UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CARLOS AUGUSTO DIETRICH

Visualização Interativa e Manipulação de Imagens 3D em Tempo Real usando Métodos Baseados na GPU

Dissertação apresentada como requisito
parcial para obtenção do grau de
Mestre em Ciência da Computação

Prof[a]. Dr[a]. Luciana Porcher Nedel
Orientadora

Dr[a]. Sílvia Delgado Olabarriaga
Co-orientadora

Porto Alegre, agosto de 2004

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Dietrich, Carlos Augusto

Visualização Interativa e Manipulação de Imagens 3D em Tempo Real usando Métodos Baseados na GPU / Carlos Augusto Dietrich. – Porto Alegre: Programa de Pós-Graduação em Computação, 2004.

71 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2004. Orientadora: Luciana Porcher Nedel; Co-orientadora: Sílvia Delgado Olabarriaga.

1. Visualização volumétrica. 2. Interação. Hardware gráfico. I. Nedel, Luciana Porcher. II. Olabarriaga, Sílvia Delgado. III. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

2D	bi-dimensional.
3D	tri-dimensional.
BSP-trees	*Binary Space Partition trees*.
PC	Computador Pessoal (*Personal Computer*).
CPU	*Central Processing Unit*[1] (Unidade de Processamento Central).
DICOM	*Digital Imaging and Communications in Medicine*.
GPU	*Graphics Processor Unit* (Unidade de Processamento Gráfico).
HU	*Hounsfield Units*.
kD-trees	*k-Dimensional trees*.
MVD	Método de Visualização Direta.
OpenGL	*Open Graphics Library*.
RGB	*Red*, *Green*, *Blue* (canais de cor vermelho, verde e azul).
RM	Ressonância Magnética (*Magnetic Resonance*).
TC	Tomografia Computadorizada (*Computed Tomography*).
VGA	*Video Graphics Array*.
VOI	Volume de Interesse (*Volume of Interest*).
WSG	*Weighted Sweep Graph*.

[1] Algumas siglas foram mantidas na sua linguagem original para facilitar a compreensão do leitor.

LISTA DE SÍMBOLOS

$c[i]$	cor (tipicamente uma grandeza vetorial) amostrada na posição i do espaço.
$\alpha[i]$	opacidade (grandeza escalar) amostrada na posição i do espaço.
P	polígono/poliedro definido por uma lista ordenada de pontos (p_1, p_2, p_3,..., p_n).
p_i	ponto i de um polígono ou poliedro.
G	grafo planar.
v_i	vértice i.
e_i	aresta i.
γ	linha de varredura.
$\vec{s}$	vetor ortogonal à linha de varredura.
v_i	vértice v_i projetado sobre o vetor $\vec{s}$.
o_s	vértice v_i projetado mais à esquerda (em relação a $\vec{s}$).
$W(v_i)$	peso de um vértice projetado (distância Euclidiana entre v_i e o_s).
$\Gamma(P, \vec{s})$	WSG definida a partir de um polígono/poliedro P na direção $\vec{s}$.
$C(\Gamma, d)$	corte em Γ na distância d.
$F(v_i)$	lista de vértices adjacentes a v_i em G.
PR	ponto de aplicação da ferramenta indicado com o mouse sobre a tela.
R	raio de ação para o cursor da ferramenta.
F_t	distância do fragmento no momento t.
d	distância (em coordenadas de tela) do fragmento à PR.
$\min(a, b)$	mínimo entre a e b.
SC	coordenadas de tela do fragmento.
TP	coordenadas de objeto do fragmento.
TC	coordenadas de textura do fragmento.
TP_{PR}	posição no espaço de objeto do fragmento correspondente a PR.
Ro_i	raio de ação i da ferramenta no espaço de objeto.
$\vec{v}_{cop}$	vetor normalizado coplanar ao plano de projeção.
$\vec{v}_{TP_{PR}-TC}$	vetor correspondente ao vetor formado entre os pontos TC e TP_{PR}, coplanar ao plano de projeção.
TC'	coordenada de textura obtida a partir da soma entre TP_{PR} e $\vec{v}_{TP_{PR}-TC}$.

LISTA DE FIGURAS

RESUMO

Este trabalho apresenta um conjunto de ferramentas que exploram as capacidades recentes das placas gráficas de computadores pessoais para prover a visualização e a interação com volumes de dados. O objetivo é oferecer ao usuário ferramentas que permitam a remoção interativa de partes não relevantes do volume. Assim, o usuário é capaz de selecionar um volume de interesse, o que pode tanto facilitar a compreensão da sua estrutura quanto a sua relação com os volumes circundantes. A técnica de visualização direta de volumes através do mapeamento de texturas é explorada para desenvolver estas ferramentas. O controle programável dos cálculos realizados pelo hardware gráfico para gerar a aparência de cada pixel na tela é usado para resolver a visibilidade de cada ponto do volume em tempo real. As ferramentas propostas permitem a modificação da visibilidade de cada ponto dentro do hardware gráfico, estendendo o benefício da visualização acelerada por hardware. Três ferramentas de interação são propostas: uma ferramenta de recorte planar que permite a seleção de um volume de interesse convexo; uma ferramenta do tipo "borracha", para eliminar partes não relevantes da imagem; e uma ferramenta do tipo "escavadeira", para remover camadas do volume. Estas ferramentas exploram partes distintas do fluxo de visualização por texturas, onde é possível tomar a decisão sobre a visibilidade de cada ponto do volume. Cada ferramenta vem para resolver uma deficiência da ferramenta anterior. Com o recorte planar, o usuário aproxima grosseiramente o volume de interesse; com a borracha, ele refina o volume selecionado que, finalmente, é terminado com a escavadeira. Para aplicar as ferramentas propostas ao volume visualizado, são usadas técnicas de interação conhecidas, comuns nos sistemas de visualização 2D. Isto permite minimizar os esforços do usuário no treinamento do uso das ferramentas. Finalmente, são ilustradas as aplicações potenciais das ferramentas propostas para o estudo da anatomia do fígado humano. Nestas aplicações foi possível identificar algumas necessidades do usuário na visualização interativa de conjuntos de dados médicos. A partir destas observações, são propostas também novas ferramentas de interação, baseadas em modificações nas ferramentas propostas.

Palavras-chave: Visualização volumétrica baseada em texturas; recorte volumétrico interativo; visualização volumétrica em tempo real; imagens médicas.

Real-time Interactive Visualization and Manipulation of 3D Images using GPU-based Methods

ABSTRACT

This work presents a set of tools that provide 3D visualization and interaction with large volumetric data relying on recent programmable capabilities of consumer-level graphics cards. The objective is providing the user with interactive volume sculpting tools. With those tools the user is able to select a volume of interest, which helps both in the its understanding as with your relationship with the whole volume. We exploit the texture-based volume rendering to develop these tools. The programmable control of calculations performed by the graphics hardware for generating the appearance of each pixel on the screen is exploited to resolve the visibility of each point of the volume in real-time. The tools proposed here allows real-time modification of visibility parameters inside the graphics hardware, extending the benefit of hardware acceleration beyond display, namely for computation of voxel visibility. Three interactive tools are proposed: a plane-based cutting tool that allows the selection of a convex volume of interest, an eraser-like tool to eliminate non-relevant parts of the image and a digger-like tool that allows the user to eliminate layers of a 3D image. These tools exploit stages of texture-based volumetric visualization pipeline where is possible taking the decision about volume visibility. Each tool comes to add some new functionality to the previous tool. With the plane-based cut, the user only approximates the volume of interest; with the eraser-like tool, he refines his selection, which is finished with the digger-like tool. Known user interaction techniques, as adopted in 2D painting systems, are used to apply the proposed tools on a volume. The strategy is to minimize the user training efforts to learn how to operate the tools. Finally, the potential applications of the conceived tools are illustrated for liver anatomy study. In these applications, it was possible to identify some needs of the user in medical datasets visualizations. We also propose some new interactive tools that extend the functionality of previous tools based on these observations.

Keywords: Texture-based volume visualization; interactive volume clipping; real-time volume rendering; medical imaging.

1 INTRODUÇÃO

1.1 Visualização de Imagens Médicas Volumétricas

Técnicas de visualização tridimensional estão provando ser úteis em uma grande quantidade de aplicações no campo da medicina, incluindo diagnóstico, planejamento de tratamento, simulação cirúrgica, treinamento médico, condução intra-operativa e tele-cirurgia (TOMBROPOULOS, 1999). Séries de imagens bidimensionais como tomografias computadorizadas (TC) e ressonâncias magnéticas (RM) são as fontes de dados específicos de cada paciente para estes tipos de aplicações. Ambos TC e RM geram uma série de imagens bidimensionais que representam secções transversais (fatias) através da anatomia do paciente. Quando empilhadas umas sobre as outras, estas imagens representam a estrutura tridimensional (imagem volumétrica) adquirida. Elas assim provêem aos médicos uma forma não-invasiva de visualizar a anatomia interna do paciente, localizar tecidos dentro do corpo e visualizar algumas propriedades fisiológicas destes tecidos. Para as aplicações fazerem uso total da informação contida dentro destes dados médicos, são necessárias técnicas de visualização e interação com imagens volumétricas, preferencialmente em tempo real.

Em geral, as abordagens para a visualização de volumes são divididas entre métodos diretos e indiretos. Métodos que não extraem explicitamente estruturas geométricas do volume são chamados de diretos. Em métodos de visualização direta (MVDs), o volume é visualizado com base numa segmentação fraca realizada por funções de transferência, que mapeiam os valores registrados nas partículas do volume para valores de cor e opacidade (ZHOU, 2003). MVDs vêm sendo propostos desde o início da década de 80, quando foram apresentados os primeiros modelos óticos para a visualização (*rendering*) por computador (BLINN, 1982; KAJIYA; VON HERZEN, 1984).

Técnicas tradicionais de MVDs contam com a eficiência da CPU (*Central Processing Unit*), como a técnica *shear-warping* proposta por Lacroute e Levoy (1994). Com a disponibilidade do hardware de texturização em computadores pessoais (PC), alguns autores passaram a explorar as capacidades do mapeamento de texturas 2D e 3D para implementar MVDs (CABRAL; CAM; FORAN, 1994; CULLIP; NEUMANN, 1993). Hoje, o avanço do hardware gráfico permite a visualização direta baseada em texturas com imagens de qualidade satisfatória e melhor desempenho do que abordagens baseadas na CPU (REZK-SALAMA *et al.*, 2000). Estas características fazem da técnica de mapeamento de texturas, hoje uma das principais linhas de pesquisa em visualização direta.

A despeito da evolução das técnicas de visualização, em muitos casos a análise de imagens volumétricas ainda é feita baseada em imagens 2D. Atualmente, a visualização de tomografias no meio clínico é feita fatia a fatia. Embora muitos radiologistas e médicos treinados sejam capazes de elaborar diagnósticos a partir deste tipo de visualização, há um grande interesse em visualizar estes dados em três dimensões. Este

interesse é ainda maior quando se trata de órgãos e tecidos que possuem uma anatomia complexa, como o caso do fígado humano e o seu sistema vascular.

A visualização 3D de imagens médicas é um tópico de pesquisa muito explorado atualmente. Um dos principais focos destas pesquisas é a interação com os dados médicos volumétricos. Assim como nestas pesquisas, técnicas interativas de inspeção de dados médicos são os motivadores deste trabalho. Em especial, a visualização do fígado em TCs abdominais, como pode ser visto no exemplo da Figura 2.1.

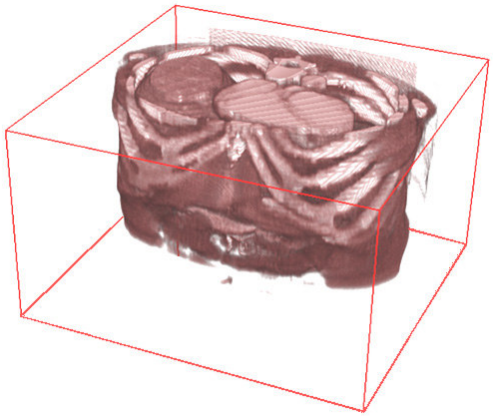
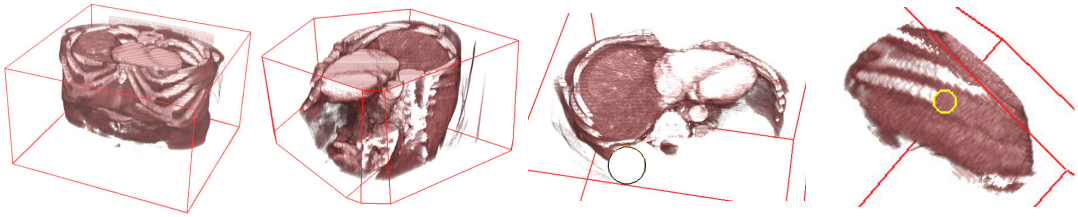


Figura 2.1: Visualização 3D com MVD de um TC abdominal.

A oclusão do objeto de interesse é um problema comum na visualização 3D de dados médicos. No caso da Figura 2.1, por exemplo, a visualização do fígado é obstruída pelo coração e algumas costelas. Alguns pesquisadores (AYLWARD; BULLITT, 2002) sugerem a segmentação da imagem volumétrica para resolver este problema. A partir da imagem segmentada, o objeto de interesse pode ser isolado e visualizado separadamente. A segmentação também permite a extração de medidas do objeto de interesse (tamanho, volume, etc), muito importante na análise de dados médicos. No entanto, a pesquisa de métodos de segmentação de imagens médicas diverge em uma série de técnicas muito distintas. Freqüentemente, o sucesso destas técnicas depende da sua adaptação aos métodos de aquisição de imagens e aos objetos de interesse. Este tipo de abordagem restringe as possíveis aplicações genéricas da técnica de visualização.

Nesta dissertação são propostas ferramentas de interação com volumes visualizados diretamente que permitem isolar e visualizar o objeto separadamente. De certa forma, também ocorre segmentação, porém implicitamente entre duas partes do processo de visualização volumétrica interativa: primeiro, na função de transferência definida na aquisição dos dados (que não é abordada neste trabalho); segundo, na aplicação interativa de ferramentas de recorte volumétrico. Por meio destas ferramentas, o usuário é capaz de definir o volume de interesse (VOI) dentro do volume. Desta forma, elas podem ser consideradas uma forma fraca de segmentação interativa, que não está ligada à aquisição ou a área de aplicação da técnica. A Figura 2.2 mostra a aplicação de ferramentas de recorte na segmentação do fígado em uma imagem TC abdominal.



(a) (b) (c) (d)

Figura 2.2: Segmentação do VOI (fígado) por meio das ferramentas de recorte interativo. (a) Imagem original (TC abdominal); (b) definição de um volume grosseiro com planos de recorte binário; segmentação do VOI com uma borracha virtual (c) e uma escavadeira virtual (d).

As ferramentas interativas propostas aqui têm o objetivo de deixar para o usuário a decisão do que deve ser visualizado ou não no volume de dados. Neste processo, o usuário pode usar sua experiência para a tomada de decisões. No entanto, as ferramentas devem ser flexíveis o bastante para que ele possa aplicar o seu conhecimento sobre a imagem. No caso da visualização do fígado em TCs abdominais, o resultado da sessão interativa ilustrada na Figura 2.2 pode ser visto na Figura 2.3.

Figura 2.3: Fígado segmentado de um TC abdominal, resultado dos recortes ilustrados na Figura 2.2.

1.2 Objetivo

Propor ferramentas interativas para auxiliar na solução da oclusão de objetos de interesse em imagens 3D visualizadas diretamente utilizando recursos de programação das placas gráficas. Estas ferramentas serão aplicadas na inspeção interativa da imagem tridimensional de fígados humanos, para auxiliar o aprendizado da anatomia deste órgão.

1.3 Principais Contribuições

As principais contribuições deste trabalho em relação a publicações anteriores são:

a) o algoritmo de representação e recorte de poliedros WSG (*Weighted Sweep Graph*), descrito na Seção 4.2;
b) a implementação da interface de controle da "borracha virtual" no hardware gráfico, descrita na Seção 4.4;
c) a implementação da interface de controle da "escavadeira virtual" no hardware gráfico, descrita na Seção 4.5;

d) a implementação da ferramenta de classificação da imagem volumétrica no hardware gráfico, descrita na Seção 6.2.

1.4 Organização

O projeto das ferramentas propostas neste estudo é muito influenciado pela programação dos dispositivos gráficos de PCs. Esta programação inclui a interface de programação para dispositivos gráficos (OpenGL) e as capacidades atuais destes dispositivos. No Capítulo 0 são vistos os detalhes da interface de programação OpenGL e o desenvolvimento do hardware gráfico ao longo da última década, que servem de contexto para a proposta das ferramentas de interação.

No Capítulo 0 são discutidos os principais conceitos sobre visualização direta de imagens 3D relacionados a este estudo. Também é apresentado o esquema necessário para a implementação do método de visualização, onde os pontos do fluxo de *rendering* no qual as ferramentas de interação podem ser implementadas são discutidos.

No Capítulo 0 são propostas as técnicas de recorte volumétrico para a visualização direta baseada no mapeamento de texturas. Como é mostrado no Capítulo 0, existem pontos no fluxo de visualização que podem ser usados para a implementação das ferramentas de interação; no Capítulo 0, é detalhada a forma como estes pontos podem ser explorados. A partir desta observação, são propostos os algoritmos para as ferramentas de recorte e sua integração ao método de *rendering* direto usando o hardware das placas gráficas.

No Capítulo 0 são mostrados os resultados da aplicação das ferramentas propostas sobre imagens médicas. Primeiro é vista a importância de ferramentas de inspeção de imagens 3D no aprendizado de anatomia, e algumas das ferramentas disponíveis atualmente. Na seqüência, são vistos os resultados da aplicação das ferramentas propostas neste estudo em imagens de TCs abdominais contrastados.

No Capítulo 0, são discutidas as principais questões envolvidas no projeto e implementação das ferramentas propostas. Também são discutidas sugestões para lidar com algumas das limitações da implementação atual das ferramentas. Finalmente, são propostas possíveis extensões das ferramentas criadas neste estudo.

2 MATERIAIS

2.1 Introdução

O projeto e a implementação das ferramentas propostas neste estudo é muito influenciado pela forma de programação dos dispositivos gráficos de PCs. Esta programação inclui a interface com dispositivos gráficos (padrão OpenGL, ver Seção 2.2) e as capacidades de programação disponíveis atualmente nestes dispositivos.

A interface de programação adotada, o OpenGL, define o fluxo de visualização (*pipeline*) de primitivas geométricas no hardware gráfico. O *pipeline* definido pelo OpenGL serve como base para a proposta das ferramentas de interação, e é discutido na Seção 2.2.

As capacidades atuais dos dispositivos gráficos de PCs têm influenciado muitas pesquisas envolvendo a visualização direta de imagens 3D (ENGEL; KRAUS; ERTL, 2003; WEISKOPF; ENGEL; ERTL, 2003). A geração atual de placas gráficas permite a programação de alguns estágios do *pipeline* de visualização. A flexibilidade da programação destes dispositivos tem possibilitado a implementação até mesmo de algoritmos de propósito geral, como a detecção de colisão entre poliedros (GOVINDARAJU *et al.*, 2003) e métodos de álgebra linear (KRÜGER; WESTERMANN, 2003). Neste estudo, a programabilidade do hardware gráfico também é explorada. Assim, na Seção 2.3 são discutidas em maiores detalhes as capacidades atuais do hardware gráfico.

2.2 OpenGL

Em 1992, a SGI® lançou uma interface de programação que veio a se tornar o primeiro padrão mundialmente reconhecido para a programação de dispositivos gráficos. Este é o OpenGL, cujo desenvolvimento foi fortemente baseado na biblioteca IRIS GL, também de propriedade da SGI®. O objetivo do OpenGL é fornecer um fluxo padronizado de visualização de imagens e primitivas geométricas, como mostra a Figura 2.1.

A definição do *pipeline* de visualização foi o passo mais importante na proposta do OpenGL. O *pipeline* gráfico recebe dados geométricos representando um objeto ou uma cena e cria imagens bidimensionais a partir deles. A aplicação provê os dados geométricos como uma coleção de vértices que podem formar polígonos, linhas e pontos. A imagem resultante tipicamente representa o que um observador pode ver de um ponto particular de visualização.

A definição do fluxo de visualização modular permitiu o desenvolvimento de hardware específico para o tratamento de cada estágio, o que pode ser visto ao

acompanhar a evolução das placas gráficas. A Figura 2.2 mostra um diagrama de blocos mais detalhado do *pipeline* de visualização do OpenGL.

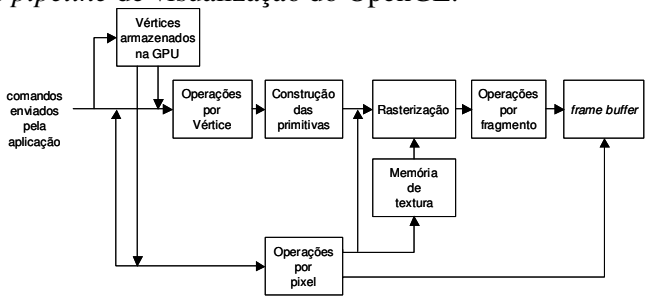


Figura 2.1: Fluxo de visualização do OpenGL (baseada em (SHREINER, 1999)).

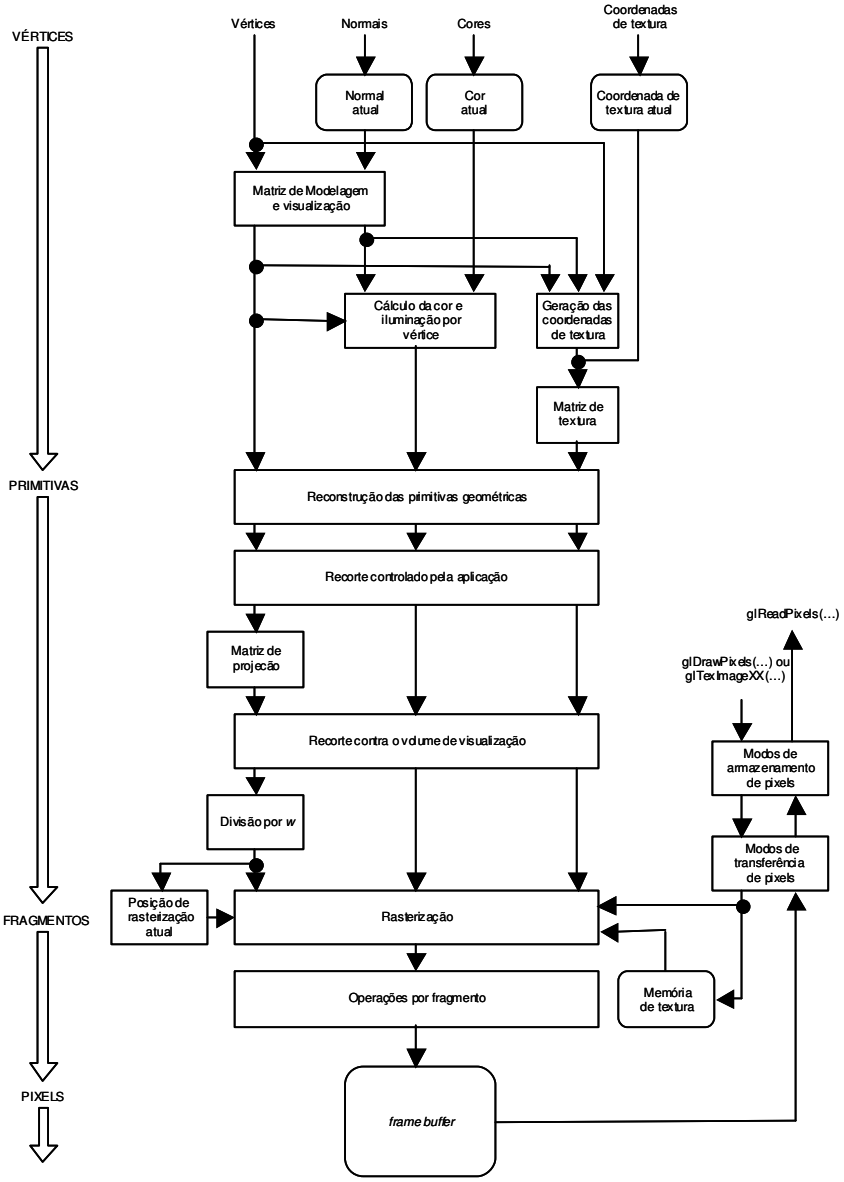


Figura 2.2: Diagrama de blocos do *pipeline* de visualização do OpenGL 1.1 (baseado em (SHREINER, 1999)).

Na maior parte do diagrama podem ser vistas três setas verticais entre os principais blocos. Estas setas representam vértices e os dois tipos primários de dados que podem ser associados a eles: valores de cor e coordenadas de texturas. Também pode ser visto que o tipo de dados se altera conforme a progressão de estágios dentro do *pipeline*: vértices tornam-se primitivas geométricas, fragmentos e finalmente pixels no *frame buffer* (área em memória para o armazenamento do conteúdo do plano de projeção) (SHREINER, 1999). Esta progressão é discutida em maiores detalhes nas seções seguintes.

2.2.1 *Pipeline* de Visualização

A aplicação envia ao hardware gráfico uma seqüência de vértices representando primitivas geométricas, tipicamente polígonos, linhas e pontos utilizando uma série de funções especiais. Um vértice define um ponto, uma extremidade de linha ou o encontro de duas arestas de um polígono. Cada vértice tem, associados a si, uma posição e vários outros atributos como cor, uma cor secundária (chamada cor especular), uma ou mais coordenadas de textura e um vetor normal. O vetor normal indica a direção da face à qual o vértice pertence, sendo tipicamente usado em cálculos de iluminação.

2.2.1.1 Transformação dos Vértices

A transformação dos vértices é o primeiro estágio no *pipeline* gráfico. Neste estágio, uma seqüência de operações é aplicada a cada vértice, o que inclui transformar a posição do vértice do espaço de objeto para o espaço de recorte (*clip coordinates*), transformar as coordenadas de textura e calcular a iluminação do vértice para a determinação de suas cores primária e secundária. Na Figura 2.3 são mostrados os estágios do diagrama apresentado na Figura 2.2 nos quais são feitas as transformações nos vértices.

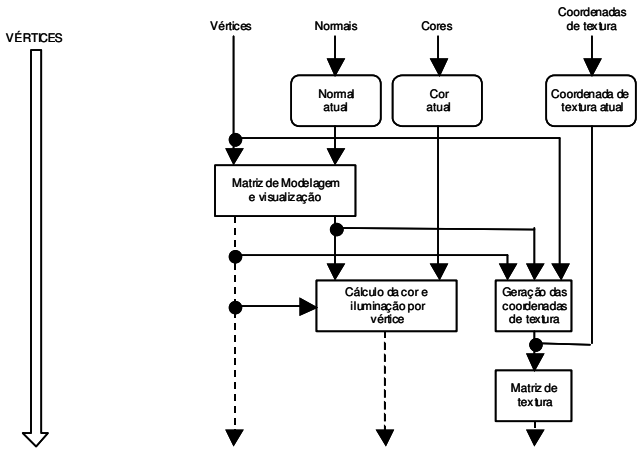


Figura 2.3: Estágio de transformação dos vértices no *pipeline* do OpenGL .

2.2.1.2 Montagem de Primitivas e Rasterização

Os vértices transformados seguem em seqüência para o próximo estágio, chamado construção de primitivas e rasterização (*primitive assembly and rasterization*). Este estágio inclui vários blocos mostrados separadamente no diagrama da Figura 2.2; no

entanto, como são operações seqüenciais configuráveis, é comum referenciá-los como um único estágio.

Primeiro, o passo de construção de primitivas reconstrói as primitivas baseado na informação enviada pela aplicação, que acompanha a seqüência de vértices (até agora, os vértices eram tratados separadamente). Isto resulta numa seqüência de triângulos, linhas ou pontos, que podem ser recortadas pelo volume de visualização, e também por qualquer plano de recorte especificado pela aplicação.

Polígonos que "sobrevivem" ao recorte devem ser rasterizados. A rasterização é o processo pelo qual a primitiva geométrica é convertida em uma imagem bidimensional. O modo como este processo é realizado deu origem ao termo "rasterizador", derivado de *raster scanning*, ou seja, o processo de formação de uma imagem feito pela varredura linha por linha (em televisão). Cada ponto desta imagem contém uma posição associada, um valor de profundidade (distância ao *frame buffer*) e um conjunto de parâmetros interpolados como cor, cor secundária e uma ou mais coordenadas de textura. Estes parâmetros são derivados dos vértices transformados que formavam a primitiva geométrica antes da rasterização. Juntos, um ponto e suas informações associadas são chamados de "fragmento". O termo fragmento é usado porque a rasterização "quebra" cada primitiva geométrica, como um triângulo, em fragmentos do tamanho de pixels para cada pixel que a primitiva cobre no *frame buffer*. Se um fragmento passar pelos testes do estágio subseqüente, ele irá somar sua contribuição no *frame buffer*. Também é importante citar aqui que não há relação entre o número de vértices de uma primitiva e o número de fragmentos que são gerados na rasterização. Um triângulo que ocupe pouco espaço no *frame buffer* vai ser dividido em poucos fragmentos. Ocupando um espaço maior no *frame buffer*, o mesmo triângulo seria dividido em um número maior de fragmentos, pois tem uma área (em pixels) maior. Na Figura 2.4 são mostrados os estágios do diagrama da Figura 2.2 no qual são feitas a construção de primitivas e a rasterização.

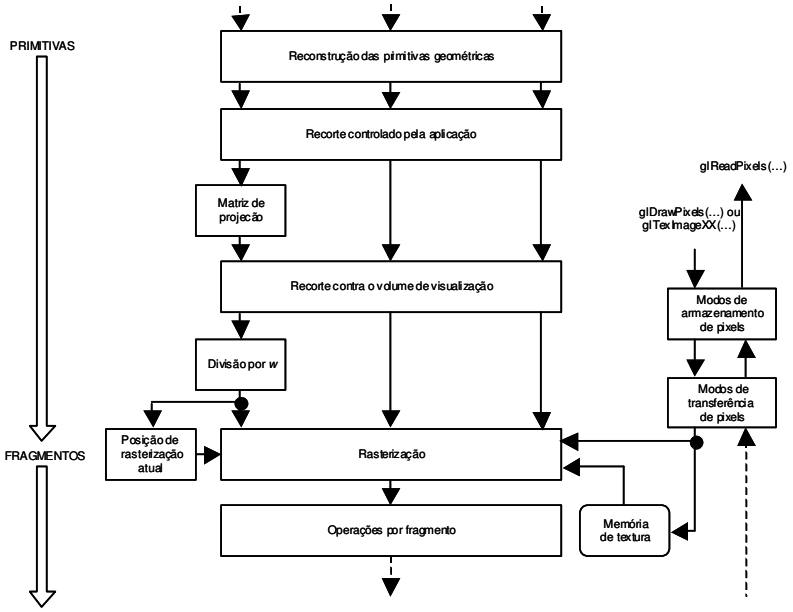


Figura 2.4: Estágio de construção de primitivas e rasterização. Neste estágio, os vértices são primeiro reorganizados em primitivas geométricas; a partir destas, o rasterizador

calcula quais porções de cada primitiva são visíveis e as dividem em fragmentos, que são os potenciais pontos de atualização do *frame-buffer*.

2.2.1.3 Operações Por Fragmento

A partir do momento em que a primitiva é rasterizada em uma coleção de zero ou mais fragmentos, o estágio de operações por fragmento realiza uma seqüência de operações e aplicações de texturas para computar a cor final de cada fragmento. As operações combinam os parâmetros de cor, cor secundária e as cores recuperadas a partir das coordenadas de textura para o cálculo da cor final do fragmento. A seqüência final de operações por fragmento envolve uma série de testes, imediatamente antes de atualizar o *frame buffer*. Durante este estágio, cada fragmento é testado em relação aos seguintes aspectos: recorte contra uma região do *frame buffer* fixada pela aplicação (*scissor test*); faixa de transparências fixada pela aplicação (*alpha test*); teste contra o conteúdo de um *buffer* auxiliar do OpenGL, através de operações lógicas controladas pela aplicação (*stencil test*) e profundidade, que determina se a visualização do fragmento está obstruída por outro em relação ao ponto de visão (*depth test*). Se qualquer dos testes falha, este estágio descarta o fragmento sem atualizar a cor do pixel no *frame buffer*. Passando nos testes, a cor final do fragmento é combinada com a cor do pixel correspondente do *frame buffer*. Finalmente, uma operação de escrita no *frame buffer* atualiza a cor e o valor de profundidade do pixel, que é armazenado em um *buffer* adicional, o *depth buffer*. Na Figura 2.5 é mostrado o estágio do diagrama da Figura 2.2 no qual é feita a seqüência de operações em cada fragmento.

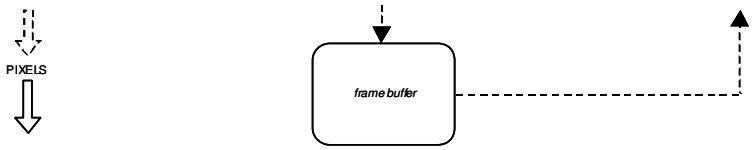


Figura 2.5: Estágio de operações por fragmento.

2.3 Hardware Gráfico

Na última década houve um grande esforço para a construção de hardware especializado para acelerar as operações realizadas pelo *pipeline* gráfico. O resultado é uma geração de placas gráficas que praticamente engloba todos os estágios do *pipeline* em um único *microchip*, com poder de processamento muito superior ao alcançado na CPU. Partes do *pipeline* gráfico também podem agora ser controladas pela aplicação, como o tratamento de vértices e fragmentos, através de conjuntos de instruções muito semelhantes às usadas na CPU.

A empresa IBM® introduziu o hardware VGA (*Video Graphics Array*) em 1987. Naquela época, a controladora VGA não era mais do que um *frame-buffer*. Isto significava que a CPU era responsável pela atualização de todos os pixels diretamente, o que é raramente feito nos dias de hoje. A empresa nVIDIA® Corporation introduziu o termo "GPU" (*Graphics Processor Unit*) no final da década de 90, quando o termo "controlador VGA" não mais descrevia adequadamente o hardware gráfico de um PC.

A evolução das GPUs está organizada grosseiramente em quatro gerações. Cada geração apresenta melhorias no desempenho e envolve maior programabilidade da GPU. Cada geração também influencia e incorpora a funcionalidade das duas maiores

interfaces de programação 3D, o OpenGL e o DirectX (o DirectX é o conjunto das interfaces de programação multimídia da Microsoft, incluindo o Direct3D para a programação de aplicações gráficas). Nas seções a seguir, as características de cada geração são discutidas em separado.

2.3.1 Aceleração Gráfica Pré-GPU

Antes da introdução das GPUs, companhias como a SGI® e Evans&Sutherland® desenvolviam hardware gráfico caro e especializado. Os sistemas gráficos desenvolvidos por estas companhias introduziram muitos dos conceitos, como transformações de vértices e mapeamento de texturas, que são utilizados hoje. Estes sistemas foram muito importantes para o desenvolvimento histórico da computação gráfica, mas, como eram muito caros, não alcançaram o mesmo sucesso de mercado das GPUs atuais. Hoje, as GPUs são muito mais poderosas e baratas que qualquer destes sistemas anteriores. A Figura 2.6 mostra em destaque os blocos do *pipeline* gráfico do OpenGL implementados no hardware das placas gráficas da geração pré-GPU.

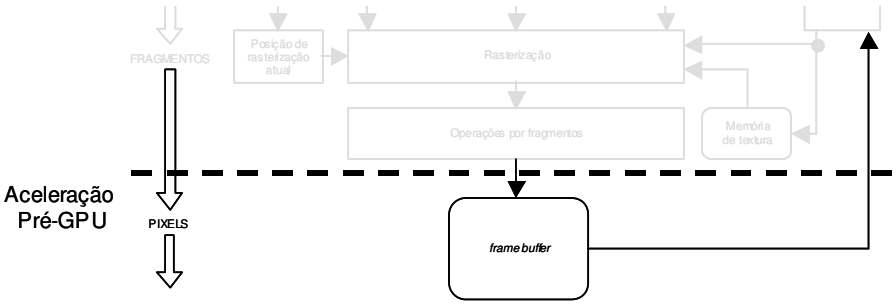


Figura 2.6: Blocos do *pipeline* gráfico do OpenGL implementados no hardware das placas gráficas da geração pré-GPU.

2.3.2 Primeira Geração de GPUs

A primeira geração de GPUs (em torno de 1998) inclui as placas gráficas TNT2 da nVIDIA®, a Rage da companhia ATI® e a Voodoo3 da companhia 3dfx®. Estas placas são capazes de rasterizar triângulos pré-transformados pela aplicação antes de serem enviados à GPU e aplicar uma ou mais texturas. Quando executam aplicações 3D ou 2D, estas GPUs praticamente livram a CPU do trabalho de atualizar pixels individualmente. Entretanto, as GPUs desta geração têm duas limitações claras. Em primeiro lugar, falta a capacidade de transformar vértices, e estas transformações ainda ocorrem na CPU. Em segundo, elas têm um conjunto muito limitado de operações para combinar texturas e calcular a cor dos fragmentos rasterizados. A Figura 2.7 mostra em destaque os blocos do *pipeline* gráfico do OpenGL implementados no hardware das placas gráficas da primeira geração.

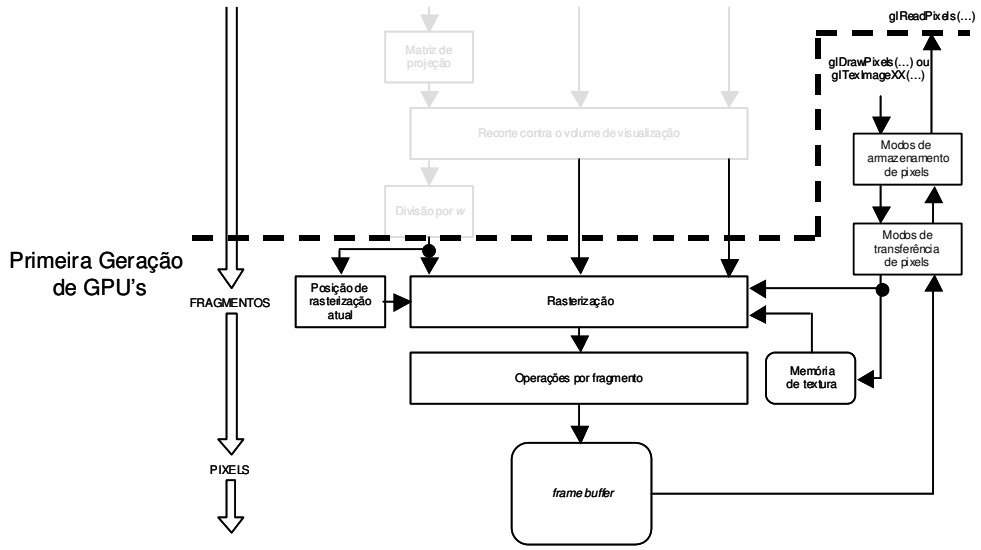


Figura 2.7: Blocos do *pipeline* gráfico do OpenGL implementados no hardware das placas gráficas da primeira geração de GPUs.

2.3.3 Segunda Geração de GPUs

A segunda geração de GPUs (1999-2000) inclui as placas gráficas GeForce 256 e GeForce2 da nVIDIA®, a Radeon 7500 da ATI® e a Savage3D da companhia S3®. Estas GPUs livram a CPU das transformações e iluminação dos vértices. A transformação rápida dos vértices era uma das principais características que diferenciavam as poderosas estações de trabalho dos PCs antes desta geração. Embora o conjunto de operações matemáticas para combinação de texturas e coloração de fragmentos tenha sido expandido nesta geração para incluir texturas com *environment mapping* (*cube map textures*) e operações matemáticas com sinal, as possibilidades ainda são limitadas. Por outro lado, esta geração é mais configurável (várias características do *pipeline* fixo podem ser configuradas pela aplicação). As GPUs desta geração embarcam praticamente todos os estágios do *pipeline* gráfico ilustrado na Figura 2.2.

2.3.4 Terceira Geração de GPUs

A terceira geração de GPUs (2001) inclui as placas gráficas GeForce3 e GeForce4 da nVIDIA®, a Xbox da Microsoft® e a Radeon 8500 da ATI®. Esta geração de GPUs trouxe as unidades programáveis de tratamento de vértices. Além de suportar os modelos convencionais de transformação e iluminação do OpenGL e do DirectX 7, estas GPUs oferecem à aplicação a possibilidade de especificar o algoritmo de processamento (*shaders*) dos vértices. O tratamento de fragmentos ganha novas opções de configuração, mas ainda não há controle suficiente para que possa ser considerado "programável". Contando com o tratamento de vértices programável, mas ainda sem esta capacidade em nível de fragmentos, esta geração é considerada uma geração de transição. Na Figura 2.8 é ilustrado o bloco do *pipeline* que pode receber um conjunto de instruções informado pela aplicação para o processamento dos vértices. O DirectX 8

e extensões do OpenGL permitem a programação de transformações de vértices para as aplicações.

2.3.5 Quarta Geração de GPUs

A quarta e atual geração de GPUs (2002 ao início de 2004) inclui as placas gráficas GeForce FX da nVIDIA® e as Radeon 9700 e Radeon 9800 da ATI®. Estas GPUs permitem o tratamento programável de vértices e fragmentos. Este nível de programabilidade trouxe a possibilidade de transferir completamente transformações complexas de vértices e operações de coloração de fragmentos da CPU à GPU. Em adição à determinação da cor final, as novas capacidades dos programas de fragmentos também incluem, por exemplo, determinar uma nova profundidade ou mesmo descartar fragmentos para evitar a atualização do *frame buffer*. O DirectX 9 e várias extensões do OpenGL expõem à aplicação os estágios programáveis da GPU. A Figura 2.8 mostra o diagrama de blocos do *pipeline* gráfico programável das GPUs atuais.

A tendência dominante no projeto de hardware gráfico é o esforço para aumentar ainda mais a programabilidade da GPU. Hoje, diversas linguagens (*C for Graphics* da nVIDIA®, *High Level Shading Language* da Microsoft®, OpenGL 2.0 *Shading Language*, etc) oferecem formas de traduzir algoritmos em seqüências de instruções que a GPU possa executar.

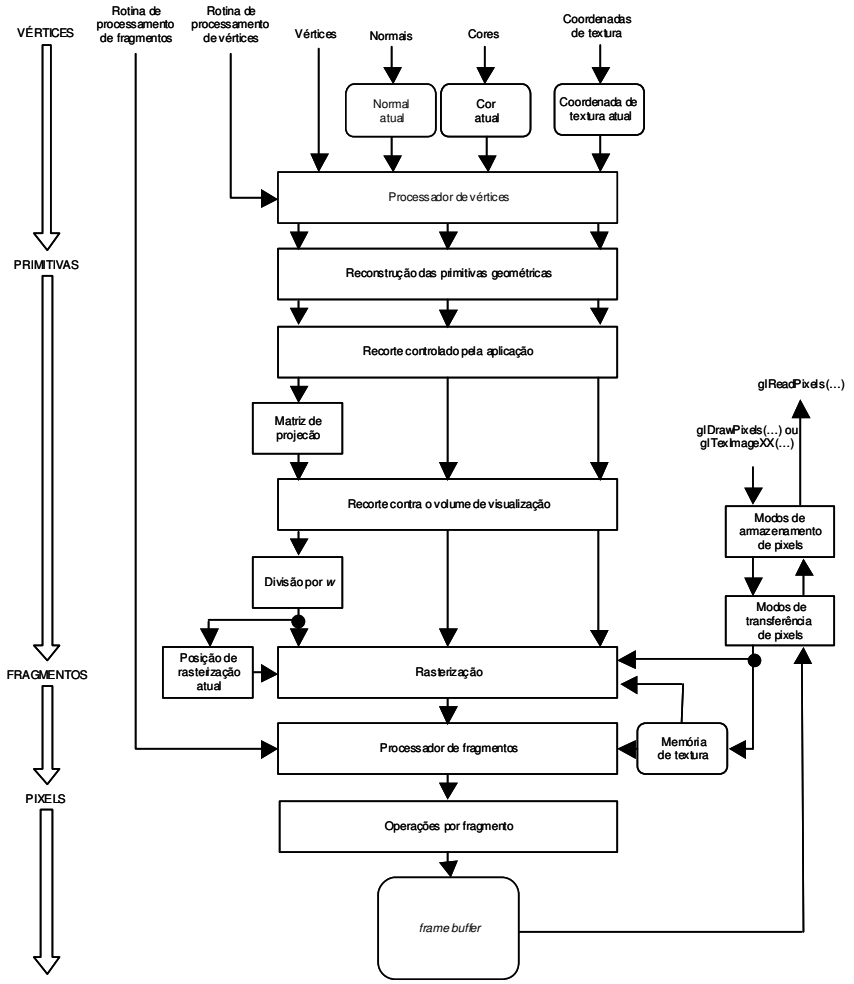


Figura 2.8: Diagrama de blocos do *pipeline* gráfico programável na quarta geração.

2.4 Conclusão

A programabilidade disponível nas GPUs tem sido explorada em diversas publicações. O controle explícito sobre o cálculo da cor de cada fragmento, em especial, possibilitou o desenvolvimento e a adaptação de uma série de algoritmos de visualização de volumes (ENGEL; KRAUS; ERTL, 2003; KRÜGER; WESTERMANN, 2003; WEISKOPF; ENGEL; ERTL, 2003). Outras capacidades disponíveis atualmente, como a multitextura, permitem a associação de uma quantidade maior de informações ao *rendering* de cada fragmento. Estas informações chegam ao processador de fragmentos como cores de textura, mas podem ser interpretadas de forma mais apropriada por *shaders* informados pela aplicação. Esta abordagem é explorada por muitos autores (ENGEL; KRAUS; ERTL, 2003; KRÜGER; WESTERMANN, 2003; WEISKOPF; ENGEL; ERTL, 2003). O *pipeline* discutido na Seção 2.2 é a base para a proposta e a programação das ferramentas de interação apresentadas nesta dissertação. No Capítulo 0 ele é revisitado, com objetivo de determinar quais pontos do fluxo de visualização podem ser explorados para a implementação das ferramentas de interação com imagens 3D. No Capítulo 0, são mostradas as formas como estas capacidades foram exploradas neste estudo.

3 VISUALIZAÇÃO DIRETA DE VOLUMES BASEADA EM TEXTURAS

3.1 Introdução

Neste capítulo é discutida a idéia geral da visualização direta baseada em texturas. Algoritmos modernos de visualização direta são baseados nos métodos apresentados por Levoy (1988), Drebin, Carpenter e Hanrahan (1988), Sabella (1988) e Upson e Keeler (1988). Desde a proposta destes trabalhos, muitos autores exploram extensões (PFISTER, 1999) e otimizações (KRÜGER; WESTERMANN, 2003; LACROUTE; LEVOY, 1994; LEVOY, 1990) dos modelos originais. O surgimento da técnica de mapeamento de texturas, no entanto, inspirou alguns autores (CABRAL; CAM; FORAN, 1994; CULLIP; NEUMANN, 1993) a explorar novas abordagens para métodos de visualização direta. Cabral, Cam e Foran (1994) demonstram que tanto a reconstrução quanto a interação com imagens volumétricas visualizadas diretamente são possíveis em hardware gráfico com a capacidade de mapeamento de texturas 3D.

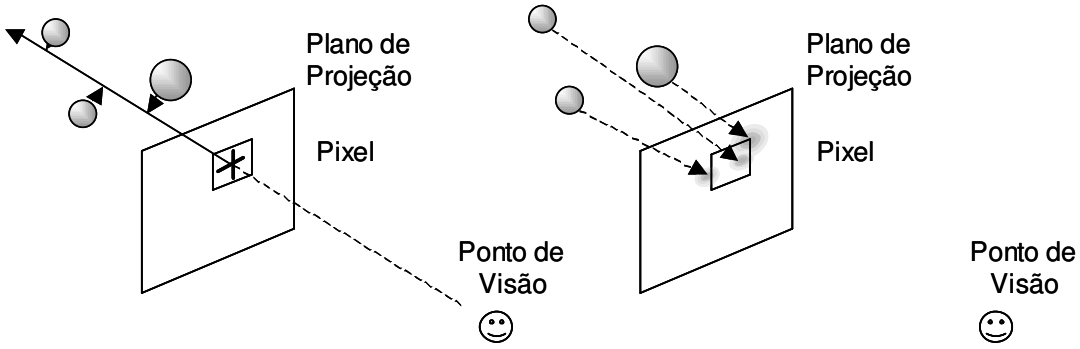
A partir da terceira geração de GPUs (2001), o mapeamento de texturas 3D está incorporado no hardware gráfico. O ganho de desempenho, somado à programabilidade do cálculo de fragmentos, motivou alguns autores (ENGEL; KRAUS; ERTL, 2003; KRÜGER; WESTERMANN, 2003; WEISKOPF; ENGEL; ERTL, 2003) a explorar as capacidades da GPU para a visualização direta. Além dos esquemas abordados em pesquisas anteriores (*slice-based rendering* (CULLIP; NEUMANN, 1993)), técnicas como o *ray-casting* proposto por Levoy (1988) também são implementados na GPU (KRÜGER; WESTERMANN, 2003). Hoje é possível dizer que a visualização direta baseada em texturas é uma das principais linhas de pesquisa em visualização volumétrica.

Seguindo a linha destas pesquisas, este trabalho explora o mapeamento de texturas para a visualização direta e implementação de ferramentas interativas para a seleção do VOI. A implementação do sistema de visualização segue o modelo de Cullip e Neumann (1993), isto é, a visualização baseada em fatias (*slice-based rendering*). Neste Capítulo, este esquema é discutido, apresentando como a sua escolha influi na proposta das ferramentas de interação. Finalmente, é mostrado o *pipeline* de visualização do esquema utilizado, onde são identificados os pontos que podem ser explorados para a implementação das ferramentas interativas.

3.2 Visualização Direta de Volumes

Desde sua introdução no final da década de 80 por Levoy (1988), diferentes MVDs foram propostos segundo abordagens que podem ser fracamente classificadas como

baseadas em imagem e baseadas em objeto. Nos métodos baseados em objeto, a contribuição de cada elemento do volume é computada sobre o plano de projeção (*splatting* (WESTOVER, 1990), texturas (CULLIP; NEUMANN, 1993; CABRAL; CAM; FORAN, 1994), etc). Já nos métodos baseados em imagem, raios são lançados dos pixels do plano de projeção em direção ao volume, e as contribuições dos elementos ao longo de cada raio são combinadas para formar a cor final do pixel associado ao raio (*ray-casting* (LEVOY, 1988), etc). Esquemas ilustrando as duas abordagens podem ser vistos na Figura 3.1.



(a) (b)

Figura 3.1: Classificação comumente usada para métodos de visualização direta de volumes. (a) Métodos baseados em imagem, onde raios são lançados dos pixels do plano de projeção em direção ao volume, e as contribuições dos elementos ao longo de cada raio são combinadas para formar a cor final do pixel; (b) métodos baseados em objeto, onde a contribuição de cada elemento do volume é computada sobre o plano de projeção.

O tema comum, entretanto, é a solução (aproximada) da integral de visualização volumétrica (MAX, 1995) para cada pixel do plano de projeção. A solução integra as cores atenuadas ao longo de cada raio de visão, entre o observador e cada pixel da tela. Basicamente, esta integral modela a forma como a luz é transportada de cada elemento do volume até cada pixel do plano de projeção. Este processo depende dos vários fenômenos em funcionamento, como a emissão, o espalhamento e a absorção da luz. Modelos físicos que estudam o funcionamento destes fenômenos são chamados de modelos óticos.

Os primeiros modelos óticos para o *rendering* foram desenvolvidos por Blinn (BLINN, 1982) e estendidos por Kajiya e Herzen (1984) para volumes povoados por partículas (elementos) de densidades não-homogêneas. Blinn (1982) adaptou a teoria de transferência de radiação do modelo de partículas para síntese de imagens realísticas de Esposito (1979), Chandrasekhar (1960) e outros.

Algoritmos modernos de visualização volumétrica são baseados em Levoy (1988), Drebin, Carpenter e Hanrahan (1988), Sabella (1988) e Upson e Keeler (1988). Como a solução analítica das equações integro-diferenciais para a transferência de luz através de um modelo não-homogêneo geralmente não é possível analiticamente (KAJIYA; VON HERZEN, 1984; MAX, 1995) e é numericamente muito custosa, foram desenvolvidas diversas simplificações para o seu tratamento computacional. Entre estas simplificações estão o espalhamento simples de luz e o não-sombreamento entre as partículas do volume.

O espalhamento simples de luz ignora múltiplas fontes de luz, o que é verdade se as partículas não são muito brilhantes (de baixa reflectividade). Pela suposição de não-sombreamento, as fontes de luz não provocam sombra enquanto iluminam o volume (mesma luminosidade em todas as partículas do volume), o que economiza esforço no cálculo da contribuição da fonte de luz.

3.3 Aproximação da Visualização Direta por Texturas

Neste trabalho assume-se o modelo ótico de absorção e emissão (*absorption plus emission*) apresentado por Max (1995). Este modelo é útil para a visualização de dados médicos (meios não-homogêneos).

Neste modelo, considera-se que o volume é povoado por uma nuvem de partículas. As partículas podem atenuar a luz que vem até elas, assim como podem emitir luz própria. Para um sistema de partículas esféricas idênticas, tem-se de Wittenbrink, Malzbender e Gross (1998) que a contribuição ao longo de um raio de visão pode ser aproximada pela seguinte soma de Riemann:

$$cor = \sum_{n=1}^{N} c[n]\alpha[n]\prod_{m=1}^{n-1}(1-\alpha[m]), \tag{3.1}$$

para N amostras ao longo de um raio. O volume é amostrado nas posições n e m, onde $c[i]$ e $\alpha[i]$ são a cor (tipicamente uma grandeza vetorial) e a opacidade (escalar) amostradas na posição i do raio, respectivamente. Em outras palavras, a cor no pixel que origina o raio é o somatório das cores $c[n]$ emitidas em cada ponto n ao longo do raio, atenuadas pela sua própria opacidade $\alpha[n]$ e a transparência dos pontos em frente ao pixel na linha de visão do observador.

Usando o modelo ótico discutido, a equação de composição *back-to-front* (de trás para frente, em relação ao ponto de visão) tem a seguinte forma (LEVOY, 1988):

$$cor = (1-\alpha_{frente})cor_{trás} + cor_{frente}\alpha_{frente}. \tag{3.2}$$

O OpenGL tem funções que suportam este tipo de combinação de fragmentos. No entanto, para que a técnica de visualização direta de volumes seja computada inteiramente no hardware gráfico, ainda são necessários dois requisitos. Em primeiro lugar, deve haver alguma forma de armazenar a imagem 3D na memória da GPU. Em segundo lugar, deve ser desenvolvido um sistema de amostragem do conjunto de dados por meio de primitivas geométricas, que são os tipos de dados suportados pela GPU.

Este é o propósito da técnica da visualização direta através do mapeamento de texturas. Basicamente, a técnica se resume a reamostrar a imagem 3D, representada por uma pilha de texturas 2D ou uma textura 3D, através de um conjunto de primitivas geométricas. A Figura 3.2 mostra um esquema dos principais passos do *rendering* volumétrico por texturas.

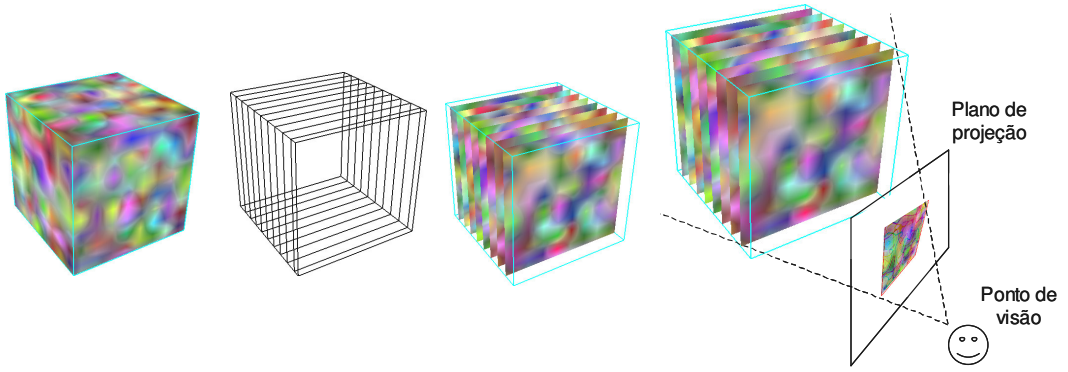


Figura 3.2: Esquema da visualização de volume de dados por texturas; (a) volume de dados; (b) polígonos de amostragem; (c) mapeamento dos polígonos de amostragem no espaço de textura; (d) *rendering* dos polígonos de amostragem no *frame buffer*.

Na técnica tradicional de visualização direta por mapeamento de texturas, a integral do raio para cada pixel é calculada através do *rendering* de polígonos de reamostragem através do volume (*slice-based rendering* (CULLIP; NEUMANN, 1993)). Deste modo, a emissão e a absorção ao longo de cada raio de visão são computadas por meio da amostragem e combinação do volume para cada passo da integração. Cada fragmento de um polígono corresponde à contribuição de um segmento de raio à integral do *rendering* volumétrico para cada pixel do *frame buffer*. A principal vantagem é que toda a tarefa pode ser feita no hardware gráfico, sendo limitada apenas pelo desempenho da GPU.

Na implementação da visualização direta por mapeamento de texturas, a técnica se resume ao *rendering* de polígonos texturizados. Segundo a notação descrita por Ikits *et al.* (2004), este *pipeline* pode ser dividido em três estágios: inicialização, atualização (*update*) e desenho (*drawing*).

No estágio de inicialização, o conjunto de dados volumétricos é armazenado como uma textura 3D na GPU. Os dados podem ser pré-processados (para a pré-classificação, por exemplo), ou podem enviar à GPU informações adicionais para o processamento em estágios posteriores. Esta decisão influi na implementação do estágio de desenho.

Atualizações no *pipeline* são exigidas quando o usuário interage com o sistema, mudando os parâmetros de visualização (mudança do ponto de visão ou do plano de projeção, por exemplo). Na abordagem de *slice-based rendering* com planos paralelos ao plano de projeção, esta mudança exige que todos os polígonos de amostragem sejam reconstruídos.

O estágio de desenho é responsável por gerar a imagem final. Neste estágio, a cor de cada fragmento dos polígonos de amostragem é calculada e combinada no *frame buffer*. A cor final comumente é a cor de textura, correspondente ao espaço do conjunto de dados amostrado por este fragmento. A combinação da cor do fragmento com a cor do pixel correspondente no *frame buffer* é feita de acordo com a Equação 3.2.

Na Figura 3.3, é mostrado o *pipeline* de visualização direta por texturas, baseado no diagrama da Figura 2.8.

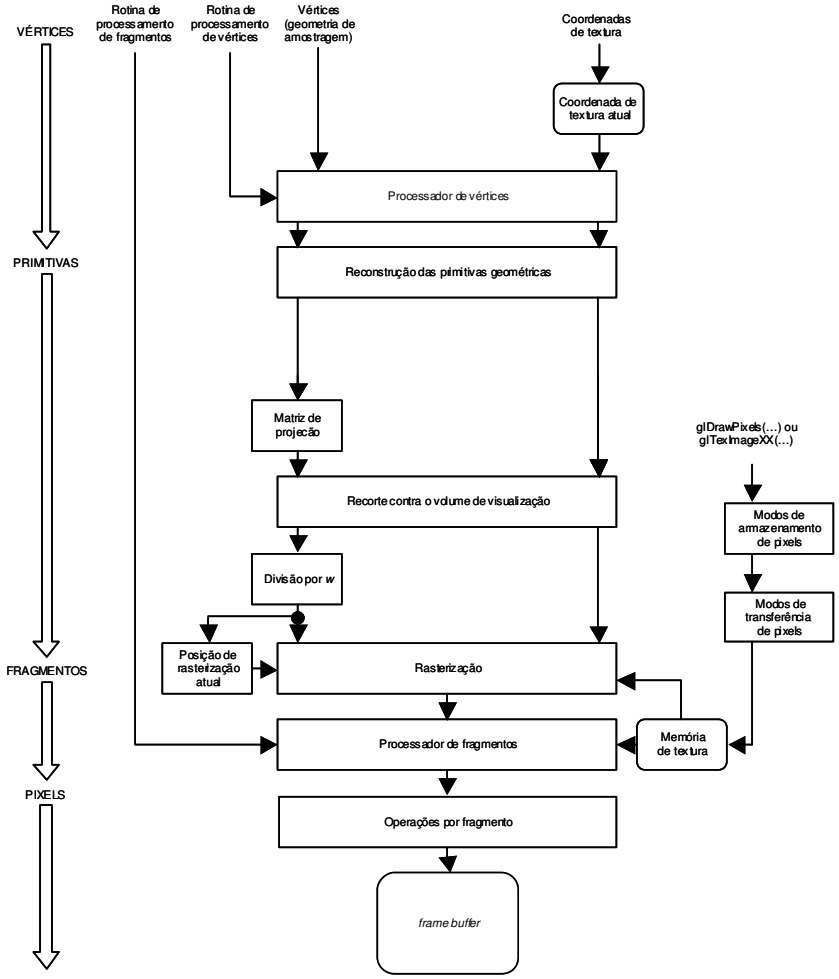


Figura 3.3: *Pipeline* de visualização direta por texturas.

A flexibilidade na programação do cálculo da cor dos fragmentos no estágio de desenho possibilita a adaptação de diversos algoritmos de visualização na GPU. Krüger e Westermann (2003), por exemplo, propõe um sistema de *ray-casting* baseado no método de Levoy (1988). Os autores sugerem um novo fluxo de visualização iterativo para a construção da imagem final. Embora eles relatem um ganho de desempenho significativo em relação à técnica de Cullip e Neumann (1993), o objetivo neste Capítulo é discutir um modelo de visualização suficientemente genérico para servir de suporte à proposta das ferramentas de interação. O fluxo de visualização sugerido por Cullip e Neumann (1993) se encaixa nesta exigência, sendo também a base de outras publicações recentes (ENGEL; KRAUS; ERTL, 2003; WEISKOPF; ENGEL; ERTL, 2003).

3.4 Conclusão

A técnica de visualização de volumes baseada em texturas proposta por Cullip e Neumann (1993) já está sedimentada como uma forma eficiente de visualização direta (ENGEL; KRAUS; ERTL, 2003). No entanto, a flexibilidade da programação das GPUs permite adaptar outras técnicas de visualização direta ao hardware gráfico. As texturas

3D, neste contexto, são vistas como uma forma eficiente de armazenamento e recuperação de dados, manipuladas pela técnica de visualização implementada na GPU. Um bom exemplo é a técnica de *ray-casting* proposta por Krüger e Westermann (2003). A despeito das vantagens de uma ou outra técnica, as ferramentas propostas nesta dissertação têm o objetivo de ser adaptadas a qualquer método de visualização volumétrica baseada em texturas. Para isto, neste capítulo são discutidos apenas os principais aspectos relacionados ao *rendering* de texturas, que resultam no diagrama da Figura 3.3.

O diagrama ilustrado na Figura 3.3 serve como base para a proposta das ferramentas de interação. Ele ilustra os principais blocos do *pipeline* gráfico usados pela implementação da visualização direta de volumes baseada em texturas. A partir da análise do fluxo de dados entre estes blocos, no Capítulo 0 são identificados os pontos onde as ferramentas de interação podem ser implementadas.

4 RECORTE VOLUMÉTRICO BASEADO NO MAPEAMENTO DE TEXTURAS

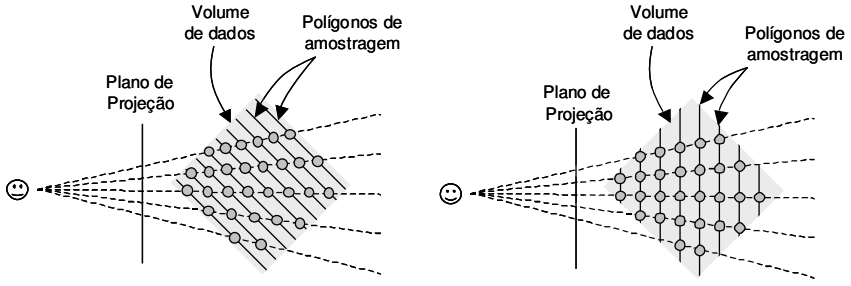
4.1 Visão Geral das Ferramentas

Seguindo uma tendência recente, este projeto explora a técnica de mapeamento de texturas para a visualização direta, integrando ferramentas interativas ao *pipeline* de visualização volumétrica baseada em texturas. Como é discutido no Capítulo 3, este *pipeline* pode ser dividido em três estágios: inicialização, atualização (*update*) e desenho (*drawing*) (IKITS *et al.*, 2003). Nesta seção, os estágios deste *pipeline* são discutidos em relação ao sistema implementado, para posicionar a proposta das ferramentas.

No estágio de inicialização, o volume de dados é lido e enviado para a memória da GPU como uma textura 3D. O volume pode ser processado antes do envio, para aplicação de uma função de transferência (pré-classificação da imagem) ou para o cálculo de parâmetros exigidos pela técnica de visualização (como o gradiente por voxel, para aplicação de técnicas de iluminação). O processamento do volume neste estágio influi na implementação do estágio de desenho. No sistema proposto nesta dissertação, nenhum processamento é feito sobre o volume de dados antes do seu envio à GPU.

O estágio de atualização é responsável pela atualização das estruturas usadas no estágio de desenho. As atualizações são necessárias quando o usuário interage com o sistema, mudando os parâmetros de visualização ou recortando o volume. Os parâmetros de visualização dizem respeito à posição da câmera, e cada mudança exige o recorte dos planos de amostragem paralelos ao plano de projeção contra o volume (*viewport-aligned slices*). O recorte pode ser produzido por três ferramentas diferentes: o recorte planar, a borracha virtual e a escavadeira virtual.

O conjunto de polígonos de amostragem deve ser atualizado cada vez que o recorte planar é usado ou os parâmetros de visualização são mudados. Existem muitos esquemas de amostragem possíveis, sendo que os principais são descritos por Cullip e Neumann (1993). Basicamente, os autores propõem a amostragem por planos recortados contra o espaço de textura (polígonos). Estes planos podem ser paralelos ao plano de projeção ou alinhados a um eixo ortogonal do espaço de textura, como pode ser visto na Figura 4.1.



(a) (b)

Figura 4.1: Posições dos planos de amostragem no espaço de textura (CULLIP; NEUMANN, 1993). (a) Planos alinhados a um eixo ortogonal do espaço de textura e (b) planos paralelos ao plano de projeção.

Para realizar este procedimento, é proposto um algoritmo que explora a coerência espacial entre os polígonos de amostragem do volume, a WSG (Weighted Sweep graph). A WSG é capaz de codificar um VOI convexo para a construção eficiente dos polígonos necessários para a sua amostragem, em qualquer das abordagens sugeridas por Cullip e Neumann (1993). Uma simples modificação no algoritmo permite que a estrutura também suporte operações de recorte planar no VOI. Assim, esta estrutura integra a amostragem exigida pela técnica de visualização baseada em texturas com a primeira ferramenta de recorte. A WSG é proposta na Seção 4.2, e a implementação do recorte planar sobre a estrutura é discutida na Seção 4.3.

A borracha e a escavadeira virtuais são implementadas na GPU e servem para identificar fragmentos que não devem contribuir para a formação da imagem final. Estas ferramentas são usadas para atualizar uma segunda textura 3D, a "máscara de recorte", seguindo o paradigma proposto por Weiskopf, Engel e Ertl (2003). Esta segunda textura é usada como uma máscara para indicar se cada fragmento do volume real é visível ou não.

A borracha virtual é usada para marcar como "não visíveis" todos os texels que estão dentro de um volume imaginário cilíndrico definido pelo usuário. A escavadeira virtual é similar à borracha virtual, mas limita a área de ação da ferramenta a porções próximas à superfície do volume. Para isto, o procedimento da escavadeira exige dois *pixel shaders* executados em duas passagens pelo *pipeline* de *rendering*. As duas ferramentas atualizam a máscara de recorte, que armazena o VOI na GPU. Como a máscara de recorte é uma textura 3D e não pode ser atualizada diretamente pelo resultado do *rendering*, é usada uma textura 2D intermediária para o armazenamento do resultado dos *shaders*. Esta textura é copiada por meio da CPU para a máscara de recorte. As ferramentas implementadas na GPU, borracha e escavadeira, são detalhadas respectivamente nas seções 4.4 e 4.5.

O estágio de desenho combina os parâmetros fixados pelo usuário para a construção da imagem final. No sistema proposto nesta dissertação, este estágio é implementado em um *pixel shader*. O shader recebe como parâmetros os polígonos de amostragem construídos pela WSG (VOI armazenado na CPU), a textura 3D com o volume real e a máscara de recorte. Para cada fragmento dos polígonos de amostragem, o *shader* busca a cor do fragmento na textura com o volume real e testa a sua condição na máscara de recorte. Se o fragmento for visível, ele segue para adicionar sua contribuição no *frame buffer*; caso contrário, ele é descartado.

A Figura 4.2 ilustra um esquema simplificado do sistema proposto. Nele são identificados os principais procedimentos e estruturas de dados, assim como o seu posicionamento (estágios de desenho/atualização e implementação na CPU/GPU). O VOI final é definido pela combinação das informações contidas na WSG e na máscara de recorte. Definido desta forma, ele é comum a todos os estágios do sistema. A atuação do usuário sobre as ferramentas de recorte, mostrada em arcos, atualiza a informação do VOI.

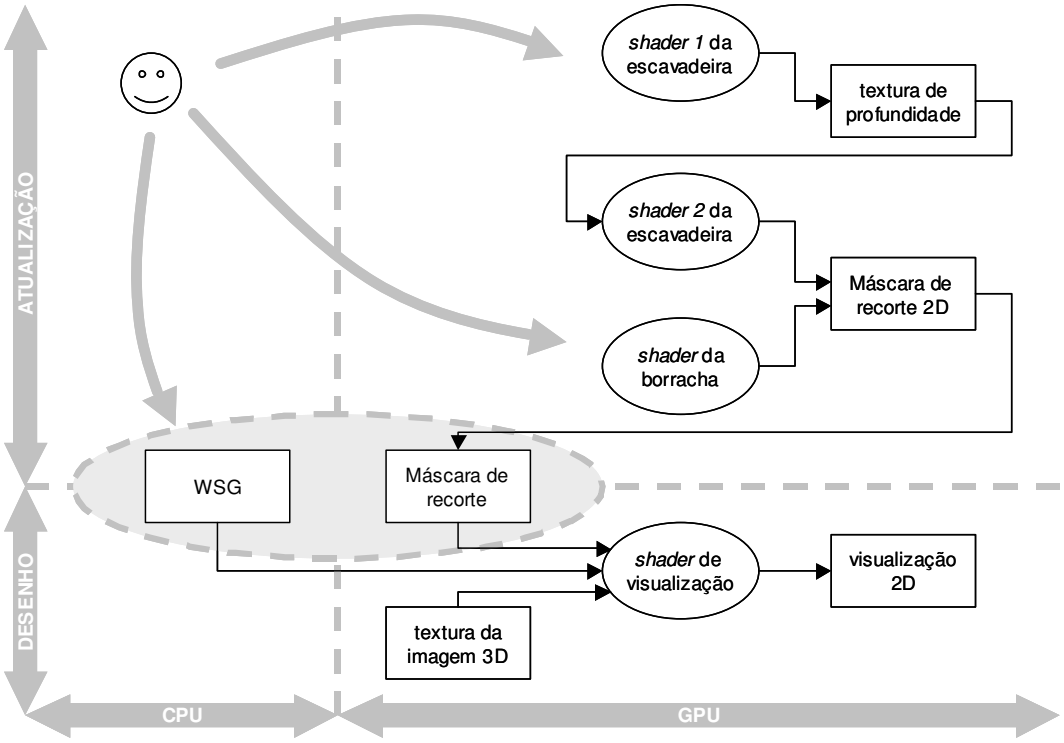


Figura 4.2: Diagrama simplificado do fluxo de visualização e atualização do VOI. Nas operações de recorte, o objetivo é atualizar a WSG e a máscara de recorte. Na visualização, o VOI (mostrado dentro da elipse de contorno tracejado) é combinado com o volume real para gerar a imagem final. O usuário controla os procedimentos de atualização VOI.

4.2 Weighted Sweep Graph

Nesta seção descreve-se o algoritmo de recorte por varredura de plano (*plane-sweep*) usado pela ferramenta de interação. O algoritmo de varredura pode ser visto em mais detalhes em Benley e Ottman (1979), Hertel *et al.* (1983) e Nievergelt e Preparata (1982). Nesta seção, ele é usado para explorar a coerência entre os polígonos recortados contra o espaço de textura. O uso desta abordagem permite propor uma forma eficiente de construir os polígonos de amostragem sem a necessidade de calcular o recorte, como é descrito nos parágrafos a seguir.

A principal idéia da abordagem *plane-sweep* é reduzir um problema estático de n dimensões para um (talvez mais simples) problema dinâmico de $(n–1)$ dimensões. Isto é obtido pela varredura de uma coleção de objetos em um dado espaço por um hiperplano,

mantendo a cada instante o subconjunto de elementos interceptados pelo hiperplano (denominados "elementos ativos") (HERTEL *et al.*, 1983). As intersecções com o hiperplano evoluem continuamente no tempo até que um evento ocorra e altere o conjunto de elementos ativos. Esta abordagem explora a coerência entre os elementos interceptados na evolução do hiperplano de varredura.

Por propósitos didáticos, o algoritmo é primeiro discutido na formulação análoga para 2D (interseção entre linhas e polígonos convexos) antes de ser descrito para o caso 3D (intersecção entre planos e poliedros convexos).

4.2.1 Intersecção entre Linhas e Polígonos Convexos

Um polígono P no plano é definido por uma lista ordenada de pontos 2D (p_1, p_2, p_3,..., p_n). Sua representação por um grafo planar G define um vértice v_i em G para cada ponto p_i do polígono, e arestas e_k entre dois vértices v_i e v_j se os pontos correspondentes p_i e p_j compartilharem um segmento de linha no polígono. Um grafo planar subdivide o plano em uma região interna e uma região externa ilimitada. Um grafo planar é dito convexo se cada região interna é convexa e o complemento da região externa também é convexo.

O conjunto ativo (conjunto de arestas seccionadas) pode ser obtido pelo cálculo da intersecção entre uma linha de varredura γ contra um polígono convexo P, como é mostrado na Figura 4.3. O segmento resultante ab dentro de P e sobre a linha de varredura γ pode ser representado pelas extremidades a e b. A direção da linha de varredura é dada por um vetor $\vec{s}$ ortogonal à linha γ, e cada posição $\gamma(t)$ corresponde a um diferente instante de tempo. Por simplicidade, assume-se que o movimento da linha de varredura é da esquerda para a direita (uma simples rotação pode ser aplicada se $\vec{s}$ não é horizontal).

Uma mudança no conjunto ativo ocorre quando a linha de varredura passa através de um dos vértices do polígono (o que define um evento). Para encontrar os eventos é necessário projetar cada vértice v_i sobre o vetor $\vec{s}$ (referidos como $\bar{v}_i$). O vértice projetado mais à esquerda (em relação a $\vec{s}$) é chamado de origem (o_s) da linha de varredura (por simplicidade, assume-se que apenas um vértice define o_s). Para cada vértice projetado $\bar{v}_i$, define-se o peso de um vértice projetado $W(\bar{v}_i)$ como a distância Euclidiana entre $\bar{v}_i$ e o_s. A ordem dos vértices projetados (em relação aos seus pesos) dá a ordem dos eventos, como mostra a Figura 4.3.

Nota-se que as arestas do polígono interseccionadas pela linha de varredura não mudam entre os eventos. Estas arestas são chamadas de arestas-suporte, já que elas definem as extremidades ab do segmento de intersecção. Interpolações lineares nas equações paramétricas das arestas-suporte podem ser usadas para obter a e b para cada posição intermediária da linha de varredura. As arestas-suporte são atualizadas somente a cada evento processado.

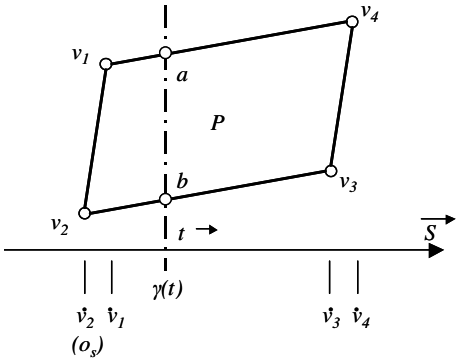


Figura 4.3: Uma linha de varredura movendo-se da esquerda para a direita. A cada instante de tempo t computa-se a sua intersecção com o polígono P. No instante t ilustrado, as arestas suporte são as arestas formadas entre os vértices v_1 e v_4 e os vértices v_2 e v_3.

Para um dado polígono P de n lados e uma direção $\vec{S}$, é possível construir uma estrutura de suporte, um grafo direcionado acíclico valorado (definição "formal" da WSG), que identifica as arestas de suporte e os tempos dos eventos necessários para atualizá-las. Uma WSG $\Gamma(P, \vec{S})$ é definida de uma representação planar G de um polígono P como se segue:

a) para cada vértice v_i em G, criar um nodo n_i em WSG;
b) para cada nodo n_i, definir uma aresta direcionada e_k entre n_i e um nodo n_j (n_i é dito "pai" de n_j) se:
    a. v_i é adjacente a v_j em G;
    b. a linha de varredura passa por v_i antes de passar por v_j ($W(v_i) < W(v_j)$).

A Figura 4.4 mostra um exemplo de WSG, onde a distância vertical entre os nodos da WSG é definida de acordo com seus pesos.

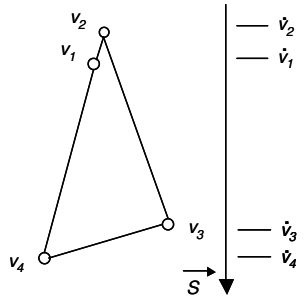


Figura 4.4: WSG codificando a ordem dos vértices atravessados pela linha de varredura γ.

Define-se um "corte" $C(\Gamma, d)$ na WSG Γ em uma distância d, como uma lista de todas as arestas $(e_1, e_2, e_3, ..., e_n)$ na WSG que satisfazem as seguintes condições:

a) e_i é uma aresta na WSG;
b) $W(v_i) \le d \le W(v_j)$, onde n_i e n_j são conectados por e_i na WSG.

Uma busca em profundidade no grafo a partir do nodo correspondente a o_s pode ser usada para calcular um corte na WSG. Uma propriedade importante no corte é que ele contém as arestas-suporte (Figura 4.4). Assim, transforma-se o problema de computar as

arestas-suporte em uma posição t da linha de varredura em um problema de computar um $C(\Gamma, t)$. Mudanças no corte acontecem quando a linha de varredura atravessa um vértice, o que corresponde a alcançar um nodo na WSG. A atualização no corte exige a remoção da aresta que conecta o nodo ao seu pai e a inserção da aresta que conecta o vértice ao seu filho. A Figura 4.5 mostra um instante de tempo da varredura do polígono e o corte correspondente na WSG.

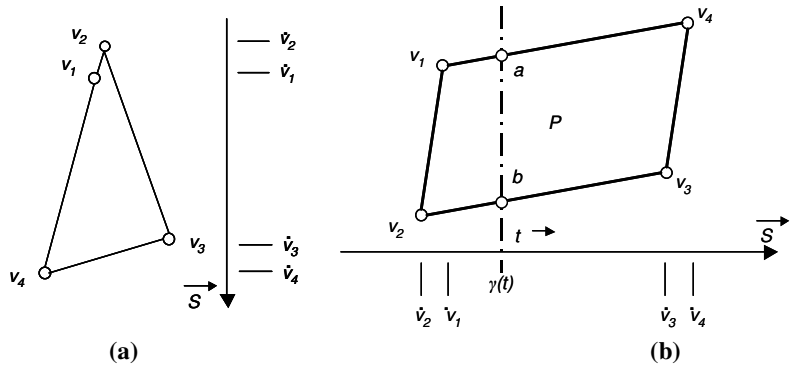


Figura 4.5: Instante de tempo t na varredura da linha γ sobre o polígono P (b). Em (a), observa-se o corte $C(\Gamma, t)$ correspondente na WSG, que contém as arestas-suporte do segmento de interseção entre a linha de varredura γ e o polígono P.

4.2.2 Intersecção Entre Planos e Poliedros Convexos

A noção de uma linha de varredura pode ser estendida em 3D para um plano de varredura γ, definido pela direção dada por um vetor normal $\vec{s}$. A computação da WSG referente a um grafo G (que representa agora o poliedro convexo P) e uma direção de varredura $\vec{s}$ é análoga ao caso 2D. A Figura 4.6 ilustra a WSG obtida para um paralelepípedo ao longo de uma direção de varredura arbitrária.

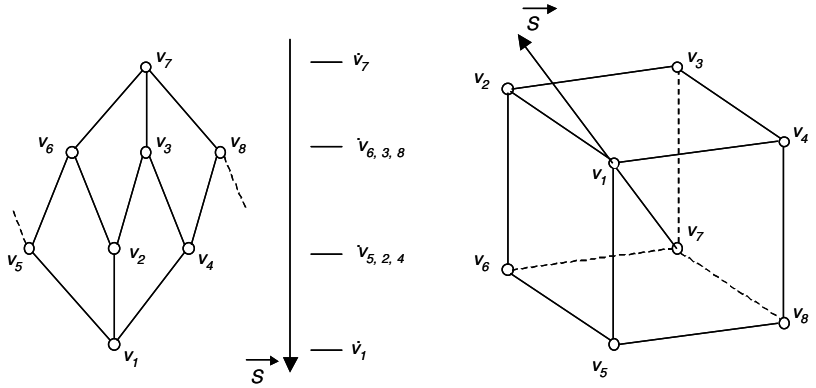


Figura 4.6: WSG resultante da varredura de um cubo por um plano definido por uma direção $\vec{s}$ arbitrária.

Uma dificuldade que surge no caso 3D é que a intersecção entre o plano de varredura e o poliedro corresponde a um polígono. No caso 2D, esta intersecção é um segmento de linha facilmente representado por duas extremidades. Para definir um polígono em 3D, não somente os vértices são necessários, mas também a ordenação

entre eles, seguindo uma convenção de orientação (sentido horário ou anti-horário). Nota-se que cada nodo no caso 2D, exceto o nodo correspondente a o_s, tem apenas um único filho, o que não é verdade no caso 3D. Também se deve considerar que o corte em 3D tem tantas arestas quantos forem os lados no polígono de intersecção, com pelo menos três (triângulo). Assim, algumas regras adicionais que controlam a ordem dos nodos filhos são sugeridas a seguir, para que seja respeitada a ordem de construção do polígono.

Considera-se que $F(v_i) = (v_1, v_2, v_3,..., v_n)$ seja a lista de vértices adjacentes a v_i em G. Esta lista pode ser obtida por uma simples visita ordenada em torno de v_i em G (em sentido horário ou anti-horário). Assume-se que os vértices em $F(v_i)$ são sempre enumerados desta maneira. Tomada uma direção de varredura $\vec{s}$, define-se v_k como o vértice de $F(v_i)$ que tenha o menor peso (ou seja, v_k será pai dos vértices de v_i na WSG). Para que a ordem de construção do polígono seja respeitada, $F(v_i)$ deve ser reordenada na forma $F(v_i) = (v_k, v_{k+1}, v_{k+2},..., v_n,..., v_{k-2}, v_{k-1})$. Este procedimento é repetido para todos os vértices de G, na fase de inicialização da WSG. Se em $F(v_i)$ não houver nenhum vértice v_k onde $W(v_k) < W(v_i)$, a ordem permanece inalterada.

Na Figura 4.7 são ilustradas três diferentes posições do plano de varredura e as WSGs resultantes. O primeiro corte é definido entre as arestas conectando os vértices v_7-v_6, v_7-v_3 e v_7-v_8, usadas para computar os pontos *a*, *b* e *c* do polígono de intersecção. O segundo corte é definido entre as arestas que conectam os vértices v_6-v_5, v_6-v_2, v_3-v_2, v_3-v_4, v_8-v_4 e v_8-v_5, que definem os pontos *d*, *e*, *f*, *g*, *h, i* do hexágono de intersecção.

A restrição de que somente um vértice define a raiz da WSG pode ser eliminada pela simples adição de um nodo "fantasma" ao grafo, que conecte os nodos de menor peso e seja considerado a nova raiz da WSG.

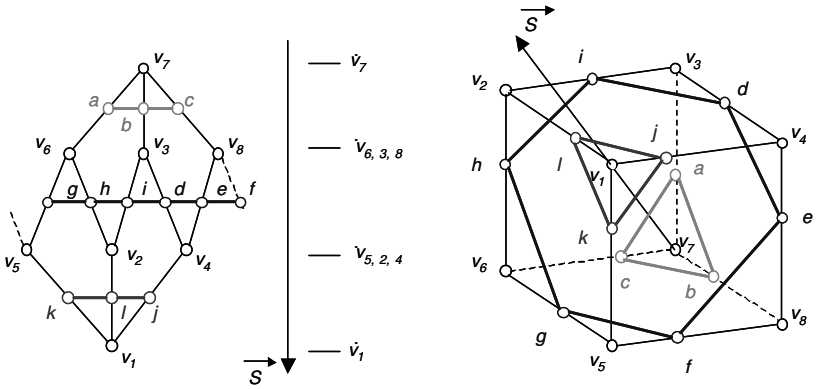


Figura 4.7: Três diferentes posições do plano de varredura e os correspondentes cortes na WSG. Os cortes na WSG são usados para computar os vértices do polígono de intersecção.

Usando a WSG como suporte, o cálculo dos polígonos de amostragem de um volume convexo qualquer é trivial. Definido um ponto de visão, a direção $\vec{s}$ ortogonal ao plano de varredura também é ortogonal ao plano de projeção. No caso do OpenGL, $\vec{s}$ pode ser recuperado diretamente da matriz de visualização da cena. Computada a WSG Γ a partir de $\vec{s}$ e do volume convexo, cada polígono de amostragem é gerado por uma busca em profundidade na WSG, $C(\Gamma, d)$. A distância d é incrementada para a construção de cada polígono.

4.3 Recorte Interativo Usando a WSG

Aqui são discutidas as formas de usar a estrutura de amostragem proposta na Seção 4.2 para construir uma ferramenta interativa de recorte volumétrico. Na primeira parte, é proposta uma modificação no algoritmo da WSG para a reestruturação do poliedro *P* a partir de um corte. A seguir, é sugerida uma forma de controle interativo da definição de um corte no poliedro.

4.3.1 Definição Interativa do Poliedro de Amostragem

No esquema proposto, o VOI é sempre definido por uma região convexa contida dentro do volume original. A especificação desta região convexa pelo usuário segue uma abordagem de "poda". Começando com uma forma convexa básica (um cubo, por exemplo), o usuário pode definir um plano que corta o volume. O recorte do volume convexo por um plano resulta em outro volume convexo, com uma face adicional. O processo pode ser repetido sempre que necessário. Cada plano de recorte é informado pelo usuário pelo seu vetor normal e um ponto no plano, que são controlados usando o mouse. O vetor normal é dado pela direção de um segmento de reta que parte do centro do VOI e termina num ponto indicado pelo usuário com o mouse. O ponto de referência no plano é controlado por um *slider* que especifica a distância entre o plano de recorte e o centro do VOI. A forma de interação proposta pode ser visualizada na Figura 4.8.

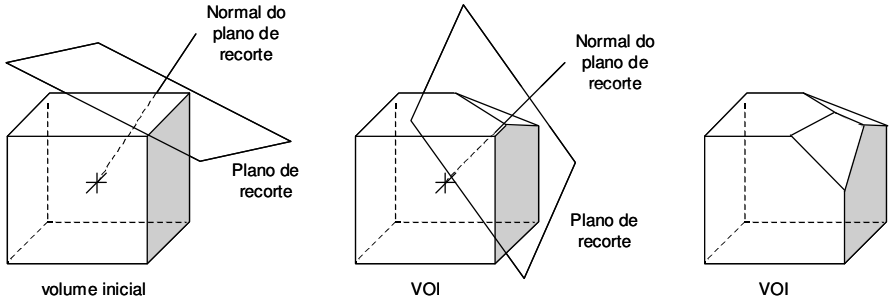


Figura 4.8: Especificação dos planos de recorte para a definição do VOI. O usuário controla a direção normal do plano usando o mouse para indicar a extremidade da linha que parte do centro do VOI. O ponto de referência no plano é controlado por um *slider* que especifica a distância entre o plano de recorte e o centro do VOI.

Na WSG que computa os polígonos de amostragem, os parâmetros informados pelo usuário são usados para calcular o recorte do poliedro. A direção normal definida pelo mouse corresponde a direção de varredura $\vec{s}$. A distância *d* exigida para o cálculo do corte indica a distância definida pelo *slider*.

Os planos de recorte podem ser adicionados em qualquer momento, e VOIs convexos arbitrários podem ser definidos sem que o desempenho da amostragem seja afetado significativamente.

4.3.2 Recorte do Poliedro de Amostragem

Dada uma direção qualquer de varredura $\vec{s}$, o grafo WSG $\Gamma(P, \vec{s})$ pode ser construído para um poliedro *P*. A partir da codificação de *P* em Γ, é possível computar o recorte $C(\Gamma, d)$ de planos (definidos por $\vec{s}$ e uma distância *d* qualquer) contra o

poliedro *P*. Supondo agora que o interesse esteja em recortar o poliedro *P* por um plano qualquer, a informação de $C(\Gamma, d)$ também pode ser usada para a reestruturação de *P*.

Um recorte de um poliedro *P* por um plano pode ser calculado a partir de um corte $C(\Gamma, d)$ na WSG $\Gamma(P, \vec{S})$. Para isto, basta considerar que o plano de recorte é definido por uma direção normal e um ponto no plano. A partir destes parâmetros, é possível calcular uma direção $\vec{S}$ e uma distância *d*, que posicionem $C(\Gamma, d)$ sobre o plano de recorte. Basicamente, a direção $\vec{S}$ corresponde à normal do plano, enquanto que a distância *d* é calculada a partir do centro do poliedro e do ponto sobre o plano de recorte. Assim, para calcular o recorte de *P*, os seguintes passos são executados para cada vértice v_i calculado a partir de $C(\Gamma, d)$:

a) adicionar v_i ao poliedro, onde os vizinhos de v_i são:
   a. os vértices v_j e v_k do poliedro, correspondentes aos nodos n_j e n_k da aresta de $C(\Gamma, d)$ que originou v_i;
   b. os vértices v_{i-1} e v_{i+1} de calculados a partir de $C(\Gamma, d)$;
b) remover da vizinhança de v_i os vértices que estiverem do lado negativo do plano de recorte.

Desta forma, a topologia do poliedro é atualizada em relação ao plano de recorte definido pelo usuário. Observa-se que o passo (b) da atualização da vizinhança de v_i requer que o conjunto de vértices calculado a partir de $C(\Gamma, d)$ seja conhecido antes dos passos de atualização do poliedro. A Figura 4.9 mostra os passos de atualização do VOI descritos acima para o recorte de um paralelepípedo por um plano.

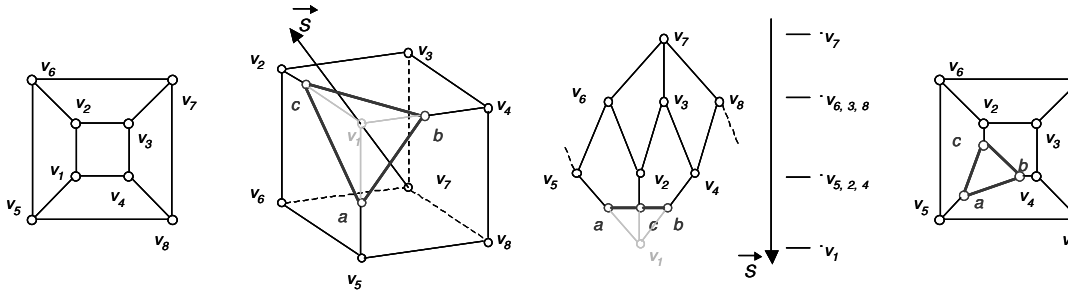


Figura 4.9: Uso da informação contida na WSG para o recorte de um paralelepípedo por um plano. (a) Grafo *G* representando o paralelepípedo; (b) posicionamento do plano de recorte sobre o paralelepípedo (a área em cinza indica a região de recorte); (c) recorte computado na WSG, a partir das informações do plano de recorte e (d) atualização do grafo *G*.

4.4 Borracha Virtual

Ferramentas do tipo "borracha" são comuns em aplicativos de editoração de imagens 2D, com uma interface de controle muito simples: com o mouse, o usuário indica posições sobre a grade da imagem que serão excluídas. No entanto, a dimensão adicional de um volume (profundidade) exige que o usuário seja capaz de descrever coordenadas tridimensionais para apontar regiões dentro do volume. Isto motivou a construção de diversos tipos de ferramentas de recorte mais "sofisticadas", tais como planos e volumes de recorte (PFISTER, 1999), por exemplo. Sua sofisticação vem tanto do número adicional de parâmetros de controle, quanto dos dispositivos de interação

comumente usados (mouse 3D, etc). A despeito de sua eficiência, estas ferramentas falham em expressar a simplicidade do controle das borrachas virtuais usadas na editoração de imagens 2D.

Inspirada por esta observação, nesta seção é feita a proposta de uma borracha virtual que atua diretamente sobre o volume. Para o usuário, no entanto, o controle da ferramenta é feito da mesma forma com que é feito no tratamento de imagens 2D. O ponto de aplicação da ferramenta é indicado com o mouse sobre a grade da imagem (ponto PR). Adicionalmente, o usuário pode indicar um raio de ação (R) para o cursor, que modifica o "tamanho" da borracha. A Figura 4.10 ilustra a forma de atuação da ferramenta.

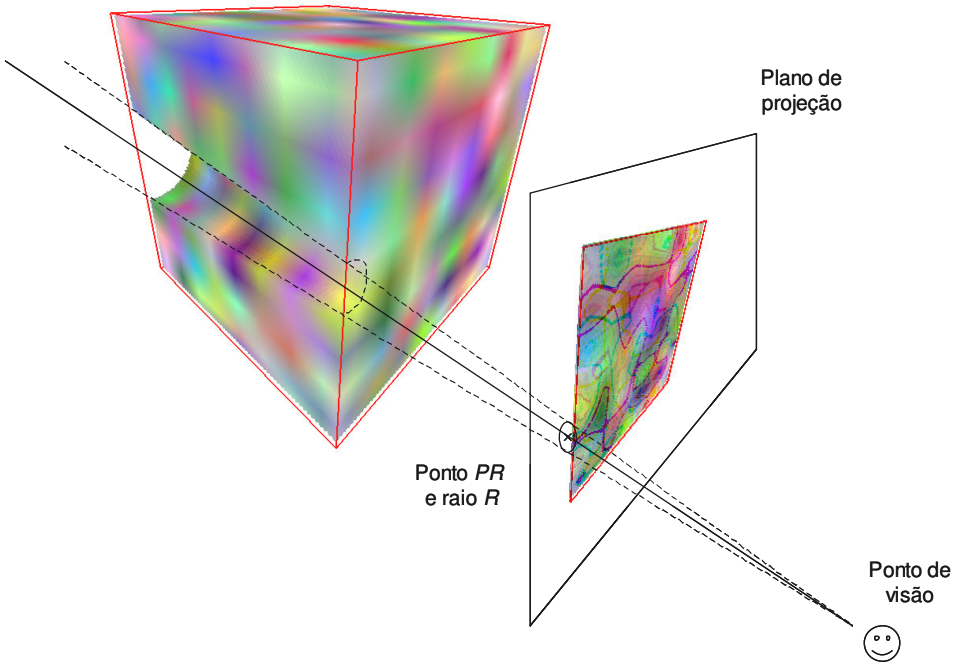


Figura 4.10: Forma de atuação da borracha virtual. O usuário aponta, com o mouse, uma posição sobre a grade da imagem que mostra a projeção do volume. A borracha tem um formato circular, com o centro sobre a posição apontada (*PR*) e raio definido interativamente (*R*). A ferramenta então exclui todas as porções do volume projetivamente correspondentes à área de aplicação da borracha.

Nas seções a seguir, são descritos principais aspectos da implementação da borracha virtual. Na Seção 4.4.1, é discutida a estrutura de dados que suporta a definição VOIs na GPU. A Seção 4.4.2 descreve o procedimento necessário para a atualização da estrutura de dados que contém o VOI. Finalmente, a Seção 4.4.2 descreve a implementação do *shader* que processa a influência do apontamento do usuário sobre cada fragmento do VOI.

4.4.1 Representação do VOI na GPU

Em Weiskopf, Engel e Ertl (2003) os autores discutem uma forma (*voxelized clipping object*) de armazenar o VOI na GPU, que pode ser usada na visualização direta baseada em texturas. Esta abordagem modela o VOI por meio de uma segunda textura

3D. Nesta textura, os *texel*s provêem a visibilidade dos *texel*s correspondentes na textura 3D que representa o volume real. Basicamente, a idéia é discretizar (em *texel*s) e modelar o VOI como uma máscara volumétrica binária. Os *texel*s dentro do VOI recebem o valor 1, e os demais recebem o valor 0. A textura que contém o VOI e a textura real são combinadas pela multiplicação de seus *texel*s. Deste modo, todos os *texel*s que não são visíveis são multiplicados por zero. Durante o *rendering*, os fragmentos dos polígonos de amostragem são mapeados simultaneamente para as posições correspondentes nas duas texturas 3D. Em cada fragmento, os valores de textura são multiplicados para gerar a cor final. Fragmentos que estão fora do VOI têm então valor nulo, e podem ser descartados por testes subseqüentes.

Os autores também discutem uma modificação nesta técnica, onde a textura 3D não é mais interpretada como uma textura binária, mas sim como valores reais representando distâncias com sinal. Cada *texel* armazena a sua distância ao ponto mais próximo do volume de recorte. Como os formatos mais comuns de texturas disponíveis atualmente (no OpenGL) são limitados a valores no intervalo $[0,1]$, a distância é mapeada linearmente para este intervalo. Desta forma, o valor 0.5 representa a superfície do VOI. A vantagem desta abordagem é que ela não sofre os efeitos da discretização binária do VOI (*aliasing*). A Figura 4.11 mostra exemplos das duas abordagem de seleção de fragmentos por meio de uma segunda textura 3D.

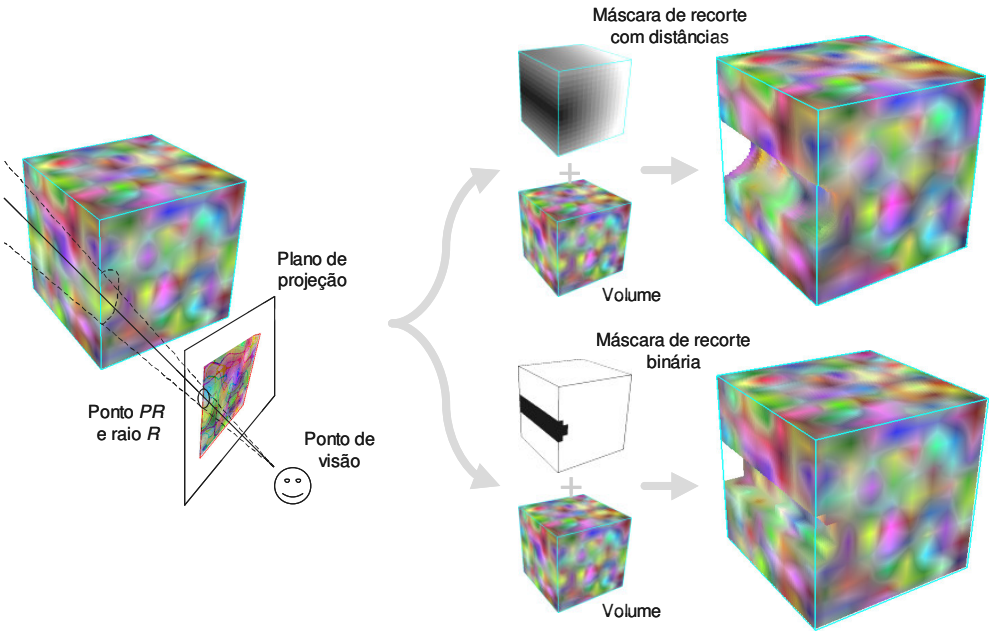


Figura 4.11: Armazenamento do VOI por meio de uma textura 3D adicional. A máscara de recorte definida pelo usuário sobre a tela pode ser armazenada em um volume binário (abaixo) ou em um volume onde cada *texel* armazena a distância ao objeto de recorte (acima). À direita, são vistas as combinações entre o volume e o VOI armazenado segundo as abordagens citadas.

Na implementação das ferramentas propostas nesta seção, o VOI é armazenado na GPU como uma textura 3D, a máscara de recorte. Cada *texel* desta textura armazena a sua distância ao ponto mais próximo do volume de recorte (conjunto de recortes definidos pelo usuário), e é inicialmente visível (inicializado com a maior distância

possível dentro do espaço de textura). A máscara de recorte é enviada pela aplicação à GPU no estágio de inicialização do sistema.

Ainda na inicialização, o sistema associa à máscara de recorte a regra de interpolação tri-linear, por meio de comandos do OpenGL. Como é discutido em Weiskopf, Engel e Ertl (2003), a interpolação tri-linear na consulta dos valores da máscara de recorte permite tanto a suavização do *aliasing* na superfície do VOI discretizado como a utilização de texturas 3D com dimensões diferentes.

Nas duas ferramentas de recorte a atualização da informação contida na máscara de recorte é feita na GPU. O OpenGL, no entanto, não possui procedimentos para gerar uma imagem 3D no *rendering* normal. Isto exige uma adaptação do resultado dos *shaders*, que é descrita na próxima seção.

4.4.2 Atualização do VOI na GPU

A atualização dos valores da máscara de recorte é feita pelos *pixel shaders* das ferramentas de recorte. No entanto, a máscara de recorte é uma textura 3D, enquanto que o *color buffer*, onde é gravado o resultado dos *shaders*, é 2D. Isto exige uma versão "plana" temporária da textura 3D, que possa ser gravada pelos *shaders* no *color buffer*. Contando com esta estrutura adicional, a máscara de recorte pode ser armazenada como uma textura 3D, e o resultado dos *shaders* é copiado para a máscara de recorte após cada atualização. Nesta versão plana, cada pixel deve corresponder a um *texel* da máscara de recorte.

Em Harris *et al.* (2003) os autores propõem que a textura 3D seja armazenada como uma textura 2D, onde as fatias da textura 3D estão dispostas lado a lado. A geometria de amostragem, neste caso, apenas garante que todos os *texel*s serão amostrados (cobre toda a área da textura 2D). Esta abordagem, no entanto, perde a interpolação tri-linear realizada no acesso aos valores da textura 3D. Como é discutida por Weiskopf, Engel e Ertl (2003), esta característica é importante para garantir a suavidade do recorte. Assim, a máscara de recorte deve continuar sendo armazenada como uma textura 3D; a mudança precisa ser feita no resultado dos *shaders*, que deve ser organizado de forma que possa ser copiado de volta para a memória da textura 3D. Nesta dissertação, é proposta uma forma de organização dos *texels* da máscara de recorte em uma versão plana, que pode ser armazenada no *color buffer*. Basicamente, os *texels* são dispostos na textura 2D na mesma ordem em que são enviados para a textura 3D. Esta organização também permite que o *color buffer* seja copiado diretamente para a memória de textura 3D sem que seja necessário reorganizar a disposição dos *texels*. A Figura 4.12 ilustra um exemplo da disposição dos *texels* de uma máscara de recorte de $4 \times 4 \times 4$ em uma textura 2D de 8×8.

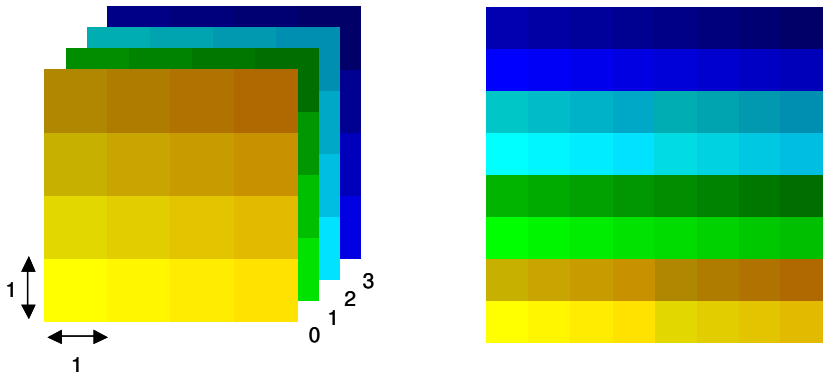


Figura 4.12: Organização das fatias da textura 3D na textura 2D intermediária. (a) máscara de recorte 3D de $4\times4\times4$ (cada "quadrado" das figuras corresponde a um *texel*) e (b) máscara de recorte plana equivalente de 8×8.

A partir da organização ilustrada na Figura 4.12, os *pixel shaders* das ferramentas de recorte precisam apenas processar e armazenar cada fragmento no *frame buffer*. A geometria de amostragem, da mesma forma que em Harris *et al.* (2003), apenas garante que todos os *texels* serão amostrados. É importante ressaltar que esta abordagem é apenas uma adaptação para implementar a atualização da textura 3D por meio do conteúdo do *color buffer*. Outras abordagens, como o uso de *superbuffers*[2] (que permitem o *rendering* direto para texturas 3D), podem ser usadas sem detrimento da implementação das ferramentas.

4.4.3 Implementação na GPU

A borracha virtual é implementada em um *shader* que recebe como entrada a máscara de recorte e as matrizes de visualização da cena, assim como PR e R, definidos interativamente pelo usuário. O objetivo do *shader* é construir uma nova versão plana da máscara de recorte, subtraindo o volume de recorte definido por PR e R do VOI armazenado. A implementação deste *shader* na linguagem Cg pode ser vista no Apêndice A.

Cada fragmento processado pelo *shader* corresponde a um *texel* da máscara de recorte. O valor do fragmento é atualizado segundo a equação

$$F_{t+i} = \min(d, F_t), \tag{4.1}$$

onde F_t representa a distância do fragmento no momento t (armazenada na máscara de recorte) e d é a distância do fragmento ao ponto de aplicação da borracha virtual, calculado como segue:

$$d = \left\| PR - SC \right\|, \tag{4.2}$$

[2] *Superbuffers*

Até o término deste projeto, a extensão ao OpenGL que permite a utilização de *superbuffers* estava prevista, mas não estava disponível para utilização.

onde SC corresponde às coordenadas de tela do fragmento. As coordenadas de tela são obtidas através da transformação da coordenada de textura[3] do fragmento pelas matrizes de visualização. Ao final do processo, cada *texel* da máscara de recorte registra a sua menor distância à máscara definida por PR.

O usuário pode definir o valor de R e modificar interativamente a área de atuação da máscara de recorte. Na implementação, o raio R indica a superfície do volume de recorte (Figura 4.10). Como é visto na Seção 4.4.1, a superfície do volume de recorte é armazenada na máscara de recorte com o valor 0.5, devido a normalização das distâncias calculadas no *shader*. Assim, o *shader* normaliza a distância calculada na Equação 4.2 em relação à R, para que o valor de R (definido em coordenadas de tela) corresponda a superfície do volume de recorte na máscara de recorte. O cálculo de d na Equação 4.2 é então normalizado em relação a R da seguinte maneira:

$$d' = d(\frac{0.5}{R}). \tag{4.3}$$

Desta forma, todos os *texel*s da máscara de recorte que estão contidos no VOI têm valor maior ou igual a 0.5.

4.5 Escavadeira Virtual

O modelo de recorte proposto na borracha virtual permite a exclusão de partes do volume. Em imagens médicas, no entanto, é comum encontrar tecidos que possuem reentrâncias na sua superfície. Estas reentrâncias podem ser uma característica anatômica do tecido ou efeito do contato com tecidos vizinhos. No caso do fígado humano, por exemplo, pode se notar resultado do contato com o coração (Figura 2.1 e Figura 2.3). Ao observar o modelo de interação mostrado na Seção 4.4, nota-se que a borracha virtual não pode ser usada para a remoção de partes do volume que estão em concavidades na superfície do VOI. A aplicação da borracha virtual atravessa todo o volume, enquanto que a intenção é remover algo que está na superfície.

Nesta seção é proposta uma ferramenta de recorte que permite a exclusão de partes da imagem localizadas na superfície do volume, a escavadeira virtual. O controle da ferramenta é feito da mesma forma que na borracha virtual. O ponto de aplicação da ferramenta é indicado com o mouse sobre a grade da imagem (ponto PR). O usuário também pode indicar um raio de ação (R) para o cursor, que modifica a área de atuação da escavadeira. Esta ferramenta, no entanto, age na superfície do VOI. O raio R é usado para calcular a área de atuação em torno do ponto na superfície do VOI correspondente ao ponto PR, definindo um volume de recorte com formato esférico. O efeito sobre o volume é semelhante à colocação de uma "bomba" sobre a sua superfície (WORMS 3D, 2003). A Figura 4.13 ilustra a forma de atuação da ferramenta.

---

[3] coordenadas de textura

Na implementação do sistema proposto nesta dissertação, os espaços de objeto e de textura ocupam o mesmo subespaço de R^3 definido entre (0, 0, 0) e (1, 1, 1). Desta forma, as coordenadas de objeto (usadas para o desenho dos polígonos de amostragem) são equivalentes às coordenadas de textura.

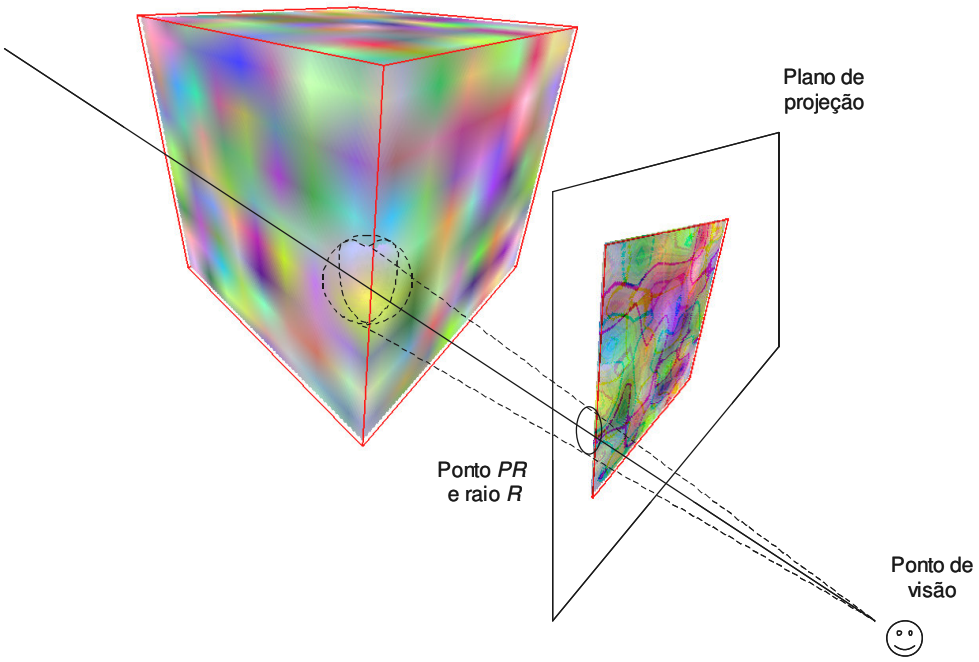


Figura 4.13: Forma de atuação da escavadeira virtual. O usuário aponta, com o mouse, uma posição sobre a grade da imagem (projeção do volume). A ferramenta então marca todos os fragmentos dentro da esfera de recorte virtual como "não visíveis". O centro da esfera é definido pela projeção do ponto indicado pelo usuário na superfície do VOI, enquanto que o seu raio é calculado em relação ao raio definido na tela (R).

A seção seguinte descreve o algoritmo que processa a influência do apontamento do usuário sobre cada fragmento do volume.

4.5.1 Implementação na GPU

A implementação da escavadeira virtual é muito semelhante à da borracha virtual. Ela exige, no entanto, um parâmetro adicional: a informação sobre a posição correspondente ao ponto *PR* na superfície do volume. Esta informação é necessária para que seja possível determinar a partir de que ponto o volume será escavado. Para permitir isto, foi necessário dividir a implementação da escavadeira virtual em dois *pixel shaders*. O primeiro gera uma textura 2D onde cada *texel* armazena a posição (em coordenadas de objeto) do primeiro fragmento visível na superfície do volume, em relação ao plano de projeção. O segundo, da mesma forma que na borracha virtual, gera a nova versão plana da máscara de recorte. A implementação destes *shaders* na linguagem Cg pode ser vista nos Apêndices B e C, respectivamente.

O resultado do primeiro *pixel shader* é uma textura 2D (textura de superfície). Cada *texel* desta textura corresponde a um pixel na janela de visualização (*frame buffer*). O *shader* recebe como parâmetros a geometria de amostragem da textura 3D (VOI definido pela WSG) e a máscara de recorte. A superfície do VOI é determinada pela intersecção entre o VOI definido pela WSG e a máscara de recorte. O valor de cada *texel* da textura de superfície é gerado segundo a equação

$$TP = \begin{cases} (0,0,0), se\ F_t \leq 0.5 \\ TC \end{cases}, \tag{4.4}$$

onde TP representa a cor do *texel* e TC é a coordenada de textura do fragmento. Para cada *texel* do VOI (onde $F_t \geq 0.5$) a posição no espaço de textura (TC) é gravada no *frame buffer*. A posição do *texel* no espaço de textura é armazenada nos canais RGB do pixel. Após o processamento do *shader*, o *frame buffer* é copiado para a memória de textura da textura de superfície. Desta forma, a textura de superfície corresponde à imagem construída pelo *shader* de visualização do volume (Figura 4.2), onde a cor de cada pixel é a coordenada de textura do último fragmento que foi projetado sobre ele. A partir desta textura, é possível recuperar a posição no espaço de textura do *texel* correspondente a PR (TP_{PR}).

A implementação da escavadeira ainda tem outra diferença em relação à da borracha virtual. Como pode ser visto na Equação 4.2, na borracha virtual o cálculo da distância da projeção do *texel* ao ponto PR é feito no espaço de tela. Na escavadeira virtual são calculadas as distâncias entre TP_{PR} e cada *texel* na máscara de recorte no segundo *pixel shader*. As posições destes *texel*s, no entanto, estão no espaço de textura. A transformação projetiva não garante uma relação entre as distâncias calculadas nestes espaços, como é ilustrado na Figura 4.14.

A solução proposta aqui é calcular a distância entre os *texel*s no espaço de tela. Para isto, o vetor formado entre TP_{PR} e a coordenada TC do *texel* é rotacionado até estar paralelo ao plano de projeção (estes vetores podem ser observados na Figura 4.15). Isto é obtido segundo a equação

$$\vec{v}_{TP_{PR}-TC} = \left\| (TP_{PR} - TC) \right\| \cdot \vec{v}_{cop}, \tag{4.5}$$

onde $\vec{v}_{cop}$ é um vetor normalizado e coplanar ao plano de projeção. No OpenGL este vetor pode ser obtido a partir da matriz de modelagem e visualização (eixos de rotação *x* e *y*). O vetor $\vec{v}_{TP_{PR}-TC}$ é então somado a TC, do qual se obtém uma coordenada no espaço de textura TC'. Este procedimento é ilustrado na Figura 4.15. A partir de TC', o procedimento continua como descrito nas equações 4.1, 4.2 e 4.3, considerando TC' como a coordenada de textura do fragmento.

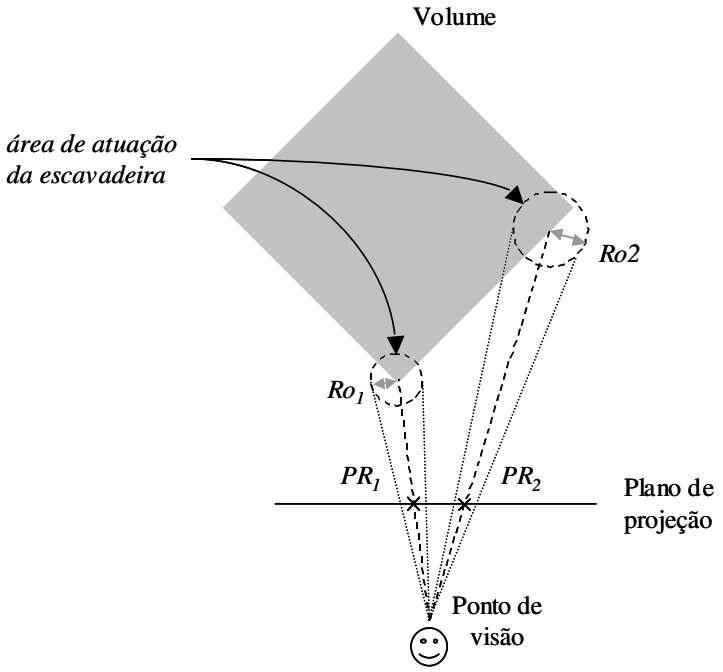


Figura 4.14: Raios de ação no espaço de objeto (Ro_i) resultantes da aplicação da ferramenta em duas posições da tela (PR_i), onde o mesmo raio R é utilizado.

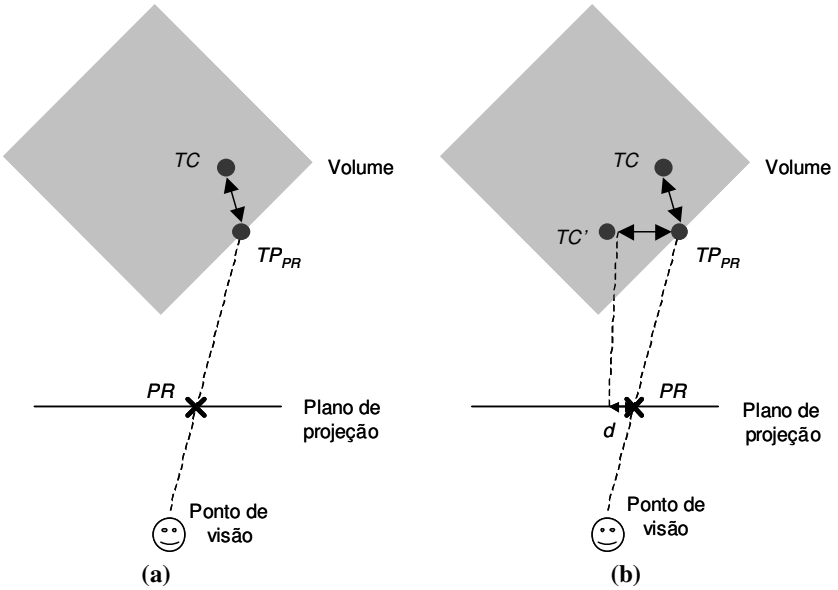


Figura 4.15: Cálculo da distância entre os *texel*s no espaço de tela no segundo *pixel shader*. Basicamente, a distância entre TP_{PR} e TC (a) é transformada para o espaço de tela (b), a partir do qual se procede da mesma forma do que na borracha virtual.

4.6 Conclusão

As ferramentas de recorte descritas neste capítulo foram desenvolvidas de acordo com necessidades identificadas na interação com imagens médicas 3D. A cada

ferramenta proposta, o usuário pode ter uma melhor definição do VOI dentro da imagem 3D. A ferramenta de recorte na amostragem é capaz de definir apenas uma versão grosseira do VOI. No entanto, ela reduz o número de fragmentos de amostragem a cada recorte, aumentando o desempenho do fluxo de *rendering*. A borracha e a escavadeira virtuais permitem ao usuário duas maneiras de dar o "acabamento" ao VOI, permitindo a definição não apenas de VOIs bastante detalhados, como também côncavos.

5 APLICAÇÃO DAS FERRAMENTAS DE VISUALIZAÇÃO E RECORTE INTERATIVOS EM IMAGENS MÉDICAS

5.1 Introdução à Aplicação Médica

O sistema de visualização interativa proposto no Capítulo 4 oferece ao usuário ferramentas para inspeção de imagens volumétricas. Estas ferramentas podem ser aplicadas para a remoção de partes não-relevantes do volume, e proporcionar uma visão mais clara do objeto de interesse. Esta capacidade é útil na visualização de imagens médicas que contenham partes complexas da anatomia humana. Neste Capítulo, são mostrados experimentos de aplicação do sistema proposto na visualização da anatomia de um órgão com anatomia muito complexa, o fígado humano.

A complexidade do sistema vascular do fígado torna qualquer intervenção cirúrgica neste órgão uma operação de grande risco. A Figura 5.1 mostra um esquema da topologia vascular do fígado.

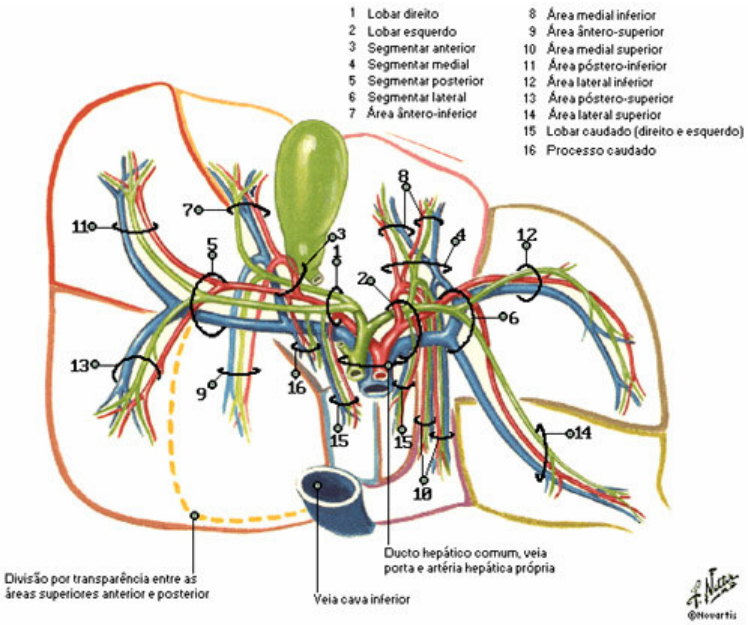


Figura 5.1: Distribuição dos vasos sanguíneos no interior do fígado humano, em vista posterior (NETTER, 1999).

Como pode se visto na Figura 5.1, o sistema vascular do fígado possui muitos vasos sanguíneos importantes e com várias ramificações. Esta topologia também varia significativamente em cada pessoa, o que dificulta ainda mais a criação de um modelo

mental da anatomia deste órgão. Tradicionalmente, o cirurgião em treinamento para este tipo de operação constrói este "modelo mental" através do estudo de Atlas de anatomia humana (ilustrações 2D) e imagens de TC e RM. Recentemente, o uso de Atlas em meio digital tem se difundido no meio acadêmico, permitindo a exploração de modelos 3D virtuais do corpo humano. No entanto, mesmo nestes aplicativos as opções do usuário para exploração interativa do modelo ainda são restritas.

A exploração interativa de modelos anatômicos 3D normalmente restringe as opções do usuário ao controle de câmera e seleção de regiões pré-segmentadas da anatomia. A Figura 5.2 mostra exemplos de pacotes comumente utilizados para esta aplicação. Estes pacotes oferecem modelos ilustrativos (2D e 3D) de partes da anatomia humana. O usuário pode inspecionar os modelos a partir de uma câmera virtual, que se desloca interativamente pelo espaço no qual o objeto é mostrado. Embora tenha liberdade na escolha da posição de visualização, o usuário tem poucas opções de interação com os modelos visualizados. Normalmente, estas opções são restritas à seleção de partes pré-segmentadas do modelo. A restrição de qualquer ação do usuário na inspeção de um modelo anatômico tridimensional pode dificultar sua compreensão da estrutura do objeto de interesse, ou de sua relação com objetos vizinhos.

Estas restrições são menores (mas ainda significativas) nas estações de trabalho de TCs e RMs. Nestas estações, também é possível reconstruir modelos 3D a partir das imagens adquiridas. No entanto, a interação com os modelos reconstruídos ainda é restrita. Além disto, estas estações têm um alto custo se comparadas a computadores pessoais.

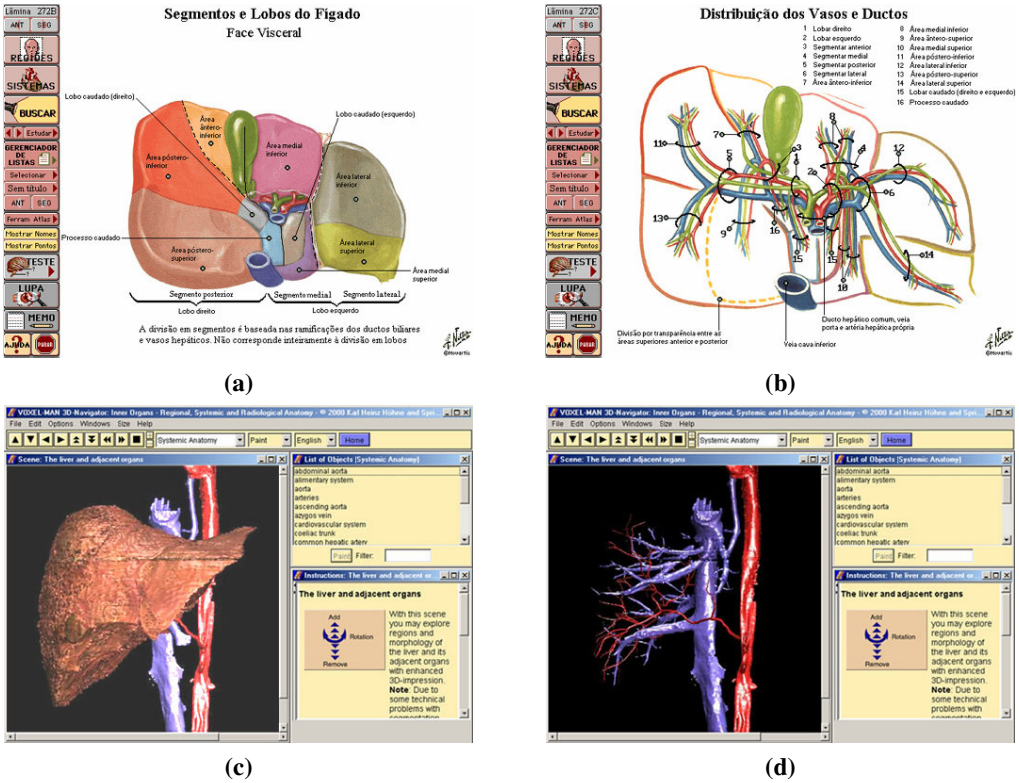


Figura 5.2: Exemplos de pacotes de visualização estruturas anatômicas comumente utilizados para o treinamento médico. (a) e (b): Atlas Interativo de Anatomia Humana Netter (1999); (c) e (d): VOXEL-MAN *3D Navigator: Inner Organs* (2000).

Nos experimentos conduzidos com imagens médicas neste Capítulo, o sistema de visualização proposto no Capítulo 4 é usado para a inspeção da anatomia do fígado humano. Através das ferramentas de recorte, o usuário remove as estruturas que circundam o órgão e visualiza seu interior. São avaliados o desempenho computacional e o resultado visual do sistema proposto. Na Seção 5.2 são discutidos os materiais utilizados nos experimentos. Na seção seguinte, são descritos os procedimentos usados na condução dos experimentos. Finalmente, são mostrados e discutidos os resultados dos experimentos nas seções 5.4 e 5.5, respectivamente.

5.2 Materiais

São usados dois volumes na condução dos experimentos. O primeiro volume é uma textura 3D construída proceduralmente, onde cada *texel* é opaco e tem uma cor atribuída aleatoriamente. Esta textura tem dimensões de $8\times8\times8$ e três canais de cor por *texel*. O segundo volume é um TC abdominal contrastado (angiograma). Neste tipo de aquisição é injetado um líquido de contraste na corrente sanguínea. Este líquido tem uma alta densidade radiológica e se espalha pelo sistema circulatório, realçando os vasos sanguíneos em imagens adquiridas por raios-X. As 128 imagens que formam o volume foram adquiridas em um aparelho Toshiba® Asteion Helicoidal, na Santa Casa de Porto Alegre, e armazenadas como fatias axiais no formato DICOM (*Digital Imaging and Communications in Medicine*). Cada fatia tem resolução de 512×512 pixels, onde cada pixel corresponde a um volume de $0.625\times0.625\times3$ milímetros cúbicos.

No segundo volume usado nos experimentos, cada voxel armazena um valor de densidade radiológica. Para que fosse possível visualizá-lo por meio da técnica de mapeamento de texturas, foi construída uma função de transferência que mapeia valores de densidade radiológica para valores de cor (RGB) e opacidade. A função de transferência usada nestas imagens é uma combinação da *window*[4] definida pelo operador da estação do tomógrafo com duas cores, feita no *shader* de visualização. A *window* utilizada neste volume tem nível central em 400 HU (coeficiente de densidade radiológica) e largura de 2000 HU. O resultado da aplicação da *window* define a opacidade de cada voxel. No *shader* de visualização, a cor vermelha é atribuída a fragmentos com opacidade inferior a 0.5; caso contrário, o fragmento recebe a cor branca. Esta função de transferência resulta em uma imagem onde os tecidos de baixa densidade radiológica (tecidos moles) têm cor vermelha e tecidos de alta densidade radiológica (ossos e líquido de contraste) têm cor branca.

Os experimentos para medição do desempenho do sistema foram conduzidos em duas CPUs. A primeira CPU é processador Intel® Pentium IV de 2 *gigahertz* e 512 *megabytes* de memória primária. A segunda é um processador Intel® Pentium III de 700 *megahertz* com 64 *megabytes* de memória primária. A GPU utilizada em todos experimentos é uma ATI® Radeon 9700 com 128 MB de memória, instalada na primeira CPU.

O sistema foi desenvolvido para plataforma Windows 2000 Professional, utilizando o compilador Visual C++ .NET, em modo *debug* (sem otimizações no código feitas pelo

[4] *window*
Função de transferência linear em forma de rampa, que possui um nível central (nível de intensidade que corresponde ao nível médio da rampa) e uma largura (que corresponde ao intervalo de intensidades abrangido pela rampa).

compilador). As texturas 3D são armazenadas com precisão de 16 bits por canal de cor, enquanto que o *frame buffer* tem 12 bits de precisão por canal de cor.

5.3 Métodos

A condução dos experimentos é dividida em duas partes. A primeira parte trata da visualização do resultado da aplicação das ferramentas, e é descrita na Seção 5.3.1. A segunda parte descreve como foram feitos os experimentos para a medição do desempenho computacional do sistema, e é descrita na Seção 5.4.2.

Em todos os experimentos, a janela de visualização usada tem resolução inicial de 512×512. Esta janela determina o tamanho da textura de superfície (descrita na Seção 4.5) e do *frame buffer*, e tem influência sobre o desempenho do *shader* de visualização (o tamanho do *frame buffer* é um dos fatores que pode aumentar o número de fragmentos de uma cena). O usuário interage com o volume através do mouse, que fixa os parâmetros de visualização (posição da câmera) e das ferramentas de interação.

5.3.1 Aplicação das Ferramentas

A aplicação das ferramentas de recorte é feita sobre os dois volumes. No primeiro volume são feitos recortes aleatórios sobre a imagem, utilizando duas resoluções para a máscara de recorte. São usadas máscaras de recorte com dimensões de $16 \times 16 \times 16$ e $64 \times 64 \times 64$. No segundo volume os recortes são aplicados para remover as estruturas anatômicas que circundam o fígado no volume. As aplicações das três ferramentas de recorte propostas são feitas em seqüência sobre a mesma imagem, ilustrando o procedimento de seleção do VOI na imagem volumétrica.

5.3.2 Desempenho das Ferramentas

As medições do tempo de processamento das ferramentas são feitas com um contador de alta freqüência, através do procedimento *QueryPerformanceCounter(...)*. Como é discutido em Abrash (2001), este procedimento permite medir intervalos de tempo com definição de até $5.04031e-6$ segundos. O segundo volume de visualização é usado em todos os experimentos para medição do desempenho.

O desempenho da ferramenta de recorte no estágio de atualização é medido nas duas partes do procedimento. A primeira parte é o recorte do VOI por um plano (que exige a reestruturação da topologia do poliedro de amostragem). A segunda é a construção e envio dos polígonos de amostragem à GPU. Os dois procedimentos são realizados na segunda CPU e são medidos em segundos (intervalo de tempo necessário para sua computação).

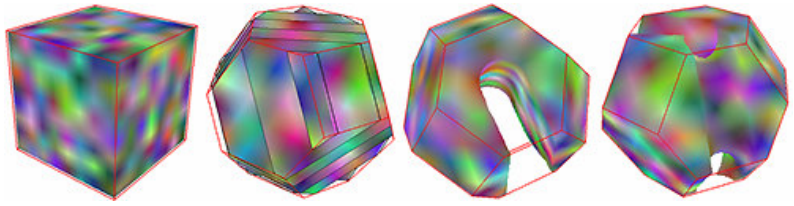
O desempenho das ferramentas de recorte no estágio de desenho corresponde ao procedimento de atualização da máscara de recorte. O desempenho deste procedimento é medido em segundos, desde o momento em que o usuário faz o apontamento na tela até o momento em que o fluxo de visualização "normal" é restabelecido, ou seja, até o momento em que possa ser visualizada a alteração do usuário na imagem final. A influência das dimensões da máscara de recorte no desempenho deste procedimento também são testadas nos experimentos, para máscaras de recorte com tamanhos de $16 \times 16 \times 16$ e $64 \times 64 \times 64$. O experimento para a medição do desempenho destas

ferramentas dispara 100 recortes seguidos no centro da tela (ponto *PR* nas coordenadas (256,256)), sem alterar o raio de ação do cursor, e apresenta o tempo médio decorrido em cada recorte.

5.4 Resultados

5.4.1 Aplicação das Ferramentas

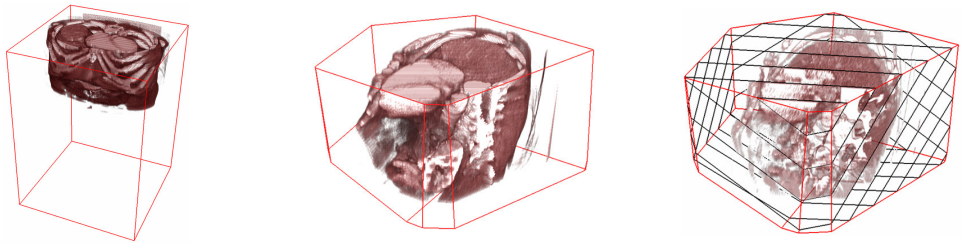
Na Figura 5.3 é ilustrada a aplicação das ferramentas de recorte sobre o primeiro volume. Na Figura 5.3 (b) é visto o resultado da aplicação de vários planos de recorte ao volume. Neste momento, o VOI está definido pelo poliedro de amostragem. Nos itens (c) e (d) da Figura 5.3, são vistos os recortes realizados pela borracha e pela escavadeira virtuais. O VOI, a partir do uso destas ferramentas, é a intersecção entre o poliedro de amostragem e o VOI armazenado na máscara de recorte.



(a) **(b)** **(c)** **(d)**

Figura 5.3: Aplicações das ferramentas de recorte sobre uma textura construída proceduralmente. (a) volume inicial; (b) recortes sucessivos no estágio de amostragem (o número de planos de amostragem foi reduzido para ilustrar o recorte realizado pela WSG); (c) volumes resultantes da aplicação da borracha e (d) da escavadeira virtuais.

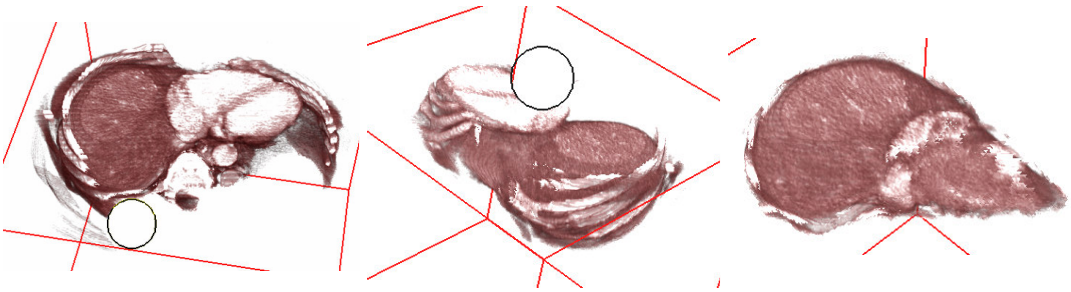
Na Figura 5.4 é ilustrada a aplicação da ferramenta de recorte na atualização sobre o segundo volume. Na Figura 5.4 (a), é ilustrado volume inicial (paralelepípedo), onde pode ser visualizado o espaço amostrado que não contribui na formação da imagem final. Na Figura 5.4 (b), após sucessivos recortes com a ferramenta, a área amostrada se resume ao contorno do fígado, ainda grosseiro. Na Figura 5.4 (c) são visualizados os contornos dos polígonos de amostragem da textura 3D recortados contra o poliedro de amostragem pela WSG.



(a) **(b)** **(c)**

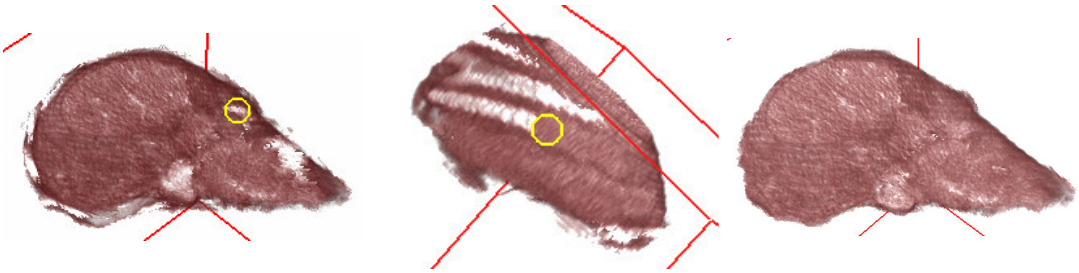
Figura 5.4: Recorte do VOI no estágio de atualização. (a) imagem e volume inicial (representado por suas linhas de contorno); (b) resultado de sucessivos recortes, definindo um VOI, ainda que bastante grosseiro; (c) polígonos de amostragem da textura (representados pelo seu contorno), ilustrando o recorte realizado pela WSG (no Apêndice D pode ser vista uma versão colorida desta figura, e no endereço www.inf.ufrgs.br/~cadietrich/dissertacao pode ser encontrados vídeos que mostram a aplicação desta ferramenta em tempo real).

Na Figura 5.5 e na Figura 5.6 são ilustradas as aplicações das ferramentas de recorte no estágio de desenho. Na Figura 5.5 (a, b) são ilustrados a aplicação da borracha virtual e o volume resultante (c). Da mesma forma, na Figura 5.6 é visualizada a aplicação da escavadeira virtual. O cursor da interface das ferramentas poder ser visualizado nas duas figuras (círculo de cor preta na Figura 5.5 e amarela na Figura 5.6).



(a) **(b)** **(c)**

Figura 5.5: Aplicação da borracha virtual (a, b) em uma visualização 3D de um TC abdominal. O volume resultante mostrado em (c) corresponde ao parênquima do fígado; neste volume ainda podem ser observados artefatos (partes de costelas e tecidos contrastados) que não podem ser alcançados pela aplicação desta ferramenta, por estarem em reentrâncias do parênquima (no Apêndice D pode ser vista uma versão colorida desta figura, e no endereço www.inf.ufrgs.br/~cadietrich/dissertacao pode ser encontrados vídeos que mostram a aplicação desta ferramenta em tempo real).

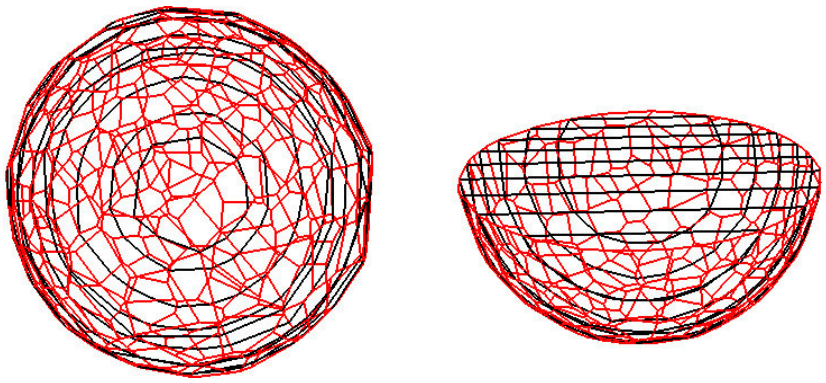


(a) **(b)** **(c)**

Figura 5.6: Aplicação da escavadeira virtual (a, b) sobre o VOI mostrado na Figura 5.5 (c) e o volume resultante (c) (no Apêndice D pode ser vista uma versão colorida desta figura, e no endereço www.inf.ufrgs.br/~cadietrich/dissertacao pode ser encontrados vídeos que mostram a aplicação desta ferramenta em tempo real).

5.4.2 Desempenho das Ferramentas

O tempo de recorte de um poliedro por um plano foi medido para dois poliedros com topologias diferentes. O tempo medido no recorte de um poliedro de 8 vértices é de $1.54545e^{-4}$ segundos. O recorte aplicado no teste eliminou um vértice do poliedro original, adicionando outros três (primeiro recorte mostrado na Figura 4.8). O segundo recorte foi aplicado ao poliedro da Figura 5.7, de 578 vértices. O poliedro ilustrado foi construído a partir de uma série de recortes eqüidistantes ao centro do poliedro original (paralelepípedo). O recorte seccionou o poliedro ilustrado na Figura 5.7 (a) em duas partes, eliminando 184 vértices do poliedro. O tempo de recorte medido foi de $1.33819e^{-3}$ segundos.



(a) (b)

Figura 5.7: Poliedro usado no teste de desempenho do recorte no estágio de atualização. (a) Poliedro (visto por suas arestas na cor cinza) e o contorno dos polígonos de amostragem (na cor preta) recortados pela WSG; (b) Poliedro recortado e polígonos de amostragem.

O desempenho da WSG no recorte e envio dos polígonos de amostragem é visto no gráfico da Figura 5.8. O gráfico relaciona o espaçamento entre os planos de amostragem dentro de um paralelepípedo e o intervalo de tempo decorrido no procedimento.

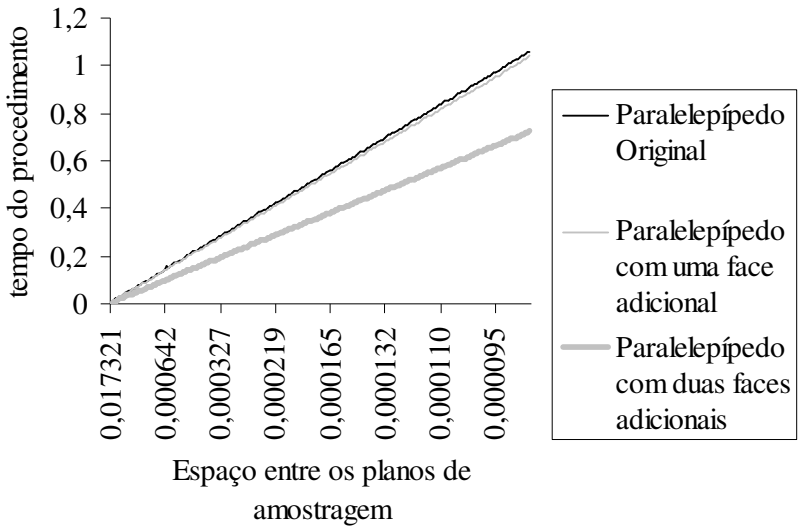


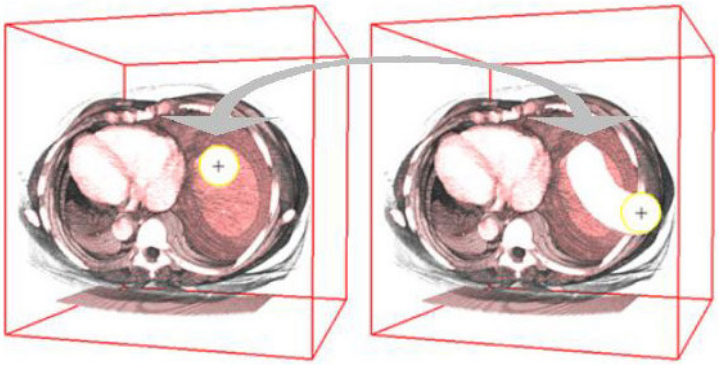
Figura 5.8: Desempenho da WSG na amostragem de diferentes poliedros. Observa-se que a aplicação dos recortes (que resultam em faces adicionais) pode melhorar o desempenho do algoritmo.

O tempo de atualização médio da máscara de recorte de dimensões $64\times64\times64$ na borracha virtual é de 0.096 segundos. O tempo medido na máscara de recorte de $16\times16\times16$ é 0.007 segundos. Da mesma forma, os tempos médios de aplicação da escavadeira virtual são de 0.115 e 0.046 segundos.

5.5 Discussão

5.5.1 Aplicação das Ferramentas

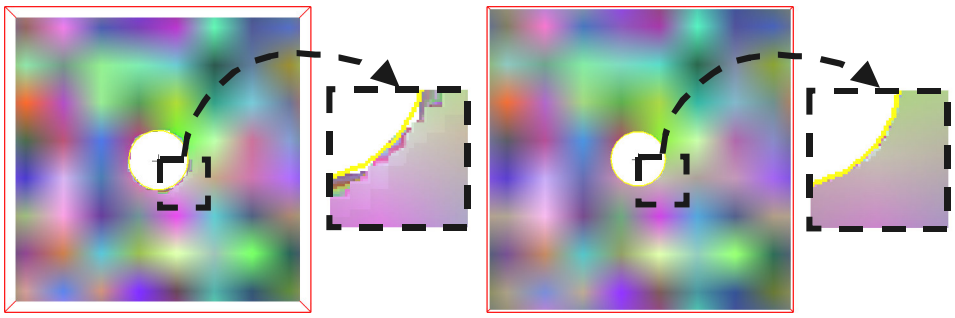
A limitação na precisão do *frame buffer* e da memória de textura têm influência no resultado do recorte processado na GPU e na qualidade da imagem final. Como é discutido no Capítulo 4, o procedimento destas ferramentas atualiza a máscara de recorte a cada nova inserção do usuário. O resultado do *rendering* dos *shaders* é armazenado no *frame buffer*, copiado para a CPU e novamente copiado para a memória de textura. A textura de superfície, atualizada pelo *shader* da escavadeira virtual, é copiada diretamente do *frame buffer* para a memória de textura. Estas cópias afetam os valores armazenados nas texturas e a consistência do resultado das ferramentas, devido à precisão limitada do *frame buffer* em relação à memória de textura e ao resultado calculado nos *shaders*. O erro é maior no caso da máscara de recorte: valores calculados que sejam próximos ao limiar de recorte, ao sofrerem qualquer variação no seu conteúdo, alteram o aspecto da superfície do VOI. Esta variação pode ser vista na Figura 5.9. A Figura 5.9 (a) ilustra o primeiro recorte realizado pelo usuário com a borracha virtual. Na mesma posição da imagem, na Figura 5.9 (b), pode ser vista a variação da área onde a ferramenta foi aplicada inicialmente, devido as atualizações do *buffer* da máscara de recorte feitas ao longo da aplicação da borracha.



(a) **(b)**

Figura 5.9: Erro na consistência do resultado da aplicação da borracha virtual (a). A perda de precisão ilustrada em (b) (“pico” no círculo removido) é resultado das cópias realizadas durante as constantes atualizações da máscara de recorte, entre *buffers* de precisões diferentes.

O uso de diferentes resoluções para a máscara de recorte também influencia na consistência do recorte feito com as ferramentas de recorte na GPU. Enquanto que baixas resoluções resultam em um tempo de atualização menor, altas resoluções resultam em um recorte mais consistente. A Figura 5.10 mostra em detalhes o resultado do uso de duas resoluções diferentes para a máscara de recorte. A limitação no tamanho da memória de textura disponível nas GPUs atuais ainda exige que sejam usadas baixas resoluções para o armazenamento da máscara de recorte.



(a) (b)

Figura 5.10: Aplicação da borracha virtual no primeiro volume, usando máscaras de recorte com tamanhos diferentes. A baixa resolução ($16\times16\times16$) da textura usada em (a) produz resultados piores do que a alta resolução ($64\times64\times64$) usada em (b).

5.5.2 Desempenho das Ferramentas

O desempenho do recorte por planos suportado pela WSG depende da complexidade do poliedro de amostragem. A complexidade do poliedro está relacionada ao seu número de vértices e faces. No entanto, mesmo para poliedros muito complexos (como o ilustrado na Figura 5.7 (a)), o tempo de resposta do sistema permite mais de 700 atualizações por segundo, suficiente para a interação em tempo real.

O desempenho do procedimento de construção dos polígonos de amostragem depende do poliedro que define o VOI. Duas características do poliedro são importantes na medida do desempenho da WSG neste procedimento: sua complexidade e o seu tamanho. A complexidade do poliedro influencia na complexidade do grafo da WSG. Com um grafo mais complexo, o custo do caminhamento para a construção de cada polígono de amostragem é maior. Já o tamanho do poliedro influencia no número de polígonos de amostragem que serão construídos. Quanto mais polígonos forem necessários, mais vezes o grafo da WSG será percorrido, aumentando o custo do procedimento. O resultado dos experimentos para a medição do desempenho da WSG é ilustrado no gráfico da Figura 5.8. O gráfico relaciona o espaçamento entre os planos de amostragem e o intervalo de tempo decorrido no procedimento. Observando este gráfico, nota-se que o custo do procedimento depende muito mais do número de planos de amostragem do que da complexidade do poliedro: enquanto que o custo sempre aumenta para um número maior de planos, o paralelepípedo com duas faces adicionais tem desempenho superior ao paralelepípedo original. Isto se deve à redução de tamanho do poliedro de amostragem no segundo recorte: dentro de um poliedro menor, com o mesmo espaçamento entre os planos, menos polígonos de amostragem são construídos.

A atualização da máscara de recorte depende de vários aspectos da CPU e da GPU. O cálculo do novo *buffer* da textura, isoladamente, é muito rápido, pois corresponde apenas ao *rendering* de 262.144 fragmentos (para a máscara de recorte de $64\times64\times64$). Neste caso, a cópia dos buffers entre a GPU e a CPU consome a maior parte do tempo. Estas cópias dependem da velocidade do barramento onde a GPU é instalada.

O sistema de visualização proposto neste estudo adiciona um custo significativo ao fluxo de visualização direta baseada em texturas. As ferramentas de recorte processadas no estágio de desenho exigem um acesso à máscara de recorte para cada fragmento processado pelo *shader* de visualização. Basicamente, cada fragmento que vai compor a imagem final deve consultar se é visível ou não. O número de fragmentos, por sua vez,

depende do numero de polígonos de amostragem, da área que cada polígono de amostragem ocupa no *frame buffer* e do tamanho do *frame buffer* (em pixels). Assim, é difícil aproximar o número de fragmentos que compõem uma imagem visualizada pela técnica de mapeamento de texturas. Aqui, a análise do desempenho do *shader* de visualização se limita a considerar que, para cada fragmento do volume real processado, um acesso adicional a uma textura 3D deve ser feito, e o desempenho no acesso depende apenas do desempenho da GPU.

6 DISCUSSÃO E CONCLUSÕES

Nesta dissertação é proposto um sistema de visualização volumétrica interativa baseada no mapeamento de texturas. São discutidas a implementação e a integração ao fluxo de visualização volumétrica de três ferramentas de recorte volumétrico interativo.

A primeira ferramenta de recorte atua no estágio de atualização da geometria de amostragem da textura 3D. A sua proposta inclui um algoritmo eficiente de recorte de planos contra poliedros convexos (WSG) e a sugestão de uma interface de controle para a ferramenta. A WSG serve de suporte ao recorte processado na CPU e também como estrutura para construção dos planos de amostragem da textura 3D. Como estrutura de suporte ao recorte por planos binários é difícil enumerar outra estrutura de dados com funcionamento semelhante. Neste ponto poderiam ser consideradas estruturas de particionamento do espaço, como *BSP-trees* e *kD-trees*. No entanto, o custo computacional do cálculo do recorte de planos contra estas estruturas é maior. Estas estruturas armazenam todos os planos de recorte, enquanto que a WSG guarda apenas o VOI resultante dos recortes. Desta forma, enquanto a WSG gera apenas polígonos dentro do seu domínio, estas estruturas testam, para cada plano, o recorte contra todo o conjunto de planos armazenados.

As duas ferramentas de recorte processadas no estágio de desenho exploram o poder computacional disponível nas GPUs atuais. Nestas ferramentas, foram propostas duas interfaces que permitem interagir com o volume a partir do plano de projeção (imagem final), baseadas na técnica de armazenamento de informação de visibilidade sugerida por Weiskopf, Engel e Ertl (WEISKOPF; ENGEL; ERTL, 2003). Estas interfaces permitem inspecionar o volume com a mesma simplicidade encontrada em aplicativos comuns de editoração de imagens 2D. O usuário, a partir de dispositivos de interação comuns (mouse), é capaz de interagir com o volume de forma eficiente.

O uso da GPU para a implementação do sistema restringe o seu desempenho apenas ao poder computacional das GPUs atuais. Nos últimos anos, é possível perceber que as GPUs avançam em um ritmo muito rápido (dobram a sua capacidade de processamento a cada ano), o que encoraja o uso destes dispositivos. No entanto, a geração atual de GPUs ainda tem uma série de restrições na precisão das operações nos *shaders* e no armazenamento de dados na memória de textura. O impacto destas limitações sobre o sistema de visualização proposto são discutidas na Seção 6.1.

Ao longo deste estudo foi possível observar que o conjunto de técnicas e estruturas propostas para a construção do sistema pode ser facilmente adaptado para a criação de novas ferramentas de interação com volumes. Duas propostas implementadas são discutidas na Seção 6.2.

Analisando cada uma em separado, as ferramentas propostas nesta dissertação são muito simples. A WSG é capaz de armazenar um poliedro convexo, suportando o recorte de planos contra este poliedro. O resultado do recorte pode ser usado tanto para

a construção de um polígono quanto para o recorte do próprio poliedro. As borracha e escavadeira virtuais são interfaces de atualização da máscara de recorte, que exploram o poder computacional da GPU para realizar esta tarefa eficientemente. Sobre este ponto de vista, a contribuição deste projeto não está em uma ferramenta em separado, mas sim no conjunto formado por elas. As estruturas propostas nesta dissertação exploram todos os estágios do *rendering* volumétrico baseado em texturas, desde a amostragem até o acabamento. Juntas, elas formam um sistema de *rendering* volumétrico, cuja implementação já provou ser eficiente o bastante até mesmo para permitir a proposta de novas ferramentas.

6.1 Limitações

As principais desvantagens da visualização direta baseada em texturas são as limitações no tamanho da memória de textura e na precisão do armazenamento dos dados na GPU (ZHOU, 2003). A memória de textura limitada requer que grandes volumes de dados sejam divididos em blocos menores, e ocorra uma troca constante entre a memória da CPU e da GPU. A falta de precisão no armazenamento de dados na GPU (no *frame buffer* e memória de textura) restringe a qualidade da imagem final. Na geração atual de GPUs, a memória de textura usa 64 bits para o armazenamento dos quatro canais de cor por *texel*, e o *frame buffer* usa de 32 a 48 bits por pixel. Esta precisão está longe da que pode ser obtida com algoritmos processados na CPU (128 bits de precisão por pixel). A última geração de GPUs suporta texturas com 128 bits de precisão, mas somente com interpolação por vizinho mais próximo (*nearest neighbor*) e sem operações de combinação (como a da Equação 3.2). Entretanto, considerando o ritmo no qual o hardware gráfico tem avançado nos últimos anos, estas limitações podem desaparecer em pouco tempo. A última geração de GPUs (equivalente à quinta geração), prevista para julho de 2004, possui precisão de 128 bits em todo *pipeline*, além de suportar múltiplas GPUs por CPU. Estas especificações, além do desempenho computacional superior ao da geração anterior, praticamente eliminam as limitações do sistema proposto.

Alguns dos problemas observados nos experimentos descritos no Capítulo 0 não exigem uma nova geração de GPUs para a sua solução. Um deles é a perda de precisão no armazenamento do VOI na GPU (ilustrado na Figura 5.9). A sugestão para a solução deste problema exige uma mudança na normalização dos valores de distância gravados na máscara de recorte: já que o problema está nos valores próximos ao limiar de recorte, estes valores devem ser alterados para que não fiquem mais tão próximos a ele. Isto leva à idéia de adotar um mapeamento não-linear das distâncias calculadas nos *shaders* para a máscara de recorte, ilustrada na Figura 6.1.

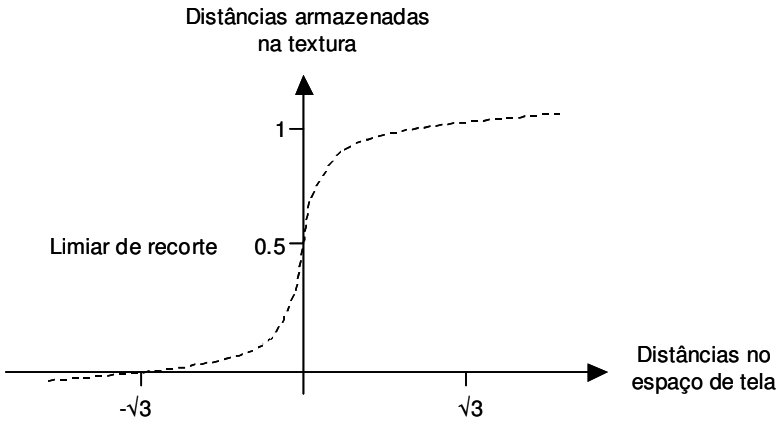


Figura 6.1: Mapeamento não-linear das distâncias calculadas nos *shaders* para a máscara de recorte.

6.2 Extensões

O sistema de visualização descrito nesta dissertação também serve como base para a implementação de outras ferramentas de interação. Este fato foi observado ao longo do estudo, na aplicação do sistema à visualização das imagens médicas. Estas ferramentas não fazem parte dos objetivos deste estudo, mas servem para ilustrar a eficiência do sistema proposto.

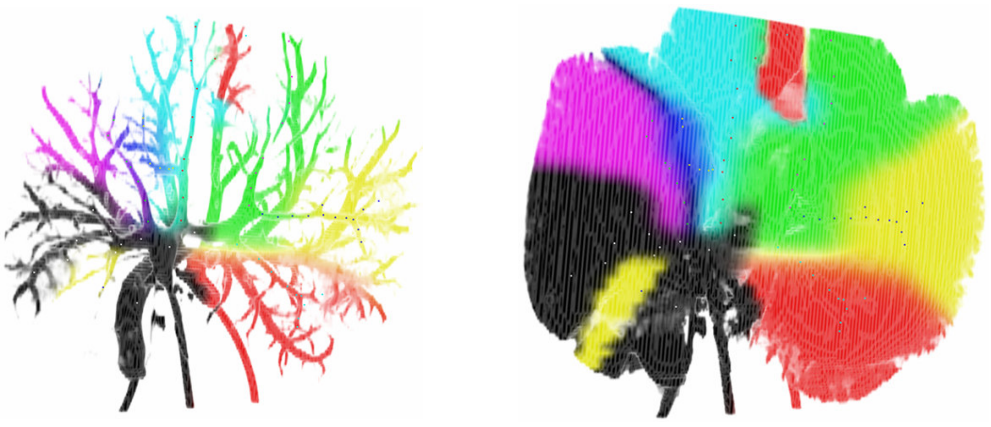
A primeira ferramenta implementada sobre o sistema permite a medição de distâncias entre pontos da imagem 3D. A medição é feita por meio de segmentos de reta, cujas extremidades são indicadas pelo usuário com o mouse. Estas extremidades correspondem a dois pontos *PR* (Seção 4.4). Por meio da textura de superfície (Seção 4.5.1), é possível recuperar a posição no espaço de textura (correspondente ao espaço de objeto) dos pontos. Contando com a posição no espaço das duas extremidades do segmento, o cálculo da distância é trivial. A Figura 6.2 mostra exemplos de medições feitas com segmentos de linha sobre o volume visualizado. Apesar de ser uma ferramenta simples, ela permite ao usuário estabelecer relações entre as partes do volume e correlacionar o modelo virtual ao modelo real.

Figura 6.2: Medições feitas com segmentos de linha sobre o volume visualizado. O volume foi adquirido de um fígado de porco *ex situ*, por TC contrastado.

A segunda ferramenta construída sobre o sistema permite ao usuário visualizar, na imagem 3D, a área de influência aproximada de cada vaso sanguíneo do fígado. Com esta informação, o usuário (médico) pode estimar a divisão entre os segmentos do fígado, e planejar futuras operações no órgão. Esta ferramenta assume que o parênquima do fígado é pré-segmentado, como é ilustrado na Figura 5.6 (c). O cálculo da região de influência de cada vaso sanguíneo sobre o parênquima é discutido em Aylward e Bullit (AYLWARD; BULLITT, 2002). Os autores classificam cada porção do parênquima em relação ao vaso sanguíneo mais próximo, considerando a distância Euclidiana. Em sua proposta, no entanto, os autores contam com a segmentação dos vasos sanguíneos e do parênquima. Esta característica não está presente no sistema proposto neste estudo.

Para lidar com esta limitação, foi construída uma interface que permite ao usuário informar para o sistema onde estão localizados os vasos sanguíneos. Por meio desta interface, o usuário pode indicar (com o mouse) pontos sobre a grade da imagem que correspondam à superfície dos vasos sanguíneos. Estes pontos correspondem a pontos PR. Para cada ponto informado, o sistema recupera TC_{PR} a partir da textura de superfície. A ferramenta usa então a capacidade de varredura da máscara de recorte (presente nos *pixel shaders* da borracha e da escavadeira virtuais) para calcular a distância de cada *texel* à TC_{PR}. Caso a distância calculada seja menor do que a armazenada no *texel*, o *pixel shader* armazena a informação de classificação nos canais de cor disponíveis do *texel* (até agora, a máscara de recorte utiliza apenas um canal de cor para armazenar a distância d). Nos testes da ferramenta, a informação de classificação usada é uma cor indicada pelo usuário, associada ao ponto PR. A Figura 6.3 mostra o resultado da aplicação da ferramenta de classificação a uma imagem TC de um fígado *ex-situ* de porco, onde o usuário aponta vários pontos de influência sobre cada vaso.

É importante observar dois pontos na proposta desta ferramenta. Primeiro, ela assume que o usuário possa modificar a função de transferência da imagem 3D. Apesar desta capacidade estar implementada no sistema, ela não faz parte dos objetivos deste estudo. Segundo, o cálculo da distância realizado nesta ferramenta é feito no espaço de textura. Como discutido na Seção 4.5.1, este tipo de abordagem pode resultar em comportamentos imprevisíveis do sistema.



(a) **(b)**

Figura 6.3: Resultado da aplicação da ferramenta de classificação sobre uma imagem TC de um fígado *ex-situ* de porco. O usuário indica vários pontos sobre os vasos sanguíneos (a) para estabelecer suas regiões de influência no parênquima (b) (no Apêndice D pode ser vista uma versão colorida desta figura).

REFERÊNCIAS

ABRASH, M. **Timer Function Performance**. nVIDIA Programming Resources. July 2001. Disponível em: < www.nvidia.com >. Acesso em: jan. 2004.

AYLWARD, S.; BULLITT, E. Initialization, noise, singularities, and scale in height ridge traversal for tubular object centerline extraction. **IEEE Transactions on Medical Imaging**, New York, v. 21, p. 61-75, Feb. 2002.

BENTLEY, J. T.; OTTMAN, Th. Algorithm for reporting and counting geometric intersections. **IEEE Transactions on Computers**, New York, v. 28, p. 643-647, 1979.

BLINN, J. Light reflection functions for simulations of clouds and dusty surfaces. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 21., 1994. **Proceedings…** New York: ACM, 1982. p. 21-29.

CABRAL, B.; CAM, N.; FORAN, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: IEEE/ACM VOLUME VISUALIZATION SYMPOSIUM, 1994. **Proceedings…** [S. l.; s. n.], 1994. p. 91-98.

CHANDRASEKHAR, S. **Radiative Transfer Theory**. New York: Dover, 1960.

CULLIP, T. J.; NEUMANN, U. **Accelerating volume reconstruction with 3D texture hardware**. Chapel Hill N.C.: University of North Carolina, 1993. (Technical Report TR93-027).

DREBIN, R. A.; CARPENTER, L.; HANRAHAN, P. Volume rendering. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 1994. **Proceedings…** New York: ACM, 1988. p. 65-74.

ENGEL, K.; KRAUS, M.; ERTL, T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In: ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, 2001. **Proceedings…** Los Angeles: ACM, 2001. p. 9-16.

ESPOSITO, L. W. Extensions to the classical calculation of the effect of mutual shadowing in diffuse reflection. **Icarus**, [S. l.], v. 39, p. 69-80, 1979.

GELDER, A. V.; KIM, K. Direct volume rendering with shading via three-dimensional textures. In: IEEE/ACM VOLUME VISUALIZATION SYMPOSIUM, 1996. **Proceedings…** [S. l.; s. n.], 1996. p. 23-30.

GOVINDARAJU, N.; REDON, S.; COLIN, M.; MANOCHA, D. CULLIDE: interactive collision between complex models in large environments using graphics hardware. In: ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, 2003. **Proceedings…** [S. l.]: ACM, 2002.

HARRIS M. J.; BAXTER III, W. V.; SCHEUERMANN T.; LASTRA A. Simulation of Cloud Dynamics on Graphics Hardware. In: ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, 2003. **Proceedings…** [S. l.]: ACM, 2003.

HERTEL, S.; MELHORN, K.; MÄNTYLÄ, M. J.; NIEVERGELT, J. Space sweep solves intersections of polyhedra elegantly. Helsinki: Laboratory of Information Processing, Helsinki University of Technology, 1983. (Report HTKK-TKO-B55).

IKITS, M.; KNISS, J.; LEFOHN, A.; HANSEN, C. Volume rendering techniques. In: FERNANDO, R. **GPU Gems**: programming techniques, tips, and tricks for real-time graphics. Boston: Addison-Wesley, 2004. p. 667-692.

KAJIYA, J. T.; VON HERZEN, B. Ray-tracing volume densities. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 1984. **Proceedings…** New York: ACM, 1994. p. 165-174.

KRÜGER, J.; WESTERMANN, R. Linear Algebra Operators for GPU Implementation of Numerical Algorithms. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 2003. **Proceedings…** New York: ACM, 2003.

LACROUTE, P.; LEVOY, M. Fast volume rendering using a shear-warp factorization of the viewing transformation. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 21., 1994. **Proceedings…** New York: ACM, 1994. p. 451-458.

LEVOY, M. Display of surfaces from volume data. **IEEE Computer Graphics and Applications**, [S. l.], v. 8, p. 29-37, May 1988.

LEVOY, M. Efficient ray-tracing of volume data. **ACM Transactions on Graphics**, [S. l.], v. 9, p. 245-261, July 1990.

MAX, N. Optical models for direct volume rendering. **IEEE Transactions on Visualization and Computer Graphics**, [S. l.], v. 1, June 1995.

NETTER, F. H. **Atlas Interativo de Anatomia Humana**. [S. l.], 1999. 1 CD-ROM.

NIEVERGELT, J.; PREPARATA, F. P. Plane-sweep algorithms for intersecting geometric figures. **Communications of the ACM,** [S. l.], v. 2, p. 739-747, Oct. 1982.

PFISTER, H. Real-time volume visualization with VolumePro. In: JAPANESE VISUALIZATION CONFERENCE, 5., 1999. **Proceedings…** Tokyo: [s. n.], 1999.

REZK-SALAMA, C.; ENGEL, K.; BAUER, M.; GREINER, G.; ERTL, T. Interactive volume on standard PC graphics hardware using multi-stage rasterization. In: ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, 2000. **Proceedings…** Interlaken: ACM, 2000. p. 109-118.

SABELLA, P. A rendering algorithm for visualizing 3D scalar fields. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 1988. **Proceedings…** New York: ACM, 1988. p. 51-58.

SHREINER, D. **OpenGL reference manual**: the official reference document to OpenGL, version 1.2. [S. l.]: Addison-Wesley, 1999.

TOMBROPOULOS, R. Z. Techniques and applications for three-dimensional visualization in medicine. In: BIOENGINEERING CONFERENCE, 1999. **Proceedings…** Big Sky: [s. n.], 1999.

UPSON, C.; KEELER, M. V-buffer: visible volume rendering. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 1988. **Proceedings…** New York: ACM, 1988. p. 59-64.

VOXEL-MAN 3D Navigator: Inner Organs. Germany: Springer-Verlag, 2000. 1 CD-ROM.

WEISKOPF, D.; ENGEL, K.; ERTL, T. Interactive clipping techniques for texture-based volume visualization and volume shading. **IEEE Transactions on Visualization and Computer Graphics**, [S. l.], July 2003.

WESTOVER, L. Footprint evaluation for volume rendering. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 1990. **Proceedings…** Dallas: [s. n.], 1990. p.367-376.

WITTENBRINK, C. M.; MALZBENDER, T.; GROSS, M. E. Opacity-weighted color interpolation for volume sampling. In: IEEE SYMPOSIOUM ON VOLUME VISUALIZATION, 1988. **Proceedings…** [S. l.: s. n.], 1988.

TEAM17 SOFTWARE LIMITED. **Worms 3D**. [S. l.], 2003. 1 CD-ROM.

ZHOU, J. **State of the art for volume rendering**. Magdeburg: Computer Vision Group, Institute for Simulation and Graphics, Otto-von-Guericke University of Magdeburg, 2003. Technical Reports.

APÊNDICE A *PIXEL SHADER* DA FERRAMENTA DE RECORTE "BORRACHA VIRTUAL"

```
struct vertex2fragment
{
    float2 TC_2D : TEXCOORD0;
};

void main(vertex2fragment v2f
    , uniform float4x4 ModelViewProj : C0
    , uniform float3 PR : C4
    , uniform sampler3D Mascara_3D : TEXUNIT0
    , uniform sampler2D Mascara_2D : TEXUNIT1
    , out float4 Fti : COLOR0 )
{
    float4 TC, Ft, SC;

    TC = tex2D(Mascara_2D, v2f.TC_2D);

    Ft = tex3D(Mascara_3D, TC.xyz);

    SC = mul(float4(TC.xyz - 0.5f, 1.0f), ModelViewProj);
    SC.xyz /= SC.w;

    Fti = float4((0.0f).xxx, length(SC.xy - PR.xy) * PR.z);

    if (Fti.a > Ft.a)
    {
        Fti = Ft;
    }
}
```

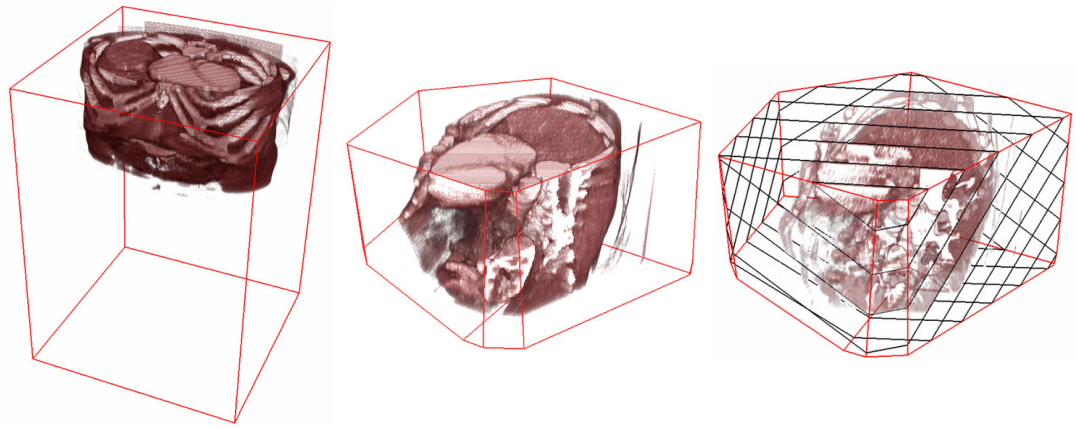
APÊNDICE B *PIXEL SHADER* 1 DA FERRAMENTA DE RECORTE "ESCAVADEIRA VIRTUAL"

```
#define SUPERFICIE_DO_VOI 0.5f

struct vertex2fragment
{
    float3 TC_3D  : TEXCOORD0;
};

void main(vertex2fragment v2f
    , uniform sampler3D Mascara_3D : TEXUNIT0
    , out float4 TP : COLOR0)
{
    TP = float4(v2f.TC_3D, tex3D(Mascara_3D, v2f.TC_3D).a);

    if (TP.a < SUPERFICIE_DO_VOI)
    {
        discard;
    }
}
```

APÊNDICE C *PIXEL SHADER* 2 DA FERRAMENTA DE RECORTE "ESCAVADEIRA VIRTUAL"

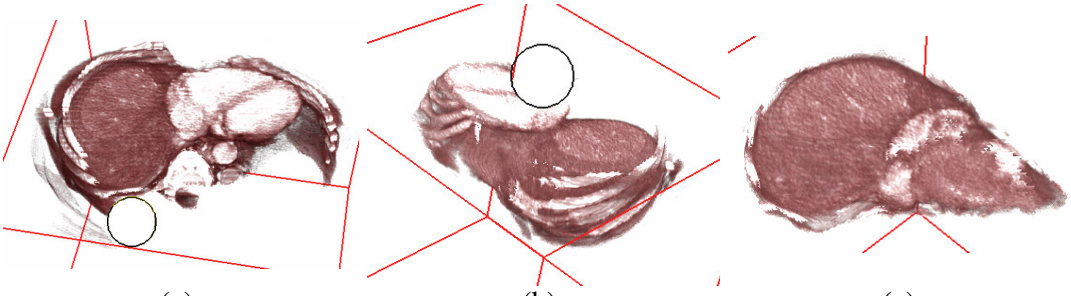
```
struct vertex2fragment
{
    float2 TC_2D : TEXCOORD0;
};

void main(vertex2fragment v2f
    , uniform float4x4 ModelViewProj : C0
    , uniform float4x4 ModelViewProj : C4
    , uniform float3 PR : C8
    , uniform sampler3D Mascara_3D : TEXUNIT0
    , uniform sampler2D Mascara_2D : TEXUNIT1
    , uniform sampler2D Superficie : TEXUNIT2
    , out float4 Fti : COLOR0 )
{
    float3 TC, TPpr, TCi;
    float4 Ft, SC;

    TC = tex2D(Mascara_2D, v2f.TC_2D).rgb;

    Ft = tex3D(Mascara_3D, TC);

    TPpr = tex2D(Superficie, (PR.xy + 1.0f) / 2.0f).xyz;

    TCi = TPpr + (ModelView._m00_m10_m20 * distance(TPpr, TC));

    SC = mul(float4(TCi - 0.5f, 1.0f), ModelViewProj);
    SC.xyz /= SC.w;

    Fti = float4((0.0f).xxx, distance(PR.xy, SC.xy) * PR.z);

    if (Fti.a > Ft.a)
    {
        Fti = Ft;
    }
}
```

APÊNDICE D VERSÕES COLORIDAS DAS FIGURAS DO TEXTO



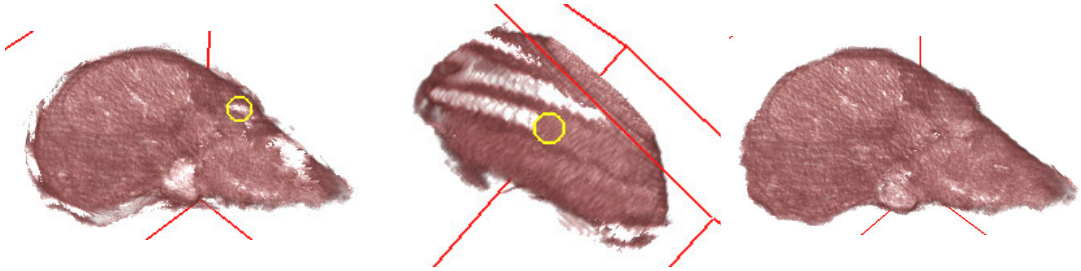
(a) (b) (c)

Figura 5-4: Recorte do VOI no estágio de atualização. (a) imagem e volume inicial (representado por suas linhas de contorno); (b) resultado de sucessivos recortes, definindo um VOI, ainda que bastante grosseiro; (c) polígonos de amostragem da textura (representados pelo seu contorno), ilustrando o recorte realizado pela WSG.



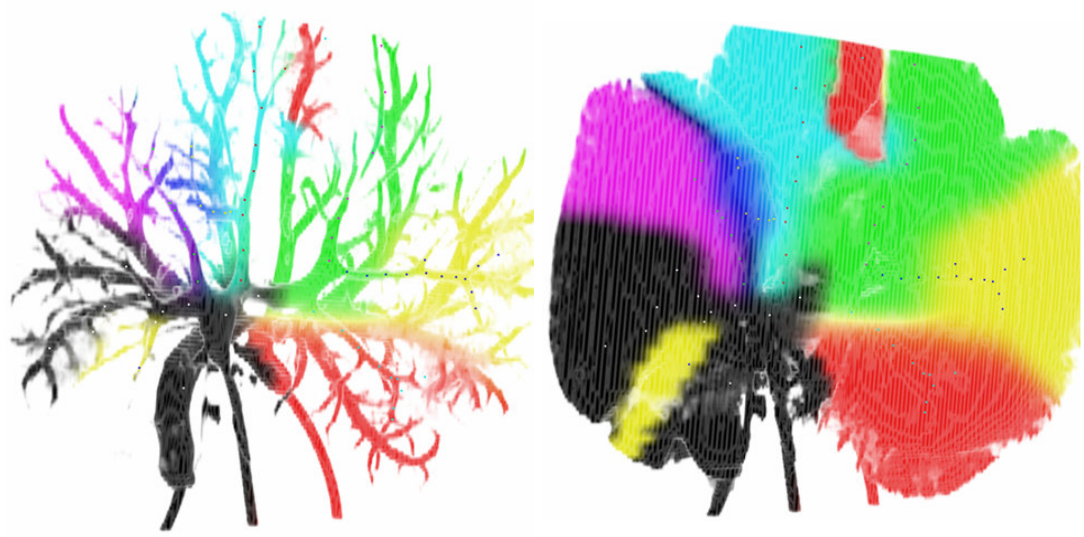
(a) (b) (c)

Figura 5-5: Aplicação da borracha virtual (a, b) em uma visualização 3D de um TC abdominal. O volume resultante mostrado em (c) corresponde ao parênquima do fígado; neste volume ainda podem ser observados artefatos (partes de costelas e tecidos contrastados) que não podem ser alcançados pela aplicação desta ferramenta, por estarem em reentrâncias do parênquima.



(a) (b) (c)

Figura 5-6: Aplicação da escavadeira virtual (a, b) sobre o VOI mostrado na Figura 5.5 (c) e o volume resultante (c).



(a) (b)

Figura 6-3: Resultado da aplicação da ferramenta de classificação sobre uma imagem TC de um fígado *ex-situ* de porco. O usuário indica vários pontos sobre os vasos sanguíneos (a) para estabelecer suas regiões de influência no parênquima (b).