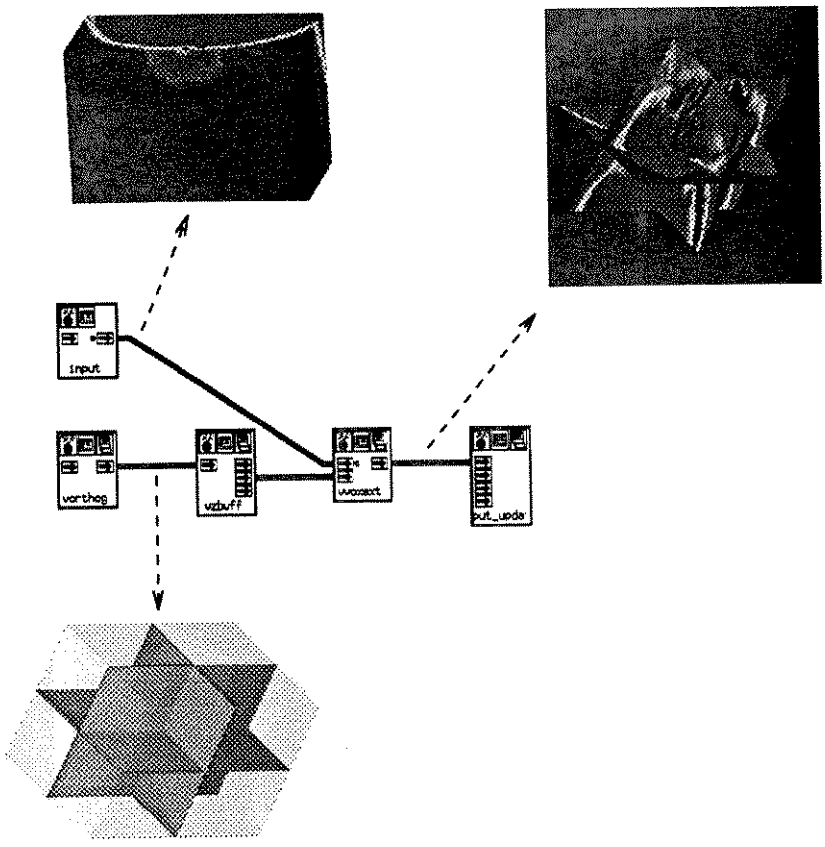


Figura 4.21: aplicação típica da rotina vvoxext

$Img_1(u,v)$ é o valor do *pixel* (u,v) na primeira imagem
$Img_2(u,v)$ é o valor do *pixel* (u,v) na segunda imagem
α é o fator de contribuição das imagens

4.6 Representando um Volume 3D

A mesma técnica empregada em 4.22c, ou seja, de combinar duas imagens em uma única, foi largamente utilizada em muitas das figuras do capítulo 3, quando se desejou enfatizar a noção de dados inseridos em um volume, ao invés de um simples *rendering*, ou mesmo fornecer a posição relativa entre o volume e o observador, ou entre o objeto e o volume que o contém (figura 4.23). O volume envolvente é determinado pelo *rendering* do volume original com *threshold* igual a zero (figura 4.23a). O mesmo efeito pode ser obtido usando a rotina `vframe` para criar um volume contendo somente as arestas, onde o volume de dados é posteriormente inserido com o uso da rotina `vinsert3d` para depois ser feito o *rendering* (veja figura 4.23b). Nada impede também que as duas técnicas sejam combinadas, resultando num *rendering* onde se vê o objeto, as faces do volume que o envolve e as arestas bem definidas.

A idéia de transparência na técnica que mostra as faces do volume envolvente na figura 4.23a não é completa, uma vez que as faces de trás deveriam ser vistas nestas condições, como colocado na figura 4.23c. O problema se resolve com um *shading* extra de um volume contendo somente os planos da parte de trás (criado por `vorthog`), combinado com o já existente (veja figura 4.24). Uma máscara obtida por *threshold* igual a zero no arquivo de *z-buffer* dos dados, dizendo a região onde a rotina `vreplace` deve fazer a correção desejada, determina a imagem final.

4.7 Visualizando Dados sem Coerência Espacial

Outro método de visualização conseguido com a combinação de operadores básicos é mostrado na figura 4.25. O volume de dados representa cortes transversais de rocha, para estudo de porosidade. Pelo fato destes dados terem sido obtidos com um espaçamento muito grande entre si, um simples *rendering* se torna inviável, pois a interpolação

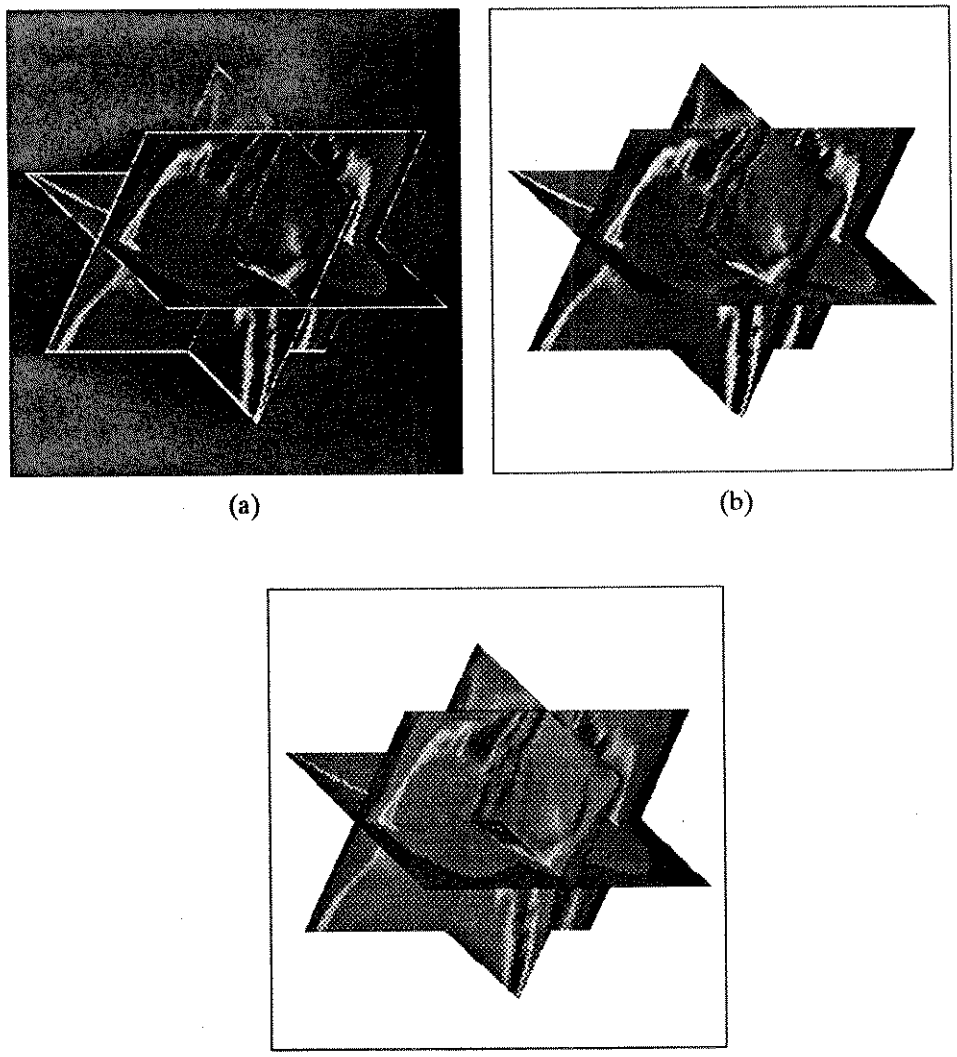
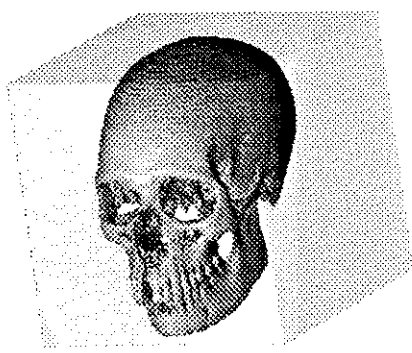
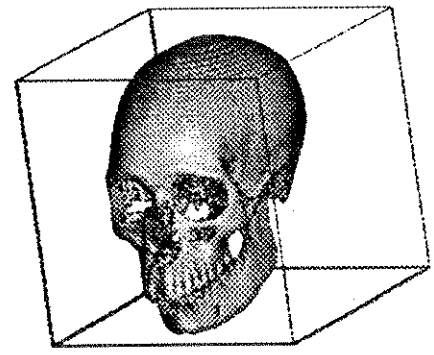


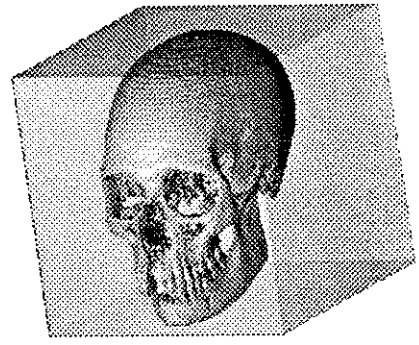
Figura 4.22: variações no display obtidas com processamento extra: (a) realce de bordas; (b) mudança na cor de fundo; (c) realce da noção tridimensional



(a)



(b)



(c)

Figura 4.23: efeitos de dados ocupando o interior de um volume: (a) com faces; (b) com arestas; (c) correção feita em (a) para mostrar faces de trás

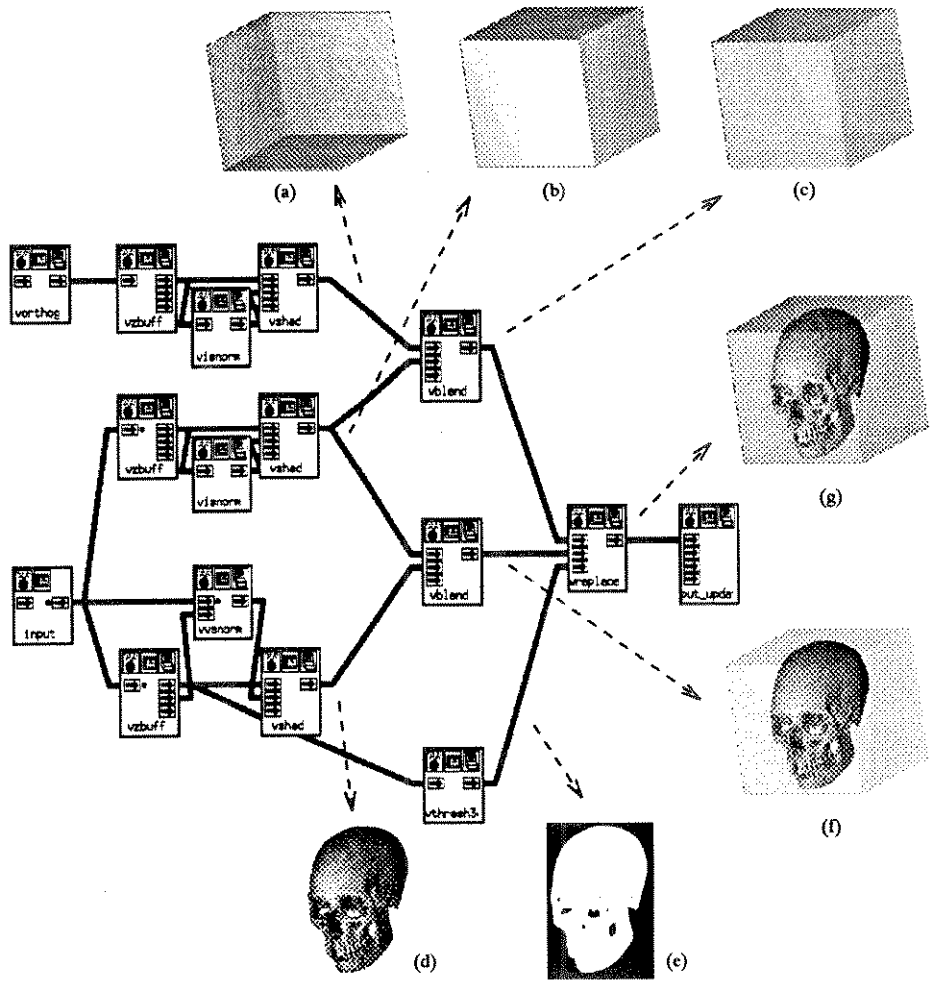


Figura 4.24: composição de imagens para efeitos de tridimensionalidade: (a) faces de trás; (b) faces da frente; (c) composição de (a) e (b); (d) objeto; (e) máscara; (f) composição de (b) e (d); (g) composição final de (c) e (f) segundo a máscara

não se aplica neste caso. O problema é ainda maior quando o número de bandas é pequeno. Uma solução seria usar a rotina `vmontage`, como na figura 4.25a, onde todas as bandas são vistas simultaneamente, ou então usar a *workspace* em 4.25c. O resultado está na figura 4.25b, que representa as bandas empilhadas umas sobre as outras, porém espaçadas, de maneira a ser mais fiel à realidade dos dados. As linhas que interligam os vértices das bandas foram colocadas propositalmente para efeito de orientação espacial, e ilustram o uso de `vimpul3d`, na criação dos volumes que foram inseridos entre as bandas do volume original pela rotina `vaband`. O resultado final é obtido na saída de `vzbuff` que fornece a densidade dos *voxels*. A justificativa do uso das rotinas `vthresh3d` e `vor` é a de tornar o fundo branco. Observe que esta forma de visualização tem aplicações muito restritas, uma vez que aumenta demasiadamente o volume original antes de ser processado por `vzbuff`.

4.8 Volumes Sintéticos

A seguir são apresentados alguns volumes sintéticos, que podem ser usados, por exemplo, para determinar volumes de interesse e implementar filtros diversos, como será visto adiante. Além da flexibilidade de `vframe`, `vimpul3d` e `vorthog` na sintetização de volumes, a rotina `vgauss3d`, pela variedade de parâmetros de entrada, possibilita inúmeras combinações de valores que permitem obter uma infinidade de resultados. Esferas, como as da figura 4.7, e elipsóides de tamanhos e distâncias focais diversas, podendo ocupar qualquer posição e orientação dentro do volume, são conseqüências imediatas dos parâmetros adotados. A figura 4.26 mostra uma animação obtida pela variação do *threshold* em um volume contendo três distribuições gaussianas. Observe que os objetos, mesmo quando estão separados uns dos outros, não são esferas perfeitas pois as distribuições se somam. Para que três esferas ocupassem o mesmo volume, as distribuições deveriam ser geradas independentemente e combinadas no mesmo volume de forma que cada *voxel* do volume resultante fosse equivalente ao valor máximo das distribuições, ao invés da soma delas. Isto é facilmente implementado usando as rotinas `vsub` (subtração) do Khoros para determinar os máximos, `vtrhesh3d` para criar a máscara e `vreplace` para criar o volume final.

Mais volumes sintéticos gerados pela rotina `vgauss3d` podem ser vistos na figura 4.27. Todas elas, com exceção da 4.27c resultaram da concatenação (por `vaband`)

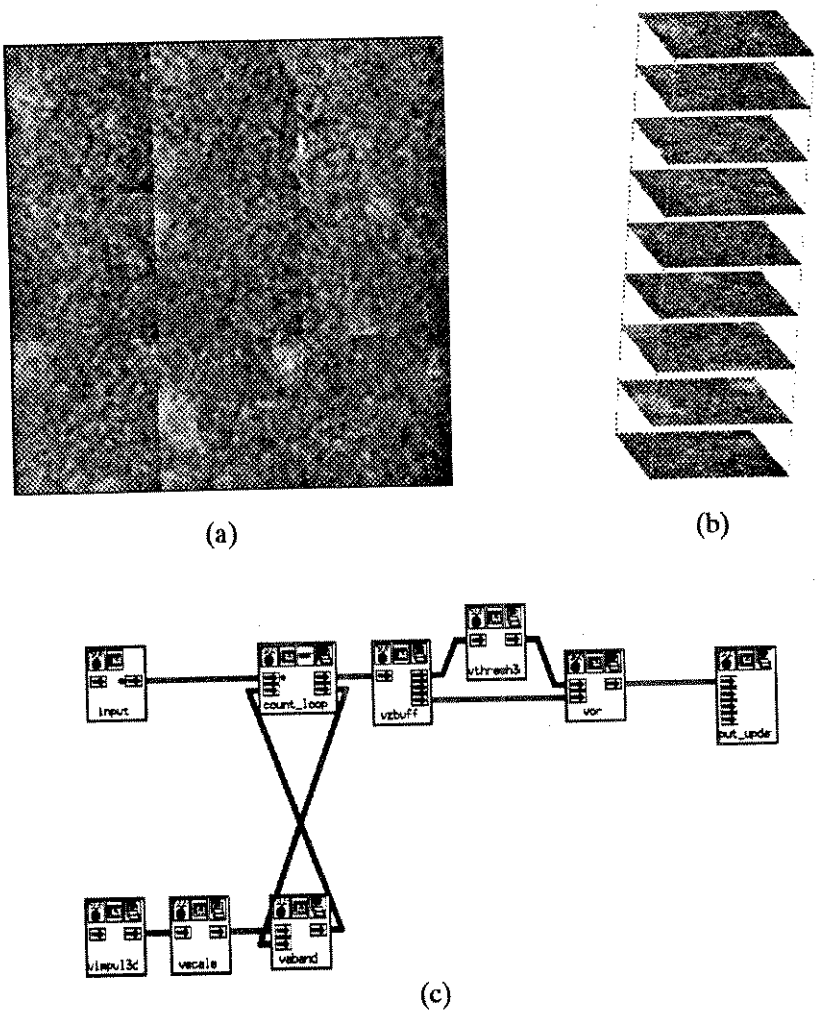


Figura 4.25: visualização de dados muito espaçados: (a) lado-a-lado pelo uso direto de vmontage; (b) empilhados, usando a workspace em (c)

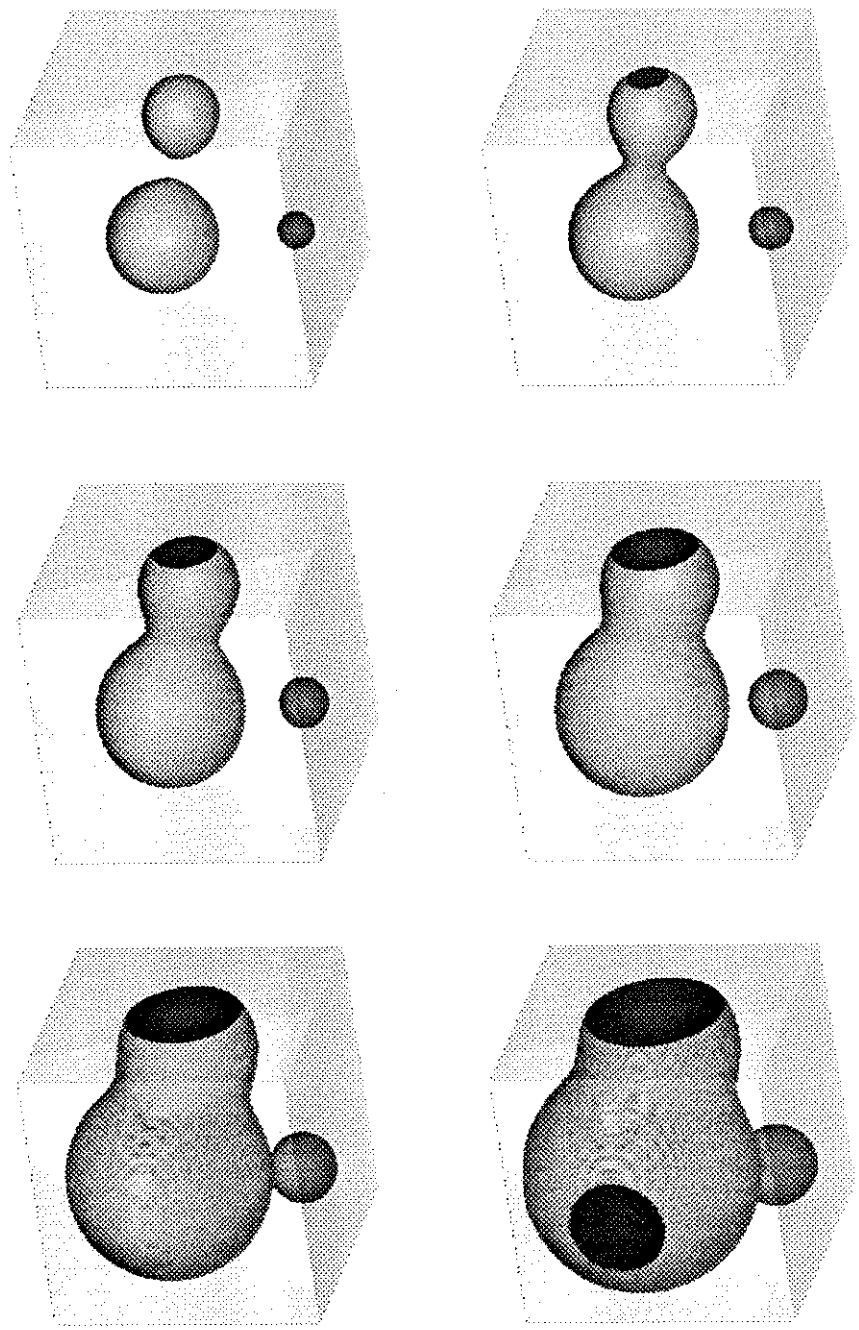
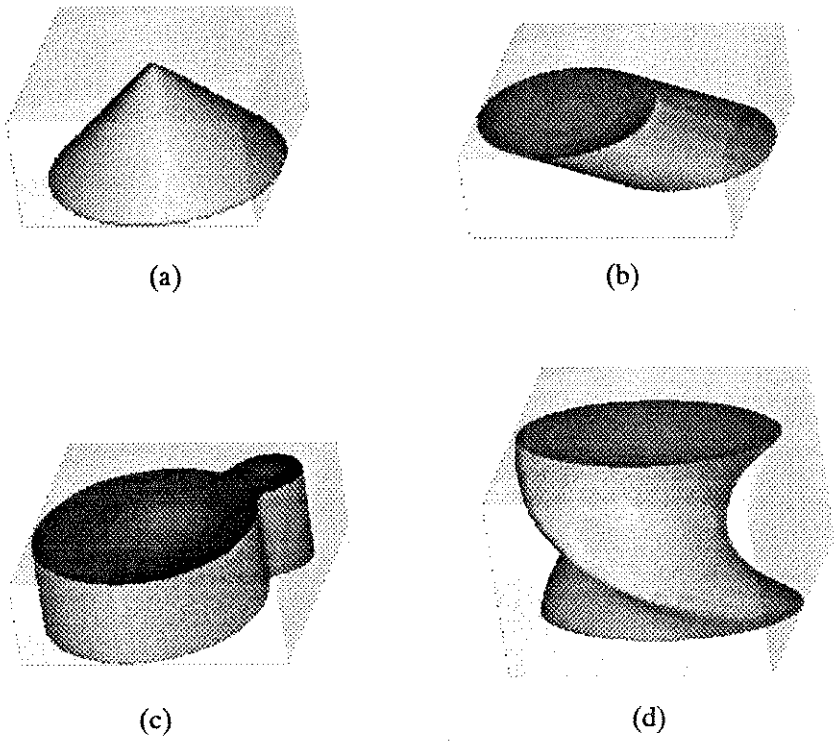


Figura 4.26: animação de threshold em volume sintético criado por vgauss3d

de sucessivas bandas geradas seqüencialmente. O cone em 4.27a foi criado com variação decrescente da variância em x, y e z, de forma que, após o *threshold*, círculos de raios cada vez menores foram sendo empilhados a cada passo. No cilindro "inclinado" da figura 4.27b houve apenas um deslocamento no centro das distribuições a cada banda gerada. O volume da figura 4.27c é o mais simples de todos, pois a mesma banda contendo duas distribuições diferentes foi expandida na direção z pela rotina **vexpand3d**. Repare que na face superior é possível identificar o centro das distribuições em tons mais claros, devido ao uso de **vrender** para mostrar a densidade dos *vozels* nas faces do volume envolvente, quando feito o *rendering*. O volume da figura 4.27d foi obtido da concatenação de distribuições onde a covariância, expressa pelo ângulo de rotação, foi mudada. A forma elíptica é obtida com valores diferentes para as variâncias em x e y. Usando rotinas do Khoros que desempenham funções lógicas pode-se fazer complementos, interseções, uniões e diferenças destes e tantos outros volumes, ampliando significativamente as possibilidades de geração de volumes sintéticos, como o exemplo da figura 4.28, que ilustra um volume resultante da união de três cilindros e seu complemento, que também pode ser interpretado como a diferença entre um cubo e o volume em 4.28a.

4.9 Exemplos de Variações no Pré-Processamento

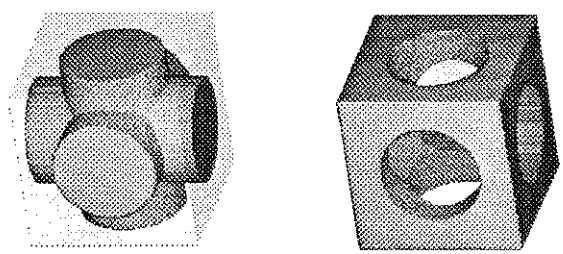
Seguindo a filosofia introduzida em Udupa e Gonçalves [32], uma série de pré-processamentos podem ser feitos sobre um volume de dados antes do *rendering*, apenas mudando a ordem em que eles são aplicados. A flexibilidade na adoção de uma ou outra metodologia só é possível quando se tem acesso a estágios intermediários do processamento e quando, de alguma maneira, pode-se decidir o próximo passo a ser dado. O conjunto de operadores apresentado pela *toolbox* V3DTOOLS possui esta característica e pode, portanto, ser usado em operações de pré-processamento, da mesma forma que oferece múltiplas possibilidades na visualização, conforme colocado até o momento, através da combinação de suas rotinas.



(a) (b)

(c) (d)

Figura 4.27: volumes sintéticos criados por vgauss3d em conjunto com vaband



(a) (b)

Figura 4.28: operações lógicas entre volumes para geração de novos volumes: (a) união de cilindros; (b) complemento de (a) ou diferença entre cubo e (a)

4.9.1 Exemplo A - interpolação, filtragem, segmentação, filtragem

A primeira variação no pré-processamento é ilustrada na figura 4.29. Nela, após ter sido interpolado (I), o volume de dados passa por uma seqüência envolvendo filtragem (II), segmentação (III) e nova filtragem (IV). Obviamente o processamento poderia ser encerrado nos pontos indicados por (II) ou (III) na figura citada, tendo como resultado final o que é tido como intermediário neste caso. Nas filtragens foram usadas as rotinas `vconv3d` para realizar uma convolução 3D com núcleo de média 5x5x5, e a segmentação foi feita por *threshold*, usando `vthresh3d`. O núcleo foi criado a partir do núcleo bidimensional 5x5, também para média, disponível no Khoros, com o uso de `vscale` para acertar o valor dos elementos do núcleo 2D (foram divididos por 5), e de `vexpand3d` para criar as outras quatro bandas.

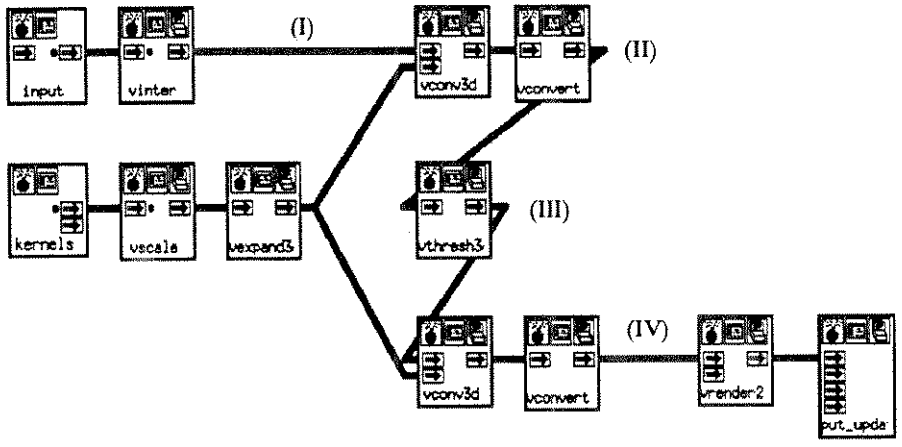


Figura 4.29: pré-processamento A: interpolação, filtragem, segmentação, filtragem

Os resultados conseguidos com a *workspace* da figura 4.29 nos pontos assinalados (I), (II) e (IV) podem ser vistos na figura 4.30, quando aplicada ao volume contendo o crânio seco. Em seqüência horizontal estão os *renderings* do mesmo volume, usando normais estimadas no espaço *voxel*, imagem e objeto, respectivamente. Em 4.30a estão os *renderings* após a interpolação (ponto I); em 4.30b estão os *renderings* do volume filtrado pela primeira vez (ponto II); e por último, em 4.30c, os *renderings* do volume resultante em IV. Perceba que a cada passo, as formas vão se tornando mais suaves, principalmente nas regiões mais ricas em detalhes, como é o caso do nariz e principalmente dos dentes.

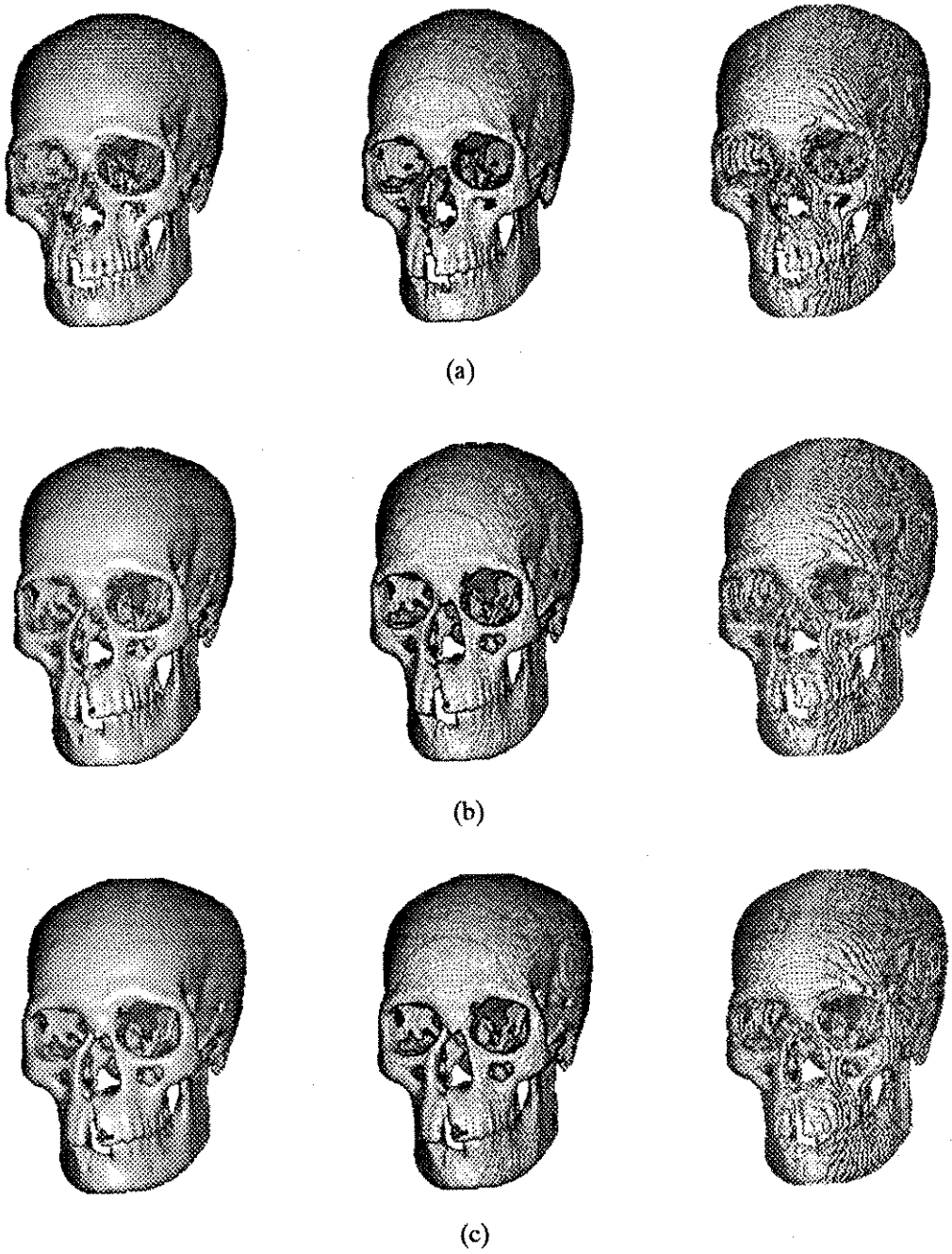


Figura 4.30: resultados do pré-processamento A: (a) após interpolação (ponto I); (b) após primeira filtragem (ponto II); (c) após segunda filtragem (ponto IV)

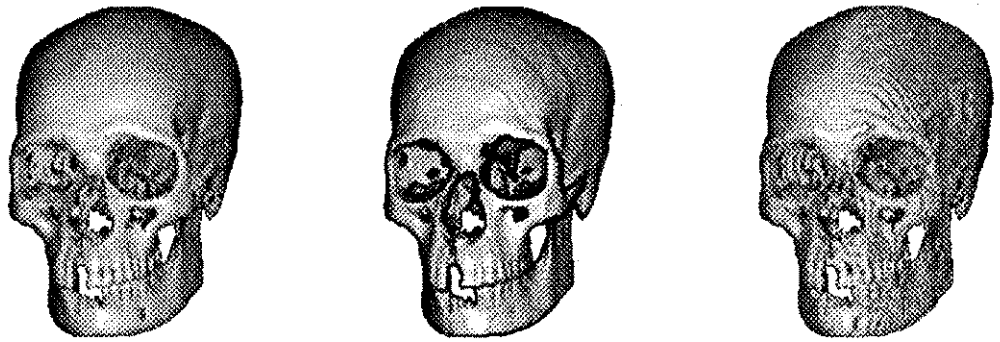
A filtragem pode ocorrer também a nível das normais estimadas. Neste caso, aos pontos vizinhos que não possuem coerência espacial, são associados erros em taxas mais elevadas. A figura 4.31 apresenta os mesmos *renderings* da figura 4.30 com normais filtradas por convolução 2D com núcleo de média 3x3, aplicado separadamente em cada uma das bandas. Observe que as anomalias causadas nas bordas dos *renderings* no espaço imagem são agravadas, tornando-se mais evidentes. É bom lembrar que o processamento das normais, a exemplo da filtragem aplicada aqui, não é possível na versão inicial da *toolbox* V3DTOOLS, pois as normais são dados internos da rotina `zbuffer`.

4.9.2 Exemplo B - segmentação, interpolação, filtragem

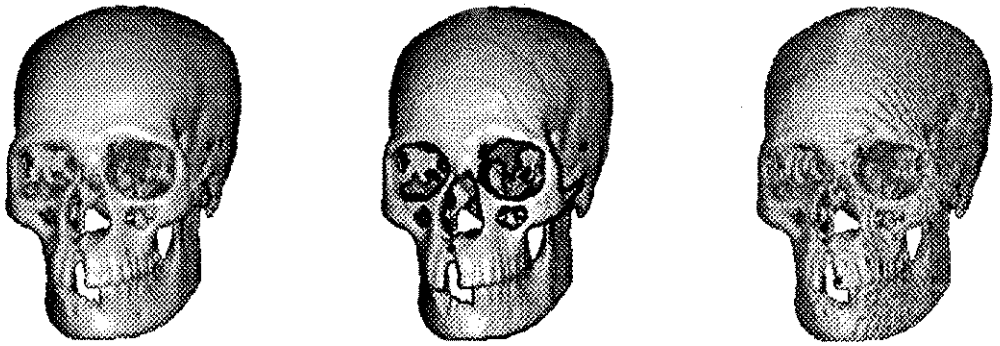
Uma outra variação possível consiste em segmentar, interpolar e depois filtrar o volume, como na *workspace* da figura 4.32. O núcleo utilizado neste caso também foi de média 5x5x5. A segmentação por *threshold* não está explícita na figura por estar embutida na rotina `vinter`, que neste caso realiza uma interpolação binária linear do tipo euclideana baseada na forma. Os resultados obtidos se encontram nas figuras 4.33 e 4.34, sendo que a diferença entre elas reside no fato de que na figura 4.34 foi realizada uma filtragem extra das normais com um núcleo de média 3x3. Em ambas, a parte (a) mostra o resultado após a interpolação (ponto I da *workspace*), com *renderings* estimando as normais no espaço imagem e no espaço objeto (espaço *voxel* não se aplica, pois trata-se de volume binário) e a parte (b) mostra o resultado no ponto indicado por II, após a filtragem final, com *renderings* no espaço *voxel*, imagem e objeto, respectivamente.

4.9.3 Exemplo C - segmentação, filtragem, interpolação

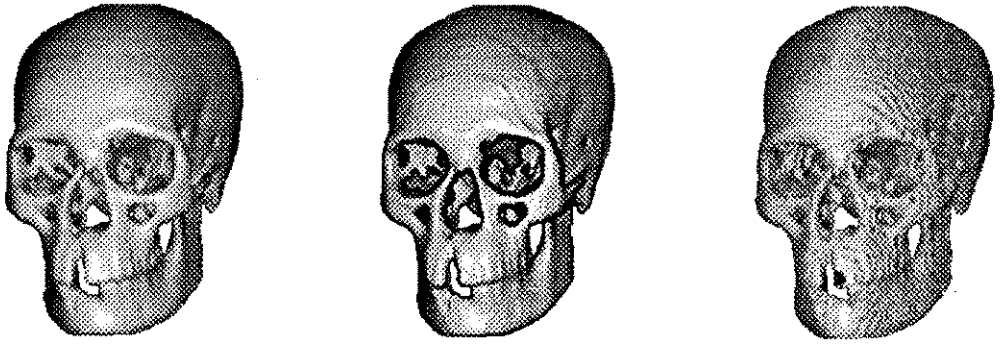
O último exemplo de combinações de rotinas para pré-processamento é visto na *workspace* da figura 4.35. Após a segmentação com `vthresh3d` (ponto I), o volume é filtrado (ponto II) utilizando a rotina `vconv3d` com núcleo de média 7x7x2 e logo em seguida interpolado linearmente em níveis de cinza por `vinter` (ponto III). As dimensões do núcleo de convolução são diferentes daquelas usadas nos dois outros exemplos para ser coerente com as proporções do volume de dados, uma vez que a filtragem é feita no volume ainda não interpolado. Os resultados obtidos no ponto III estão na figura 4.36, onde em (a)



(a)



(b)



(c)

Figura 4.31: resultados do pré-processamento A com filtragem de média 3x3 nas normais usando convolução 2D: (a) após interpolação (ponto I); (b) após primeira filtragem (ponto II); (c) após segunda filtragem (ponto IV)

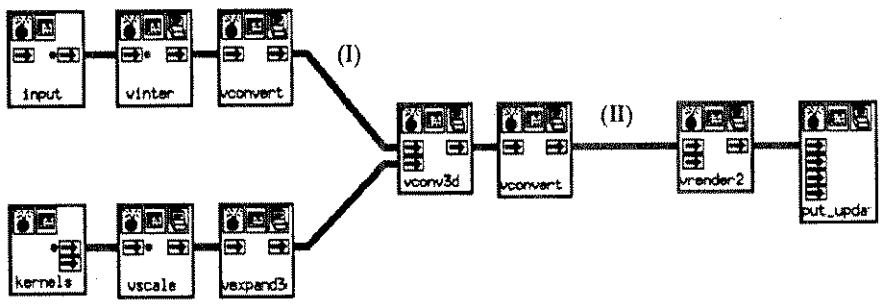


Figura 4.32: pré-processamento B: segmentação, interpolação, filtragem

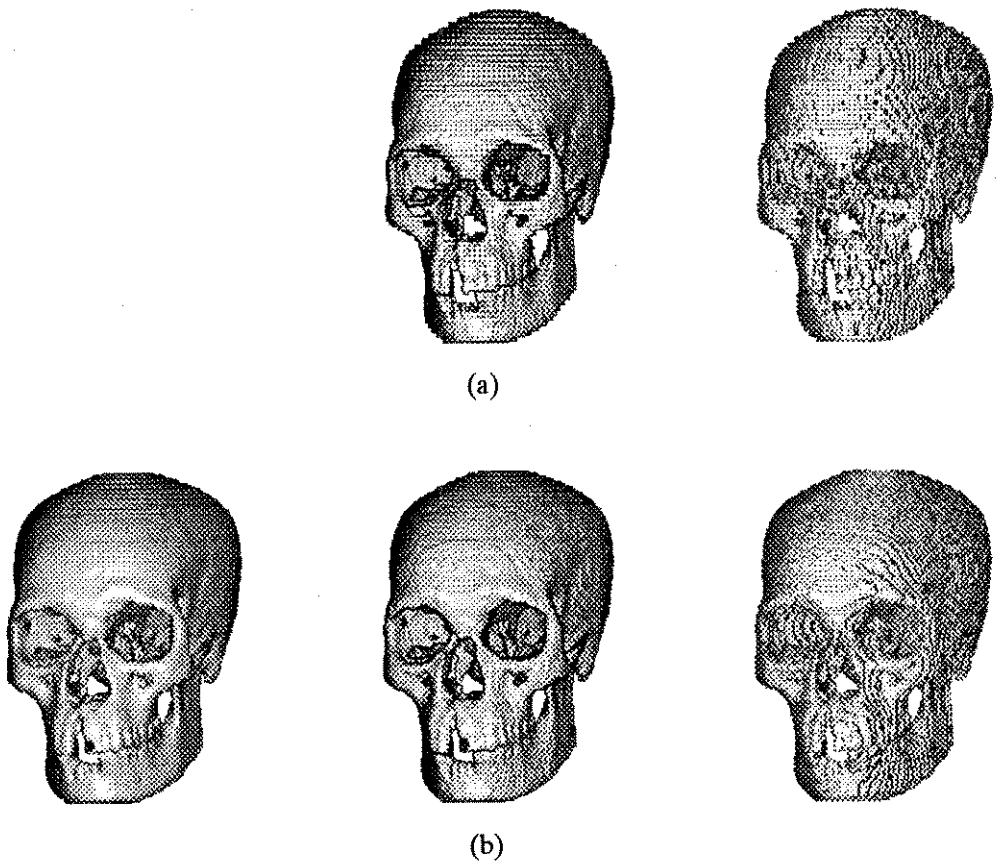
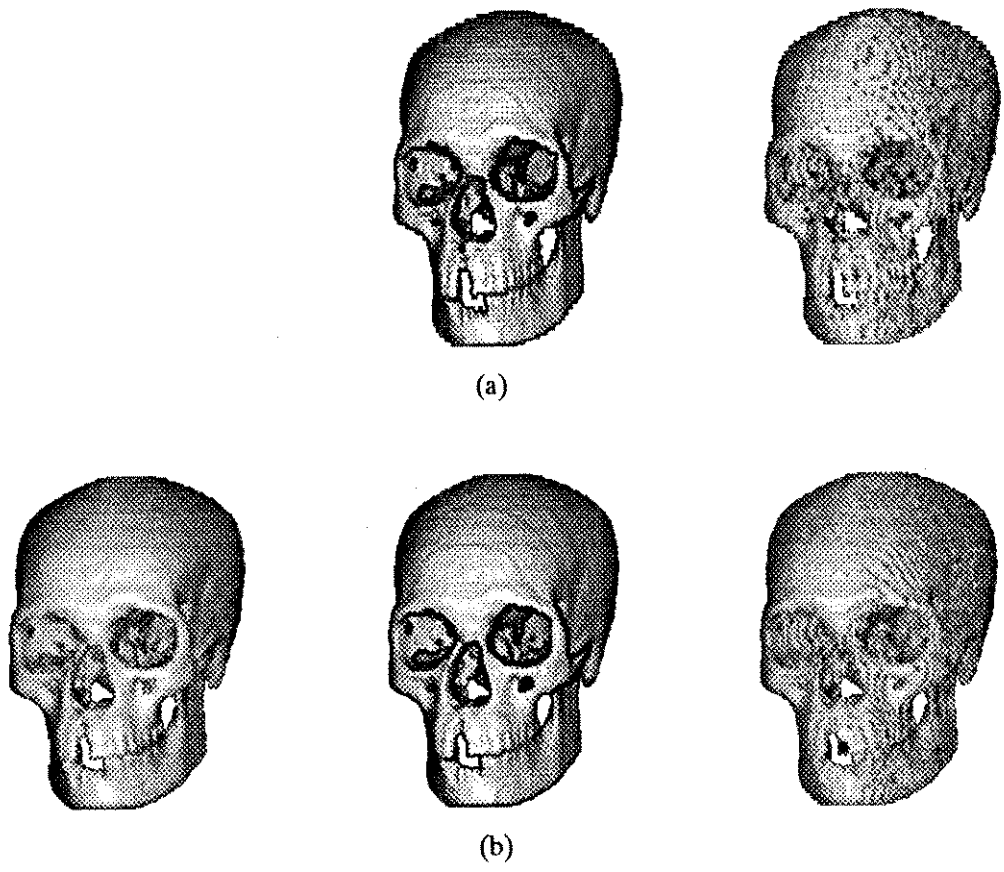


Figura 4.33: resultados do pré-processamento B: (a) após interpolação (ponto I); (b) após filtragem (ponto II)



(a)

(b)

Figura 4.34: resultados do pré-processamento B com filtragem de média 3x3 nas normais usando convolução 2D: (a) após interpolação (ponto I); (b) após filtragem (ponto II)

são apresentados os *renderings* com normais calculadas no espaço *voxel*, imagem e objeto, respectivamente, e em (b) são mostrados os mesmos *renderings*, porém com as normais filtradas por convolução 2D com núcleo de média 3x3.

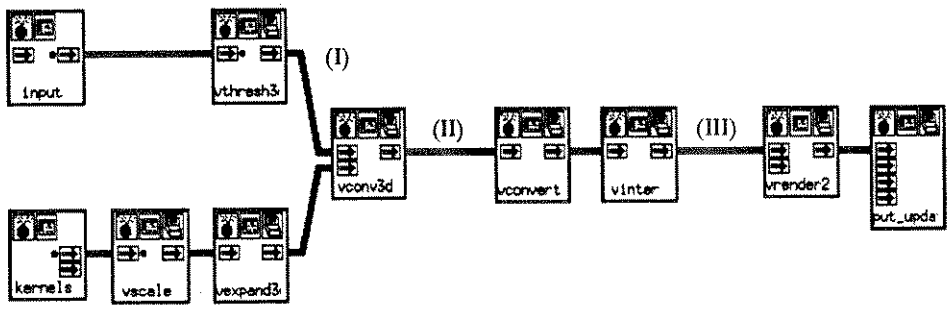


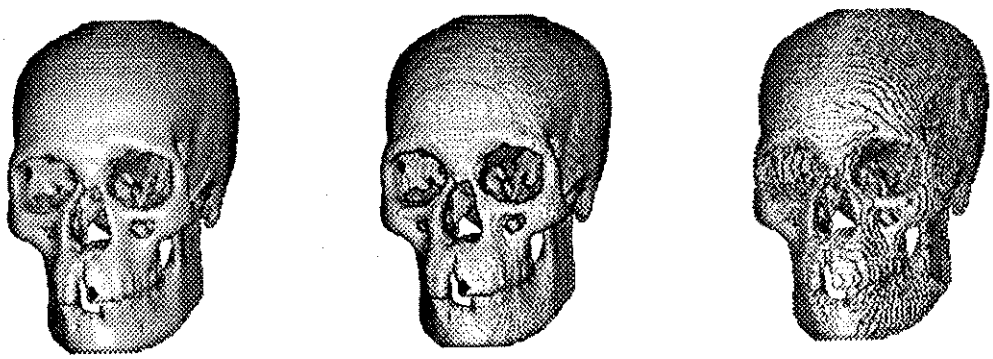
Figura 4.35: pré-processamento C: segmentação, filtragem, interpolação

4.9.4 Conclusão

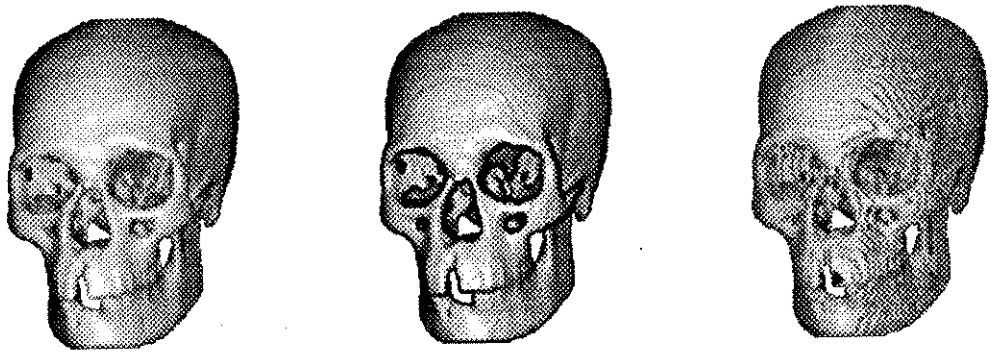
As três operações de pré-processamento que acabaram de ser apresentadas não têm a intenção de avaliar qual delas oferece um melhor resultado, ainda mais porque a definição de um bom resultado depende da finalidade para o qual o pré-processamento foi feito, e está relacionado com a origem dos dados. O objetivo destes exemplos foi o de ilustrar, mais uma vez, as inúmeras vantagens de se poder trabalhar com operadores básicos. Apesar disso, algumas delas são indicadas e empregadas em várias situações, visando uma maior rapidez ou melhor qualidade de *rendering* como discutido a seguir.

Um volume de dados em níveis de cinza é classificado como *hard* quando a superfície a ser visualizada possui alto contraste com o restante do volume, possibilitando que ela seja detetada por *threshold*. Nestes casos, o *shading* que usa as normais estimadas no espaço *voxel* oferece melhores resultados, e o processo de deteção da superfície é deixado para o final, ao ser feito o *rendering*. Este é exatamente o exemplo do pré-processamento A (figura 4.29), cujo resultado é o *rendering* mais à esquerda (normais estimadas no espaço *voxel*) da figura 4.30a, equivalente à saída no ponto I da *workspace*. Opcionalmente, após interpolado, o volume pode ser filtrado (veja o *rendering* mais à esquerda da figura 4.30b) ou ter seus vetores normais filtrados, como o *rendering* mais à esquerda da figura 4.31a.

Se o volume não for *hard*, na prática o que se faz é segmentá-lo manualmente



(a)



(b)

Figura 4.36: resultados do pré-processamento C após interpolação (ponto III): (a) sem filtragem das normais; (b) com filtragem das normais

logo de início, antes de aplicar a interpolação do tipo *shape based*. Isto se deve ao fato de que o número de fatias é menor no volume original, antes da interpolação, e o processo de segmentação manual torna-se menos trabalhoso. Este é o caso da *workspace* do pré-processamento B na figura 4.32. Embora o volume de dados utilizado seja *hard*, deve-se, a título de exemplo, supor que a segmentação tenha sido feita manualmente e não de forma automática e interna a `vinter`. O resultado obtido (ponto I) está na figura 4.33a. Sendo o volume interpolado também binário, os dois únicos tipos de *shading* possíveis usam normais no espaço imagem (à esquerda) e objeto (à direita). O *shading* no espaço objeto é o mais pobre de todos e deve ser evitado. Por sua vez, o *shading* no espaço imagem é o mais indicado dos dois. Para transformar um volume binário em um volume *hard* cinza, deve-se filtrar o volume binário. Como visto, os volumes *hard* possibilitam melhores *shadings*, pois o gradiente no espaço *voxel* aproxima melhor os vetores normais nestes casos. Na figura 4.32, o volume *hard* é obtido no ponto II. O *rendering* mais à esquerda da figura 4.33b mostra o resultado conseguido. Mais uma vez, a filtragem das normais é um processamento opcional que pode ser aplicado (veja *renderings* correspondentes na figura 4.34).

A transformação de um volume binário em *hard* cinza poderia ter sido feita no volume da figura 4.11, porém, conforme explicado na página 58, algumas partes da superfície do objeto segmentado são muito finas, e podem causar erros na estimativa das normais, mesmo depois de filtrado. Este é um caso particular, onde o *shading* no espaço imagem é a única opção disponível, como mostrado na figura 4.3.

Resumindo, se o volume for *hard* cinza, deve-se interpolá-lo e usar *shading* no espaço *voxel* que dá resultados de boa qualidade sobre o volume filtrado ou não e deixar a segmentação para o final. Se o volume for binário (originalmente ou resultante de segmentação manual de volume não *hard*), deve-se dar preferência para o *shading* no espaço imagem que fornece qualidade razoável ou transformá-lo em *hard* e usar *shading* no espaço *voxel*, que torna o processo mais caro devido à filtragem 3D, mas oferece qualidade melhor. O *shading* no espaço objeto deve ser evitado. A filtragem das normais é sempre um processamento opcional. Veja o resumo esquematizado na figura 4.37.

Para efeito de comparação, as esferas da figura 4.38, geradas por `vgauss3d`, mostram as diferenças obtidas nos *shadings*. Como a saída de `vgauss3d` é do tipo *float*, o volume criado teve que ser convertido para *byte* a fim de ser processado por `vzbuff`. Em

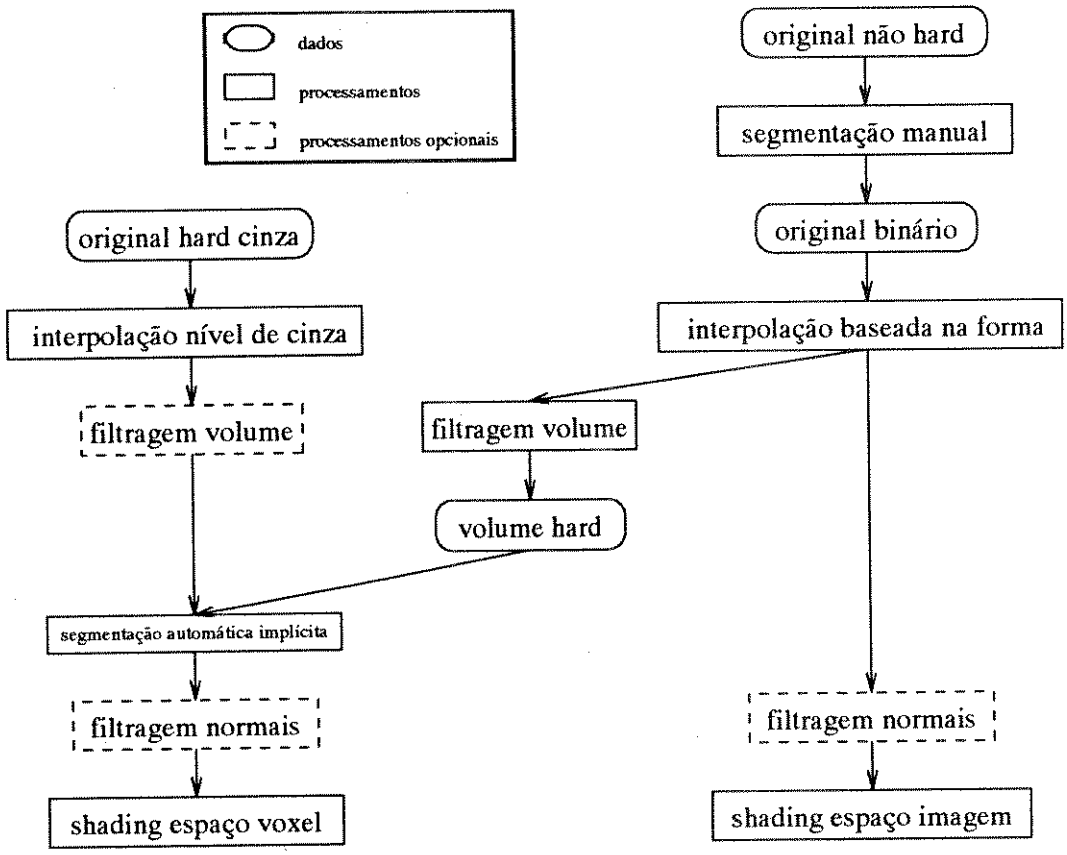


Figura 4.37: esquema resumido de pré-processamentos utilizados na prática

todas elas os vetores normais foram estimados no espaço *voxel*. Em 4.38a é mostrado o resultado estimando as normais no volume *byte*. Observe que a superfície se torna "irregular", devido à perda de informação ao transformar o volume de *float* para *byte*. Em 4.38b houve filtragem das normais usando núcleo de média na vizinhança 3x3, levando a um *shading* um pouco melhor, porém ainda com algumas irregularidades. Em 4.38c o volume *byte* foi filtrado para então ser usado na estimativa das normais, ocasionando um resultado melhor que o anterior, mesmo que alguns anéis circuncêntricos ainda possam ser vistos na parte frontal da esfera. Por último, em 4.38d tem-se o *shading* usando o volume *float* original para o cálculo das normais, que como esperado, oferece o melhor resultado.

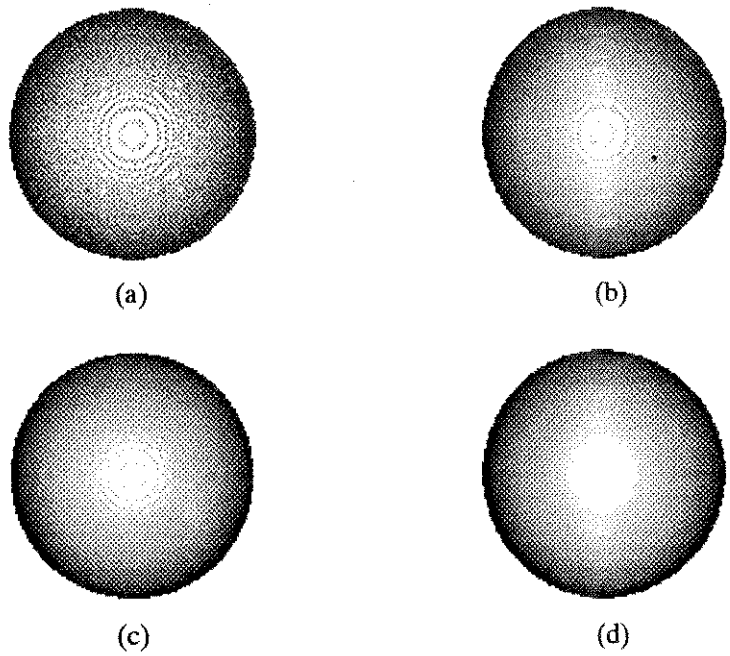


Figura 4.38: comparação entre alguns pré-processamentos que utilizam para estimativa das normais no espaço voxel: (a) volume transformado em byte; (b) volume byte com normais filtradas; (c) volume byte filtrado; (d) volume float original

4.10 Aplicações Interativas

Como colocado no início deste trabalho, a flexibilidade obtida com o uso de operadores combinados proporciona a pesquisa e o desenvolvimento de protótipos de apli-

cações mais complexas. O uso de *shell scripts* é um exemplo prático, já apresentado neste capítulo. Outros exemplos de aplicações, com um maior grau de interação, são apresentados a seguir. Trata-se de duas rotinas gráficas que fazem uso da biblioteca X11. A primeira delas, a rotina **xvplanes**, é uma simples adaptação da rotina **vplanexyz**. Tendo como entrada um volume de dados, a rotina mostra simultaneamente os três planos ortogonais entre si e paralelos aos eixos, que passam por um ponto, inicialmente o $(0,0,0)$, e um esboço do volume no canto inferior direito, com indicação da posição dos planos. A partir daí, correr o cursor com a ajuda do *mouse* por uma das três imagens equivale a navegar pelo plano correspondente, ao mesmo tempo que as coordenadas tridimensionais vão sendo mostradas, bem como o valor do *voxel* na posição corrente; pressionar o botão do *mouse* sobre uma das três imagens, corresponde a selecionar um novo ponto (x, y, z), o que conseqüentemente causa a mudança das duas outras imagens, de acordo com as novas coordenadas, e também a atualização da posição dos planos no esboço do volume. Linhas horizontais e verticais ao longo de toda dimensão de um plano marcam as posições de onde foram extraídos os outros dois planos. A figura 4.39 mostra a rotina **xvplanes** atuando sobre o volume da região orbital de um paciente. Embora não possa ser notado na figura, cada plano é identificado por uma cor diferente, sendo o plano x em vermelho, o plano y em verde e o plano z em azul. O *display* simultâneo de múltiplas fatias obtidas via alguma modalidade de aquisição de dados médicos é conhecida por *multiplanar display* (MPD) [26]. As rotinas **vplanexyz** e **xvplanes**, bem como a **vmontage** produzem este tipo de *display*. Comumente, procedimentos de MPD proporcionam as três vistas ortográficas descritas em **vplanexyz** e **xvplanes**, também chamadas de vistas sagital, axial e coronal. A vista sagital é aquela tomada ao longo do eixo x, revelando imagens paralelas ao plano y-z. A axial, também conhecida por transversa, é a vista ao longo do eixo z e representa imagens no plano x-y. A vista coronal é feita ao longo do eixo y e resulta em imagens paralelas ao plano x-z.

A outra rotina, a **xvmeasure** pode ser enquadrada como um exemplo de aplicação em análise de dados tridimensionais. Ela é capaz de medir distâncias e ângulos entre pontos de um espaço 3D, selecionados através do *mouse*. Tipicamente, as entradas são o *rendering* do volume, que representa o objeto no qual as medidas serão feitas e as coordenadas associadas à superfície do objeto, que é obtido durante o processo de *rendering* em uma das saídas de **vzbuff**. Note que os *renderings* da figura 4.22 (gerados pela *workspace* da figura 4.21) também servem de entrada para **xvmeasure**, como tantos outros *renderings*,

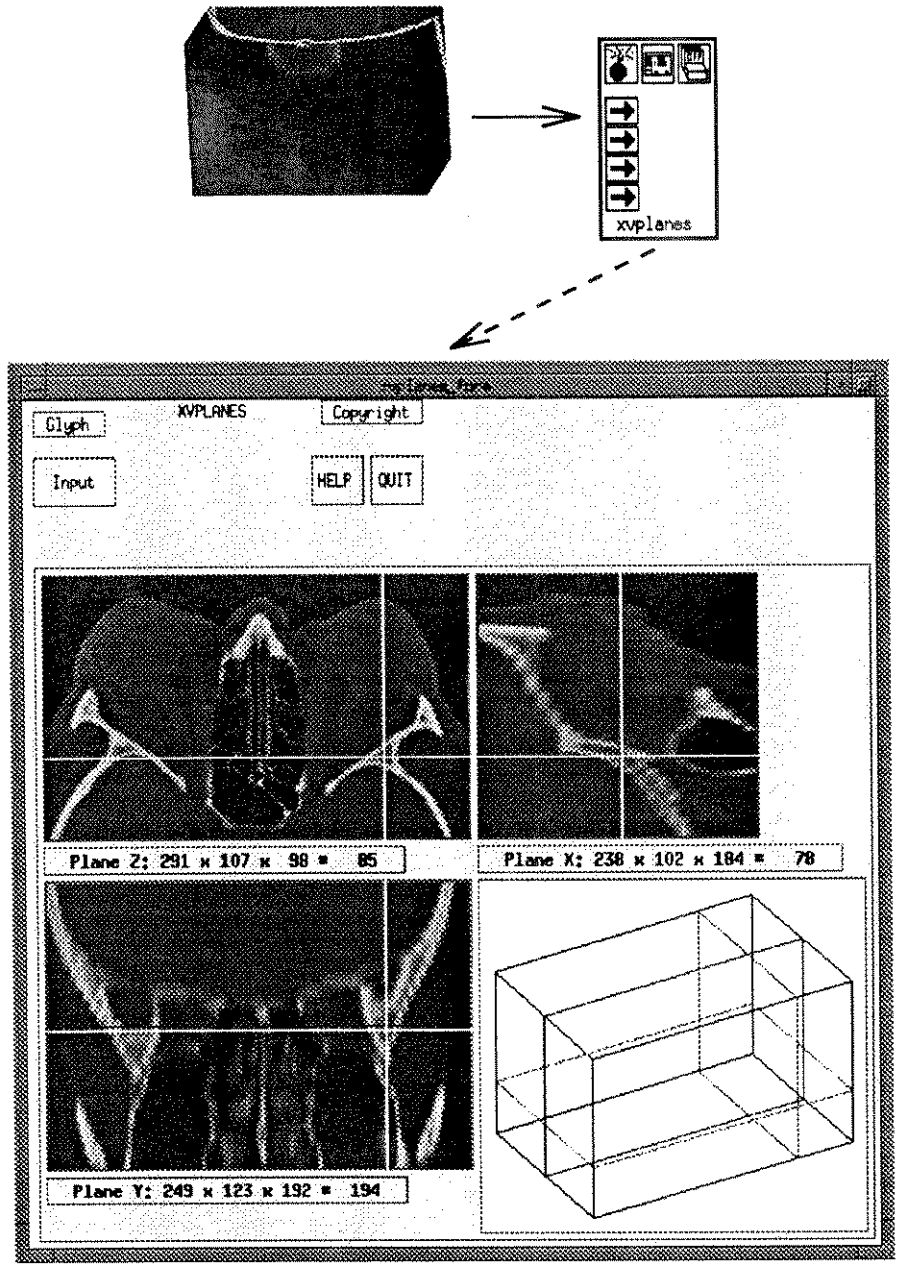


Figura 4.39: exemplo de aplicação: rotina xvplanes

desde que acompanhados dos respectivos arquivos de coordenadas. A aplicação funciona da seguinte maneira: é feito o *display* do objeto e com a ajuda do *mouse*, marca-se um ponto sobre a superfície do objeto. Embora a superfície do objeto esteja representada por uma imagem 2D, a coordenada 3D do ponto da superfície correspondente ao ponto marcado é conhecida através do arquivo de coordenadas. Os pontos marcados vão sendo acumulados e interligados por uma reta na imagem. As medidas de distância em linha reta entre os dois últimos pontos e a distância total, bem como o ângulo formado pelos três últimos pontos vão sendo mostrados a cada escolha de um novo ponto. As medidas são calculadas com base nas dimensões dos voxels do volume, que se encontram em campos pré-determinados no cabeçalho do arquivo de coordenadas. A figura 4.40 exemplifica a atuação desta aplicação sobre o crânio seco.

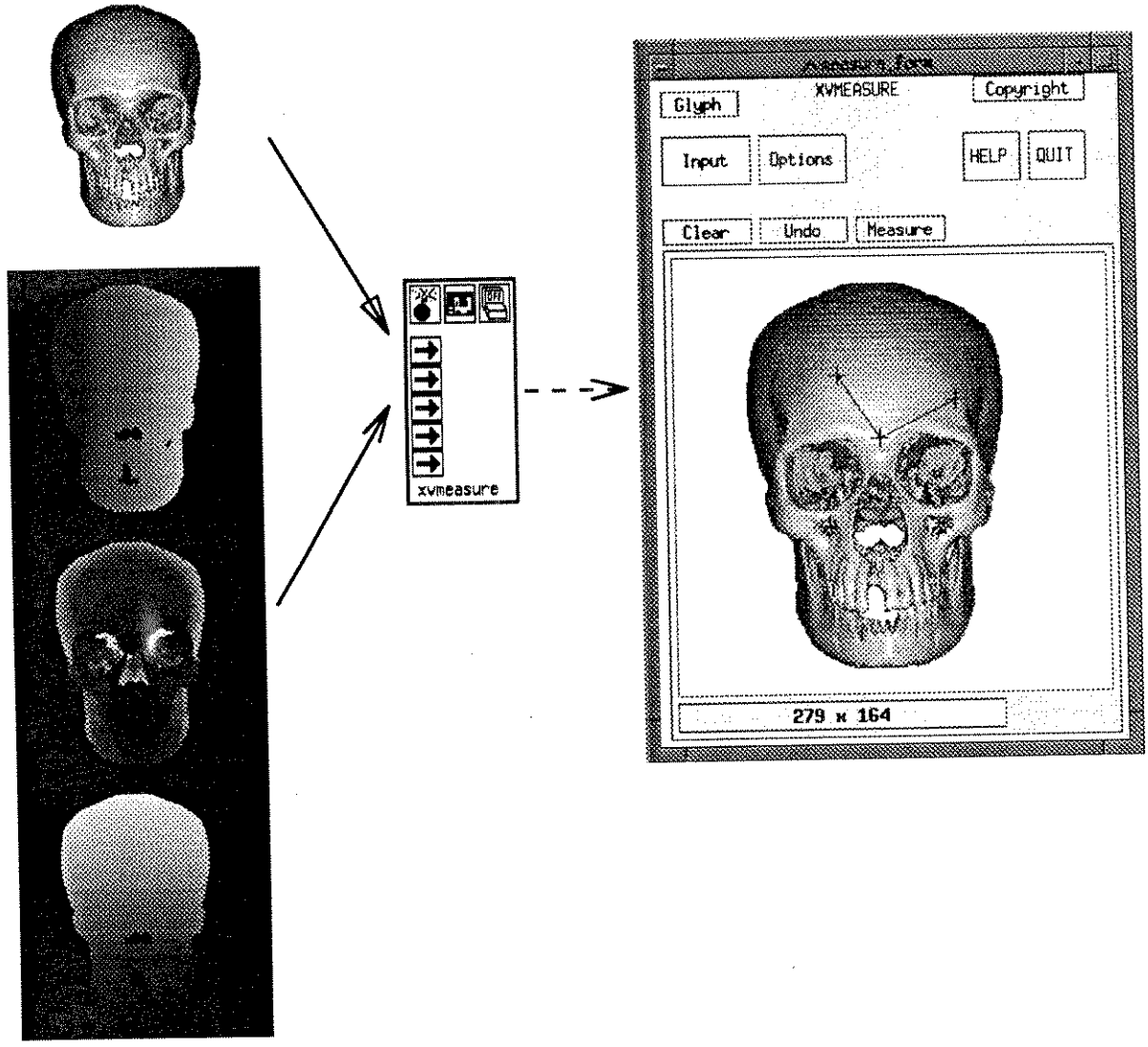


Figura 4.40: exemplo de aplicação: rotina xvmeasure

Capítulo 5

Conclusão

No capítulo anterior foram exploradas algumas das possibilidades de combinação dos operadores básicos introduzidos posteriormente. A definição dos operadores teve como referência a publicação de Udupa e Gonçalves [32], como também teve infuência da necessidade de novos operadores surgida na prática. O sistema Khoros foi escolhido como plataforma de desenvolvimento por várias razões, entre elas porque é um pacote aberto e de livre acesso, que oferece ferramentas de desenvolvimento e manutenção de *softwares*, além de uma linguagem de programação visual chamada **Cantata**, que possibilita a combinação dos operadores de forma fácil e rápida. Escritos em linguagem C, estes operadores foram agrupados sob a forma de uma *toolbox* para o sistema Khoros, chamada V3DTOOLS, e demonstraram serem bastante úteis na composição de procedimentos mais complexos para visualização, manipulação e análise de dados tridimensionais. As duas formas de composição apresentadas foram através da escrita de programas na linguagem visual do **Cantata**, que equivale à "construção" de *workspaces*, e através da escrita de programas na linguagem *shell* do Unix, ou seja, *shell scripts*. Nas *workspaces* os operadores são representados por ícones e o programa tem a forma de um grafo direcionado, no qual os nós são os operadores e os arcos entre os nós são associados aos dados que fluem entre as entradas e saídas dos operadores. Nas *shell scripts* os operadores são chamadas às rotinas correspondentes, e a comunicação entre eles se dá via canalização de dados (*pipe*) e escrita/leitura de dados em disco. Uma terceira forma possível de combinação, porém não utilizada aqui, é através da escrita de programas compiláveis, os quais fazem chamadas às funções de biblioteca dos operadores básicos. Durante a implementação de novos operadores, as funções de biblioteca

são automaticamente geradas e se tornam disponíveis quando utilizadas as ferramentas de desenvolvimento e manutenção de *softwares* do Khoros. A última opção tem a desvantagem de aumentar o código executável, o que não acontece nas *workspaces* e *shell scripts*.

Embora a eficiência no tempo de execução de *workspaces* e *shell scripts* seja inferior a muitos outros *softwares*, principalmente aqueles de uso comercial, os resultados conseguidos com a *Toolbox* V3DTOOLS podem ser considerados satisfatórios, dadas a facilidade e a flexibilidade com as quais novas combinações são feitas, proporcionando meios rápidos para a avaliação e análise de seus resultados. Exemplo disso são as diversas combinações apresentadas no capítulo 4, bem como os resultados conseguidos com estas combinações, que ilustram a filosofia da criação de processamentos de níveis mais elevados, contando somente com um número relativamente pequeno de operadores simples.

5.1 Sugestões

Embora o conjunto de rotinas implementadas na *toolbox* V3DTOOLS sejam bem abrangentes, existem ainda alguns tópicos que podem contribuir para complementar e ampliar a capacidade de processamento de algumas delas.

A rotina **vshad** poderia dispor de um processamento de *ray casting*, semelhante à rotina **vzbuff**, para permitir efeitos visuais de sombreamento em volumes contendo mais de um objeto e objetos côncavos. Obviamente esta modificação abre uma extensa faixa de possibilidades de novos *shadings* que são de maior utilidade para aplicações em computação gráfica, com o preço de elevar o custo computacional traduzido no aumento do tempo de processamento.

Como sugestões de novas rotinas, não existe na *toolbox* V3DTOOLS rotinas que apliquem transformações geométricas em dados 3D. Atualmente, transformações de translação e escala podem ser obtidas por combinação de outras rotinas. Por exemplo, a translação é conseguida pela extração do objeto de interesse com o uso da *workspace* da figura 4.19 e posicionando-o no local desejado com a rotina **vinsert3d**. A escala pode ser obtida pela rotina **vexpand3d** (interpolação de ordem zero com fatores diferentes para as três direções, porém maiores que 1) ou pela **vfft3d**, que também permite diferentes fatores de escala para cada dimensão, inclusive fatores menores que 1, com a restrição de que

estes fatores devem ser potência de 2. A escala usando `vfft3d` é implementada da seguinte maneira: uma vez gerado o espectro do volume, fatores de escala maiores que 1 são obtidos com a inserção do espectro no centro de um volume maior seguido da transformada inversa de Fourier; fatores de escala menores que 1 são obtidos pela extração do volume central do espectro seguido também da transformada inversa de Fourier. Em ambos os casos ocorre interpolação que se aproxima da cúbica. As restrições que impõem fatores de escala na potência de 2 vêm do fato do algoritmo usado na rotina `vfft3d` só operar com volumes de dimensões discretizadas desta forma. Quanto à rotação, apenas as rotações de 90 graus em qualquer direção podem ser feitas, através de sucessivas extrações de planos com a rotina `vplanexyz` seguidas de concatenações com a rotina `vaband`. Este último procedimento permite também a obtenção de fatias em orientações diferentes daquelas geradas pelo equipamento de aquisição de dados médicos. Esta técnica é conhecida por refatiamento ou *reslicing*. Com algumas alterações em `vzbuff`, a opção de estipular a distância do plano de projeção em relação ao objeto que, como mostrado na figura 4.6, causa efeitos de cortes, pode ser útil para a técnica de refatiamento, permitindo que novas fatias sejam obtidas em posições bastante diversificadas. O plano de projeção, neste caso, não deve lançar raios em direção ao objeto, mas apenas computar os *voxels* que estejam nas coordenadas do plano. Com as rotinas implementadas até o momento é possível obter este resultado através de *workspaces*.

Os três exemplos de pré-processamento do capítulo anterior (figuras 4.29, 4.32 e 4.35) utilizaram filtros por convolução 3D. Uma outra possibilidade seria implementar filtros usando a rotina `vfft3d`. A figura 2.2, que exemplifica a criação de procedimentos através de agrupamentos hierárquicos de *workspaces*, mostra tal filtro. Nele, a esfera gerada pela combinação de `vgauss3d` e `vthresh3d` serve para isolar a parte central do espectro (filtro passa baixa), que depois é usado para reconstruir o volume usando a transformada inversa de Fourier. O uso deste tipo de filtro só é viável quando se dispõe de muita memória, pois o volume de dados necessário para o espectro é muito grande. Por exemplo, um volume de dimensões 256x256x256, onde cada voxel é representado por um *byte*, totalizando 16Mb, geraria um espectro ocupando 128Mb, correspondente a um volume de mesmas dimensões onde cada voxel é representado por dois números reais, um para a parte real e outro para a parte imaginária, somando 8 *bytes*, portanto.

A adição de cor é também um ponto importante e deve ser levado em conside-

ração. Atualmente nenhuma das rotinas implementadas na *toolbox* V3DTOOLS é capaz de lidar com imagens coloridas. A adição de cor é fundamental em alguns casos, facilitando a distinção visual dos dados de uma forma mais contrastante que em níveis de cinza, o que é de grande utilidade para a fase de análise.

Finalizando, somente duas *shell scripts* foram apresentadas no capítulo anterior (`vrender` e `vrender2`), embora muitas das *workspaces* possam sofrer o mesmo processo e serem condensadas também, como as *workspaces* de pré-processamento há pouco citadas (figuras 4.29, 4.32 e 4.35), a *workspace* que faz o *display* do volume em forma de fatias empilhadas (figura 4.25b), a que representa os dados inseridos dentro do volume (figura 4.23), a que faz o *preview* nas seis vistas principais (figura 4.5), etc..

Apêndice A

Forms das Rotinas Para Entrada de Parâmetros

A.1 Rotinas de Transformações Geométricas

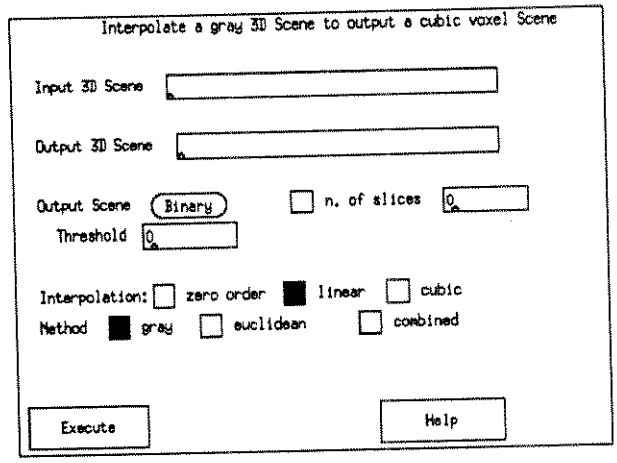


Figura A.1: form da rotina vinter

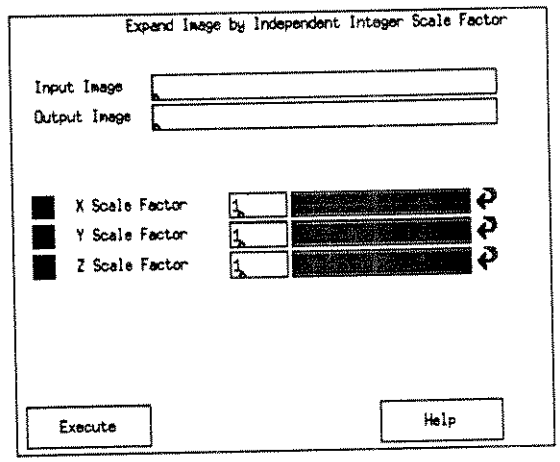


Figura A.2: form da rotina vexpand3d

A.2 Rotinas de Visualização

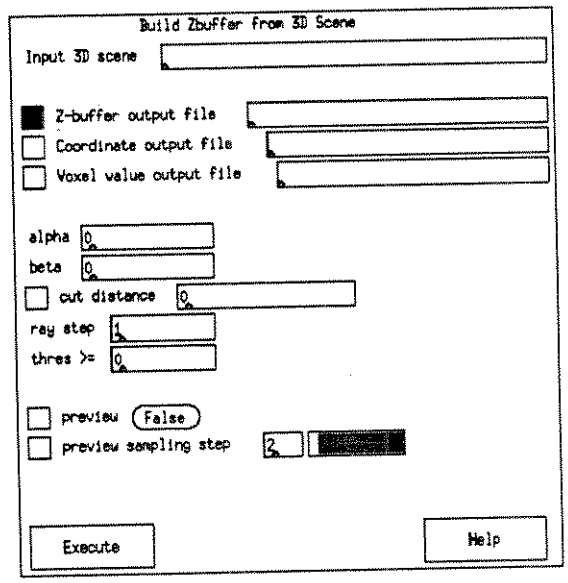


Figura A.3: form da rotina vzbuff

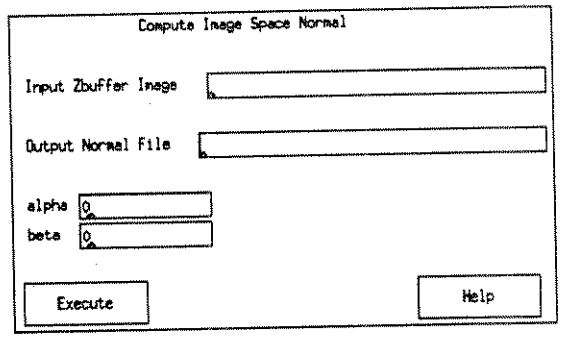


Figura A.4: form da rotina visnorm

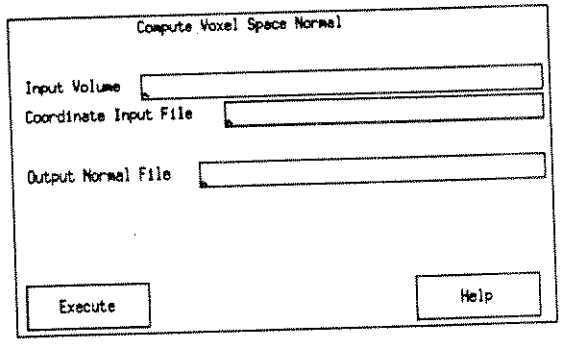


Figura A.5: form da rotina vvsnorm

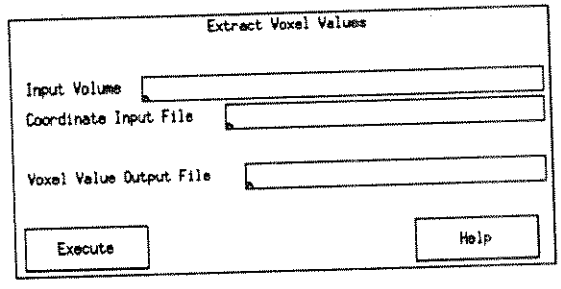


Figura A.6: form da rotina vvoxext

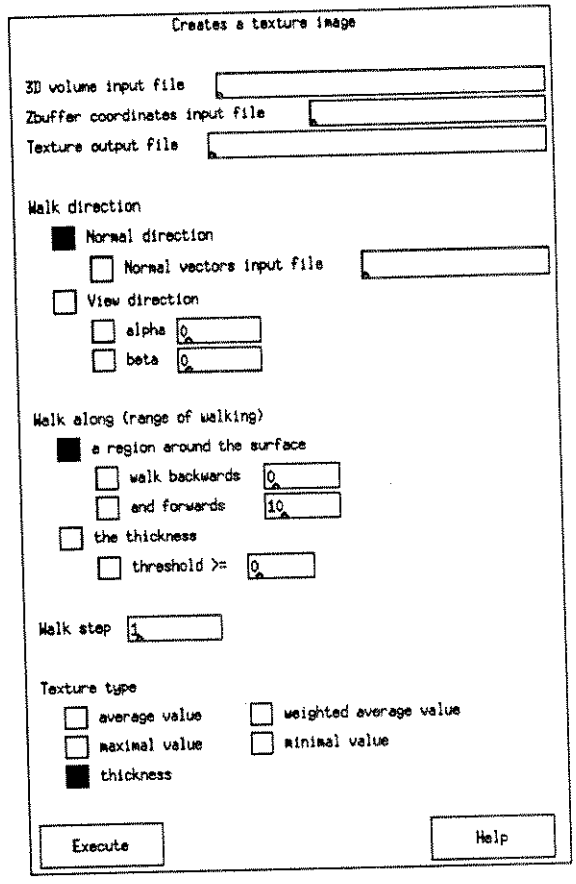


Figura A.7: form da rotina vtextu

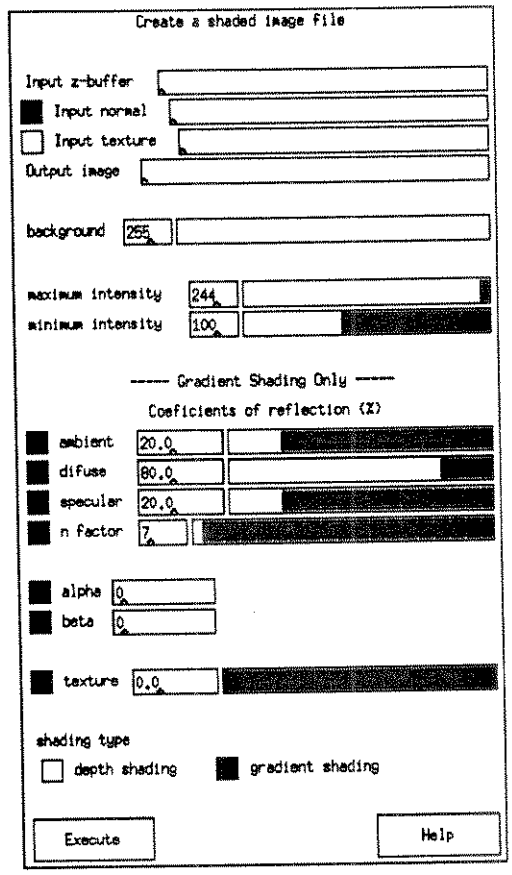


Figura A.8: form da rotina vshad

A.3 Rotinas de Processamento Puntual do Voxel

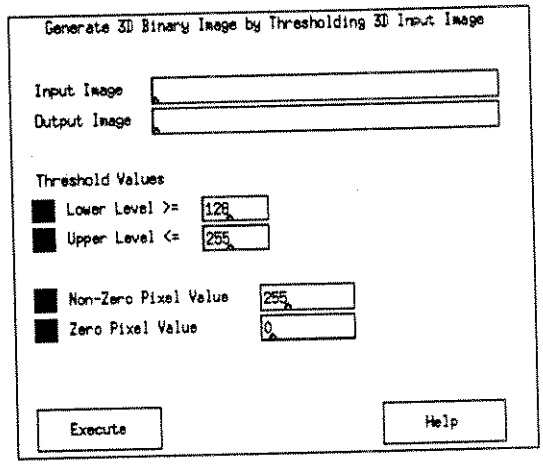


Figura A.9: form da rotina vthresh3d

A.4 Rotinas de Processamento Espacial

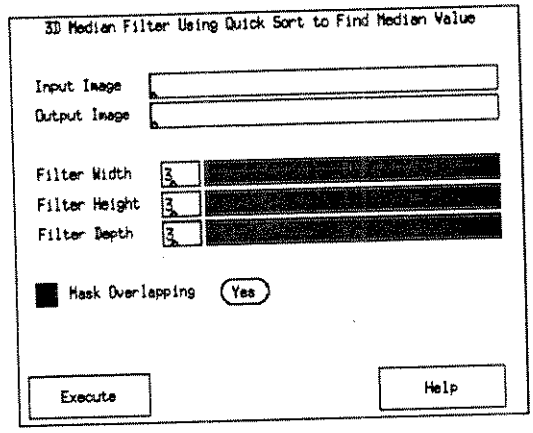


Figura A.10: form da rotina vqmed3d

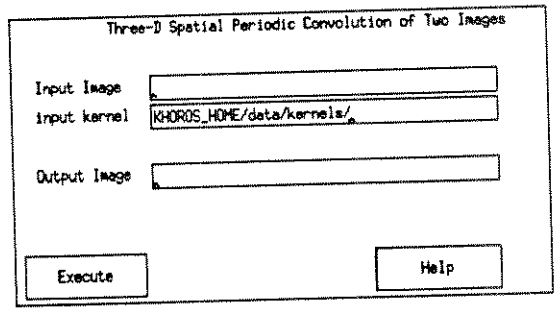


Figura A.11: form da rotina vconv3d

A.5 Rotinas de Transformada Digital de Fourier

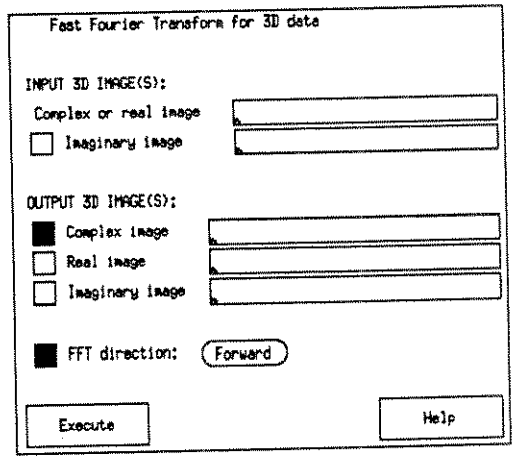


Figura A.12: form da rotina vfft3d

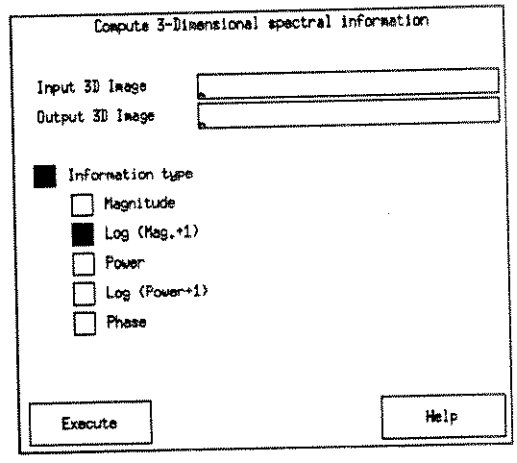


Figura A.13: form da rotina vmpp3d

A.6 Rotinas de Manipulação de Regiões

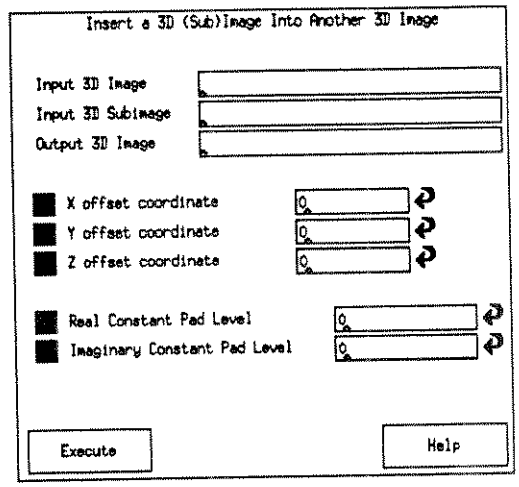


Figura A.14: form da rotina vinsert3d

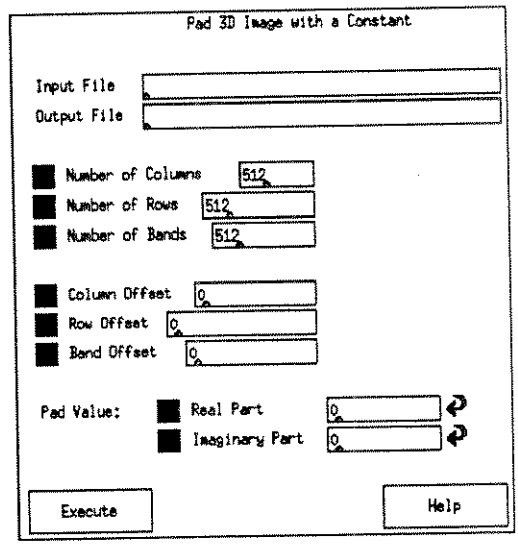


Figura A.15: form da rotina vpad3d

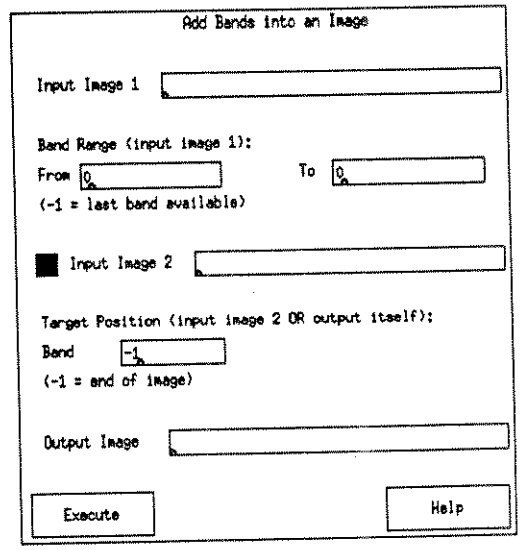


Figura A.16: form da rotina vaband

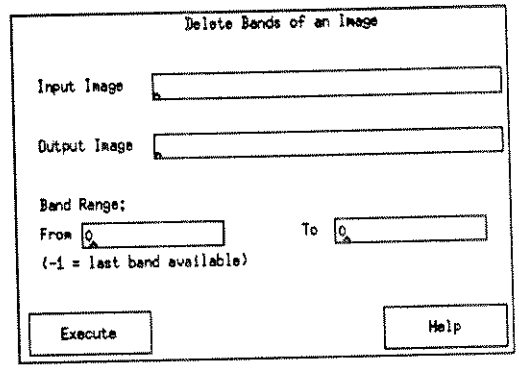


Figura A.17: form da rotina vdband

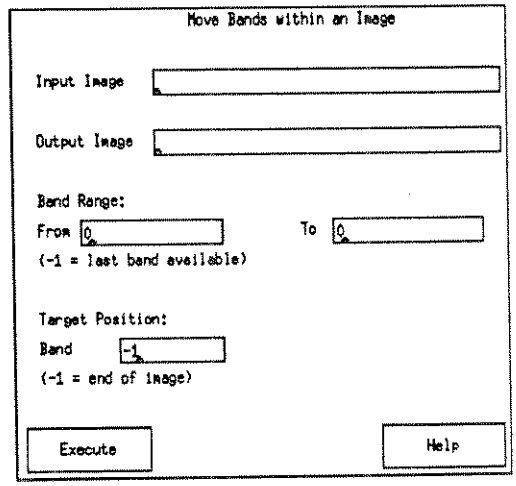


Figura A.18: form da rotina vmband

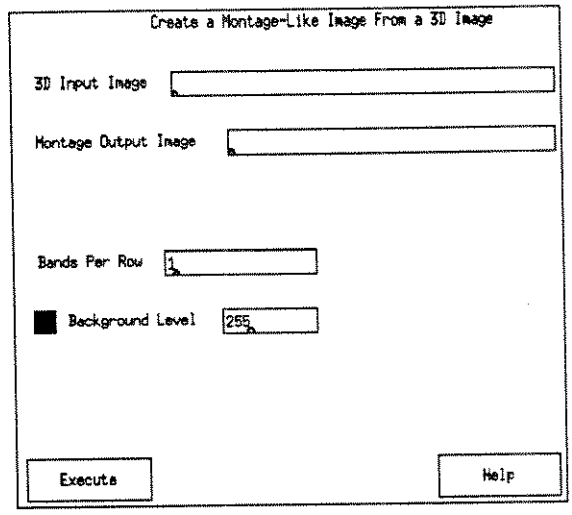


Figura A.19: form da rotina vmontage

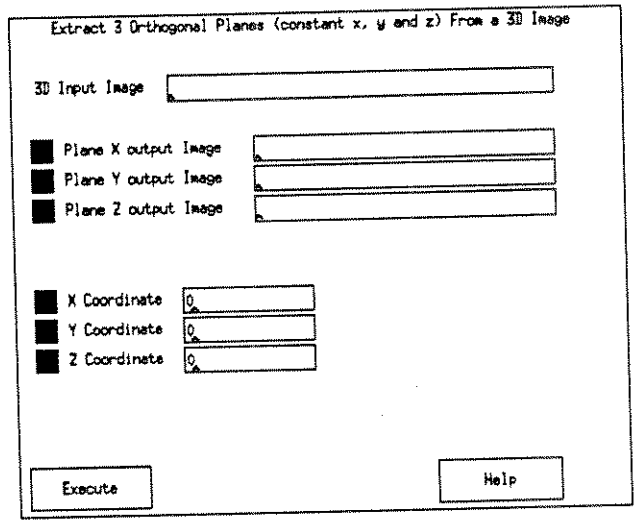


Figura A.20: form da rotina vplanexyz

A.7 Rotinas de Geração de Imagens Sintéticas

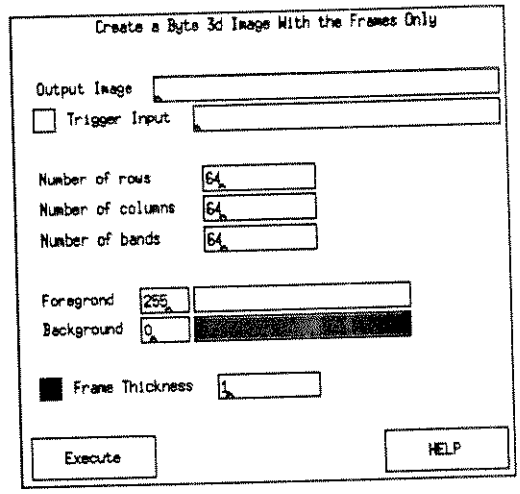


Figura A.21: form da rotina vframe

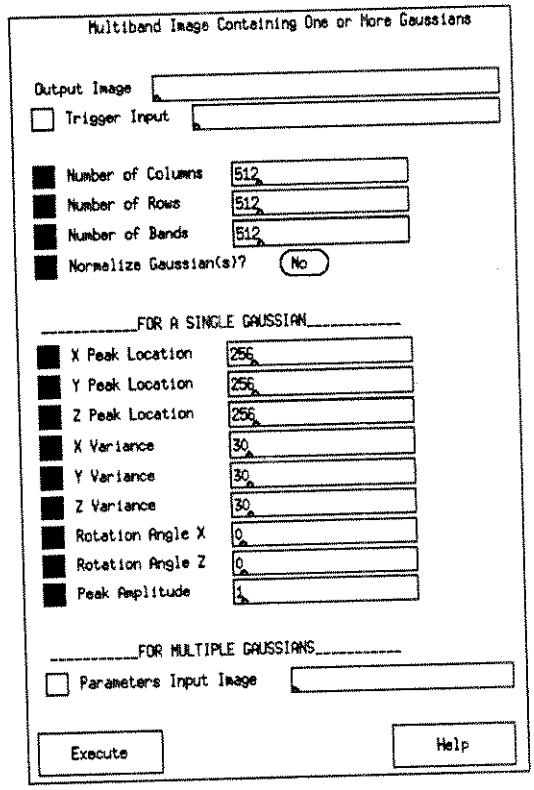


Figura A.22: form da rotina vgauss3d

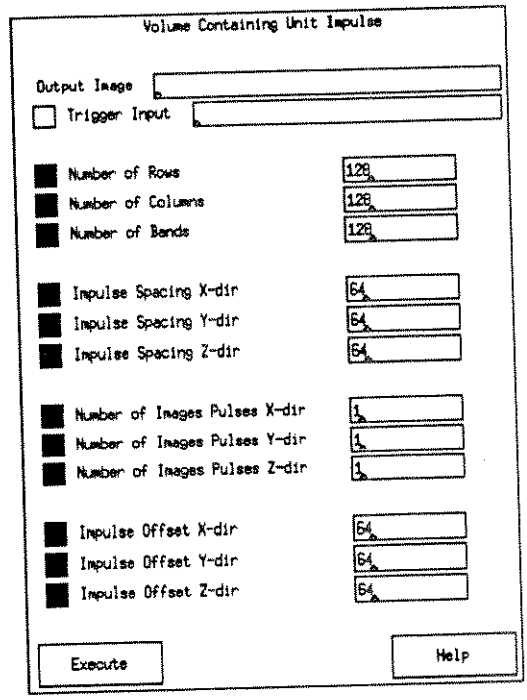


Figura A.23: form da rotina vimpul3d

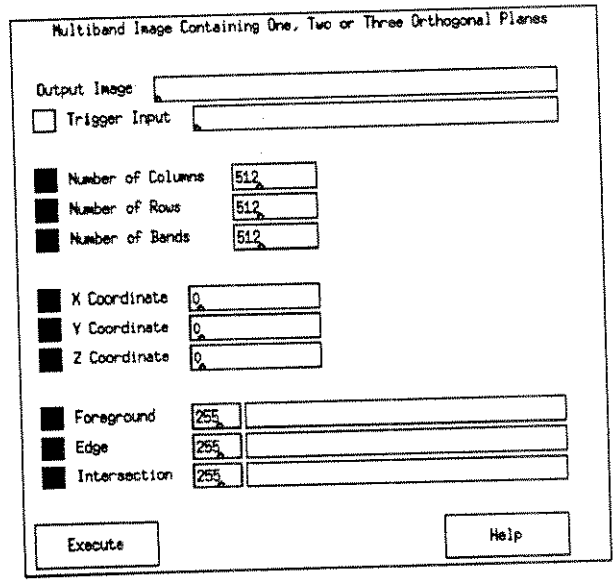


Figura A.24: form da rotina vorthog

A.8 Shell Scripts

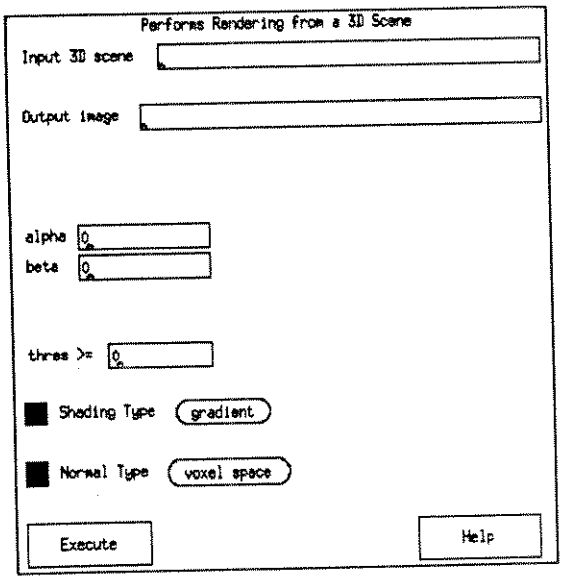


Figura A.25: form da shell script vrender

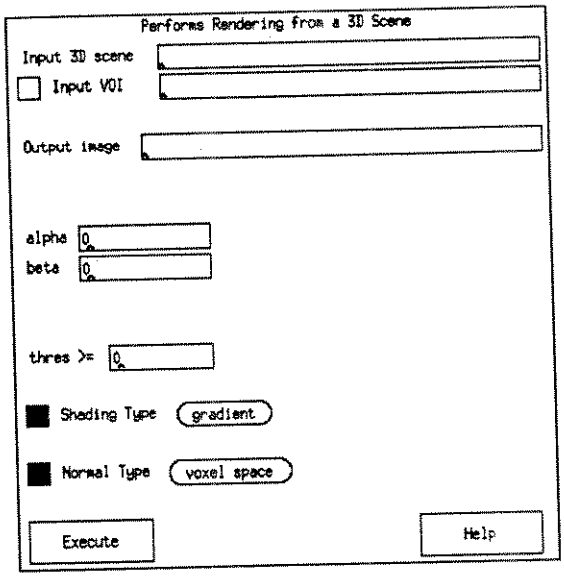


Figura A.26: form da shell script vrender2

BIBLIOGRAFIA

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

[2] R. Brown and C. Upson. personal communication.

[3] D.S. Dyer. A dataflow toolkit for visualization. *IEEE Computer Graphics and Applications*, 10(4):60–69, July 1990.

[4] A.X. Falcão. Visualização de volumes aplicada à área médica. Master's thesis, Universidade Estadual de Campinas - UNICAMP, FEE, Caixa Postal 6101, Campinas, SP - Brazil, 1993. in portuguese.

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

[6] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, October 1977.

[7] D. Geist and M.W. Vannier. Pc-based 3d reconstruction of medical images. *Computer & Graphics*, 13(2):135–143, September 1989.

[8] R.C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison Wesley Publishing Company, Reading, Mass., 2nd edition, November 1987.

[9] D. Gordon and R.A. Reynolds. Image-space shading of three-dimensional objects. *Computer Vision, Graphics and Image Processing*, 29:361–376, 1985.

[10] P.B. Heffernan and R.A. Robb. A new method for shaded surface display of biological and medical images. *IEEE Transactions on Medical Images*, MI-4:26–38, 1985.

[11] G.T. Herman. 3d display: A survey from theory to applications. *Computerized Medical Imaging and Graphics*, 17(4/5):231–242, July–October 1993.

[12] K.H. Hohne and R. Bernstein. Shading 3d images from ct using gray-level gradients. *IEEE Transactions on Medical Images*, MI-5(1):45–47, March 1986.

[13] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM J. Research and Development*, 19:2–11, January 1975.

[14] The Khoros Group, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131. *Khoros Manual, Volumes 1 and 2.* Release 1.0.

[15] R.A. Lotufo, G.T. Herman, and J.K. Udupa. Combining gray-level into shape-based interpolation. In *Visualization in Biomedical Computing 1992 (VCB'92)*, pages 13–16, Chapel Hill, North Carolina, USA, October 1992.

[16] B. Lucas et al. An architecture for a scientific visualization system. In *Proceedings of Visualization*, October 1992.

[17] D. Meagher. Geometric modelling using octree encoding. *Computer Graphic and Image Processing*, 19:129–147, 1982.

[18] D. Meagher. *The Octree Encoding Method for Efficient Solid Modelling.* PhD thesis, Rensselaer Polythecnic Institute, August 1982.

[19] J.R. Rasure, D. Argiro, T. Sauer, and C.S. Williams. Visual language and software development environment for image processing. *International Journal of Imaging Systems and Tecnology*, 2:183–199, August 1990.

[20] J.R. Rasure and C.S. Williams. An integrated data flow visual language and software development environment. *Journal of Visual Languages and Computing*, 2:217–246, April 1991.

[21] J.R. Rasure and M. Young. An open environment for image processing and software development. In *1992 SPIE/IS&T Symposium on Eletronic Imaging Proceedings*, volume 1659, February 1992.

[22] S. Raya. Softvu - a software package for multidimensional medical image analysis. In *Proceedings of SPIE*, volume 1232, pages 162–166, 1990.

[23] R.A. Reynolds. Some architectures for real-time display of three-dimentional objects: A comparative survey. Technical Report MIPG84, Dept. of Radiology, Univ. of Pennsylvania, Philadelphia, October 1983.

[24] R.A. Robb and D.P. Hanson. Analyze: A software system for biomedical image analysis. In *First Conference on Visualization in Biomedical Computing*, pages 507–518, Atlanta, 1990. GA: IEEE.

[25] D.F. Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[26] M.R. Stytz, G. Frieder, and O. Frieder. Three-dimensional medical imaging: Algorithms and computer systems. *ACM Computing Surveys*, 23(4):421–499, December 1991.

[27] U. Tiede, K.H. Hoehne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Surface rendering - investigation of medical 3d-rendering algorithms. *IEEE Computer Graphics and Applications*, pages 41–53, March 1990.

[28] H.K. Tuy and L.T. Tuy. Direct 2-d display of 3-d objects. *IEEE Computer Graphics and Applications*, 4(10):29–33, October 1984.

[29] J.K. Udupa. Interactive segmentation and boundary surface formation for 3d digital images. *Computer Graphic and Image Processing*, 18:213–235, 1982.

[30] J.K. Udupa. Display and analysis of 3d medical images using directed contours. In *Proceedings of the 6th Annual Conference and Exposition of the National Computer Graphics Association*, pages 145–155, Dallas, Tx., May 14–18 1985. NCGA '85.

[31] J.K. Udupa. Computer aspects of 3d imaging in medicine: A tutorial. Technical Report MIPG 153, Medical Image Processing Group, Dept. of Radiology, Univ. of Pennsylvania, Philadelphia, 1989.

[32] J.K. Udupa and R.J. Gonçalves. Imaging transforms for visualizing surfaces and volumes. *Journal of Digital Imaging*, 6(4):213–236, November 1993.

[33] J.K. Udupa, R.J. Gonçalves, K. Iyer, S. Narendula, D. Odhner, S. Samarasekera, and S. Sharma. 3d imaging systems. Technical Report MIPG 195, Medical Image Processing Group, Dept. of Radiology, Univ. of Pennsylvania, Philadelphia - PA - USA, June 1993.

[34] J.K. Udupa, R.J. Gonçalves, K. Iyer, S. Narendula, D. Odhner, S. Samarasekera, and S. Sharma. 3dviewnix: An open, transportable software system for the visualization

and analysis of multidimensional, multimodality, multiparametric images. In *Proceedings of SPIE: Image Capture, Formatting, and Display*, volume 1897, pages 47–58, February 1993.

[35] J.K. Udupa, D. Odhner, S. Samarasekera, R.J. Gonçalves, K. Iyer, S. Sharma, K.P. Venugopal, S. Furuie, and A.X. Falcão. *The 3DVIEWNIX Software System: User's Manual.* Medical Image Processing Group, Dept. of Radiology, Univ. of Pennsylvania, Philadelphia - PA - USA, October 1993.

[36] C. Upson et al. The aplication visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[37] M.W. Vannier, J.L. Marsh, and M.H. Gado. Three-dimentional display of intracranial soft-tissue abnormalities. *American Journal of Neuroradiology*, 4:520–521, 1983.

[38] M.W. Vannier, J.L. Marsh, and J.O. Warren. Three-dimentional computer graphics for craniofacial surgical planning and evaluation. *Computer & Graphics*, 13:263–274, 1983.

[39] C. Williams, J. Rasure, and C. Hansen. The state of the art of visual languages for visualization. *IEEE*, pages 202–209, 1992.